

PanoramaStudio Viewer 4.77

Interaktive Panoramen, Virtuelle Touren und Zoom-Bilder auf Webseiten mittels
HTML5

Dokumentation

© 2005-2024, Tobias Hüllmandel Softwareentwicklung
<https://www.tshsoft.de>

Inhaltsverzeichnis

1	Der PanoramaStudio Viewer	1
1.1	Beschreibung und Systemvoraussetzungen	1
1.2	Features	1
1.2.1	Übersicht	1
1.2.2	Unterstützte Browser	1
1.3	Verwendung mit PanoramaStudio	2
1.4	Verwendung mit Panoramen aus anderer Software	2
1.5	Steuerung des PanoramaStudio Viewers	2
1.6	Kompatibilität mit alten XML-Parameterdateien	2
1.7	Lizenz	3
2	Erstellen der Bilddaten für den Viewer	6
3	Einbinden des Viewers in eine Webseite	6
3.1	Viewer direkt in HTML einbinden	6
3.2	Viewer mittels <iframe> einbinden	7
3.3	Viewer dynamisch hinzufügen und entfernen	7
3.4	Nötige Dateien bei der Verwendung des Viewers	8
4	Beschreibung der Parameter	9
4.1	Übersicht	9
4.2	Objekte	10
	actions	10
	audio	10
	autoplay	11
	button	11
	buttonBox	12
	camera	13
	contextmenu	14
	control	14
	data	15
	events	15
	gallery	16
	galleryElement	16

gfx	17
hotspot	17
hotspots	19
image	19
include	20
key	21
lensflare	21
location	21
logo	22
mapElement	22
node	23
orientation	24
settings	24
textbox	25
toolbar	26
translate	26
video	26
view	27
4.3 Universaleigenschaften	28
4.3.1 Eigenschaften aller Objekte	28
4.3.2 Eigenschaften aller Layer-Objekte	28
4.3.3 Eigenschaften von Button und Orientation	28
5 JavaScript API	30
5.1 Zugriff auf ein Panorama	30
5.2 Events	30
5.2.1 Übersicht	30
5.2.2 Events einer panoStudioViewer-Instanz	30
5.2.3 Lokale Events auf jedem Layer	32
5.3 Funktionen	33
5.3.1 Funktionen des globalen, statischen panoStudioViewer-Objekts	33
5.3.2 Funktionen der Viewer-Instanz	34
5.3.3 Lokale Funktionen von Layern	40
5.3.4 Lokale Funktionen von Buttons	42
5.3.5 Lokale Funktionen von Textboxen	42
5.3.6 Lokale Funktionen von Audio-Objekten	43
5.4 Eigenschaften	43

1 Der PanoramaStudio Viewer

1.1 Beschreibung und Systemvoraussetzungen

Der *PanoramaStudio Viewer* dient der interaktiven Darstellung von 3D-Panoramen, virtuellen Touren und großen 2D-Zoom-Bildern in Webseiten.

Beim *PanoramaStudio Viewer* handelt es sich um eine kleine, hochoptimierte HTML5-Anwendung, die in jedem HTML5-kompatiblen Webbrowser läuft.

Der Viewer kann dadurch plattformunabhängig auf fast allen Systemen praktisch beliebig große Bilder und Panoramen schnell und performant in Webseiten eingebettet anzeigen. Durch Verknüpfung von Panoramen mittels Hotspots lassen sich auch virtuelle Touren über mehrere Panoramen hinweg erstellen.

1.2 Features

1.2.1 Übersicht

- Interaktive und plattformunabhängige Präsentation von Panoramen und großen Bildern, sowohl lokal als auch im Internet
- Kleiner, hochperformanter HTML5 Betrachter in einer einzelnen JavaScript-Datei
- Zeigt partielle Panoramen, 360°x180°-Kugelpanoramen, Gigapixel-Panoramen, virtuelle Touren und 2D-Zoom-Bilder
- Schnelle Darstellung beinahe beliebig großer Bilder durch dynamisches Laden möglich
- Hochoptimierter JavaScript-Code für schnelle Darstellung
- Unterstützung sphärischer, zylindrischer und kubischer Panoramen
- VR-Modus via WebXR, WebVR oder emuliert mittels Split-Screen-Darstellung
- JavaScript-Schnittstelle zur Steuerung des Viewers
- Unterstützung von Hotspots und virtuellen Touren
- Kann für eine realistische Darstellung Lens Flares simulieren und Audio-Dateien für Geräusche und Musik abspielen
- Auto-Play für automatisches Schwenken, Neigen und Zoomen im Panorama möglich
- Nutzerinteraktion mittels Tastatur, Maus und Touch-Geräten
- Unterstützung von Bewegungssensoren zur Navigation in Panoramen
- Sehr umfangreiche Konfigurationsmöglichkeiten

1.2.2 Unterstützte Browser

Auf folgenden HTML5-fähigen Browsern und Geräten läuft der PanoramaStudio Viewer in der aktuellen Version:

- Safari iOS 4+
- Browser in 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 Verwendung mit PanoramaStudio

Der *PanoramaStudio Viewer* ist in die PanoramaStudio-Applikation bereits integriert. Einfache Webseiten mit einem interaktiven Panorama oder Zoom-Bild können somit direkt aus PanoramaStudio heraus automatisch und komfortabel gespeichert werden. Dabei generiert PanoramaStudio den nötigen HTML-Code und alle nötigen Dateien entsprechend den Nutzereinstellungen automatisch. Diese Ausgabe aus PanoramaStudio unterstützt bereits viele der in dieser Dokumentation beschriebenen Parameter und Möglichkeiten des Viewers.

Hinweis zur Erstellung eines Viewer-Panoramas

Für die Erstellung eines interaktiven Panoramas oder Zoom-Bildes sollte zunächst immer die Export-Funktion in der PanoramaStudio-Applikation verwendet werden. Diese erzeugt eine erste Variante aller nötigen Dateien automatisch.

Die in dieser Dokumentation beschriebenen Funktionen und Möglichkeiten des Viewers gehen allerdings weit über die in PanoramaStudio enthaltenen Konfigurationsmöglichkeiten hinaus.

Fortgeschrittenen Nutzern soll diese Dokumentation dazu dienen ein bestehendes Panorama bis ins Detail an eigene Vorstellungen anzupassen und ggf. auch die Integration in eine Webseite zu bearbeiten.

Alle Möglichkeiten, Parameter und Optionen zur individuellen Anpassung des Viewers werden dazu im Folgenden beschrieben.

1.4 Verwendung mit Panoramen aus anderer Software

Der Viewer kann prinzipiell alle zylindrischen und sphärischen Panoramen sowie Bilder, die mit anderer Software erzeugt wurden, verarbeiten und darstellen. Dazu können diese Bilder in PanoramaStudio mittels *Datei→Import/Export→Panoramabild importieren...* importiert und anschließend für den Viewer gespeichert werden.

1.5 Steuerung des PanoramaStudio Viewers

In einem Panorama, das mit dem *PanoramaStudio Viewer* dargestellt wird, kann sich der Nutzer interaktiv bewegen. Betrachter können die Ansicht mittels Maus- oder Touch-Eingabe schwenken und neigen. Zusätzlich wird das Ein-/Auszoomen mit dem Mausekranz unterstützt. Wird die Toolbar des Viewers eingeblendet, stehen dort weiterhin Buttons zum Navigieren und Zoomen, für ein Start und Stop des Auto-Play und weitere Funktionen zur Verfügung.

1.6 Kompatibilität mit alten XML-Parameterdateien

Der PanoramaStudio Viewer nutzt ab Version 4.0 JavaScript/JSON als Format für Parameterdateien. Die zuvor genutzten Parameterdateien auf Basis von XML werden jedoch weiterhin gelesen.

Einige Browser (z.B. Chrome) blockieren jedoch das Lesen von XML-Dateien mittels JavaScript im lokalen Dateisystem, so dass mit diesen Browsern XML-Parameterdateien nur über *http://* bzw. *https://* geladen werden können.

1.7 Lizenz

Lizenzbedingungen

PanoramaStudio Viewer - Version 4
für HTML5-kompatible Browser
Copyright © 2005-2022, Tobias Hüllmandel Softwareentwicklung
Internet: <http://www.tshsoft.de>
E-Mail: support@tshsoft.de

LIZENZVEREINBARUNG

1. LIZENZ: Der Programmautor (Tobias Hüllmandel Softwareentwicklung) stellt Ihnen die Software "PanoramaStudio Viewer 4", nachfolgend als "Software" bezeichnet, zur Verfügung, das außerdem eine elektronische Dokumentation und Lizenzbedingungen ("Lizenz.txt") umfasst und gewährt Ihnen eine Lizenz zur Verwendung des Produkts im Einklang mit den folgenden Bedingungen.

2. KOMMERZIELLE/PRIVATE NUTZUNG:

2.1 PRIVATE NUTZUNG

Die private/nicht-kommerzielle Nutzung der Software, auch im Internet, ist kostenlos und erfordert keine weitere Lizenz. Darüber hinaus benötigen gemeinnützige Vereine und akademische Einrichtungen ebenfalls keine weitere Lizenz.

2.2 GEWERBLICHE/KOMMERZIELLE NUTZUNG

Bei kommerzieller Nutzung der Software im Internet muss spätestens nach einer 30-tägigen Testphase eine gewerbliche Lizenz erworben werden.

Eine kommerzielle Nutzung im Internet liegt grundsätzlich vor, wenn der Betreiber der Internetpräsenz diese gewerblich nutzt, keine Privatperson, kein gemeinnütziger Verein oder keine akademische Einrichtung ist.

Folgende Lizenzen sind verfügbar:

SINGLE-DOMAIN-LIZENZ

Eine Single-Domain-Lizenz ist jeweils für eine Webseite und deren Domain gültig, sowie für evtl. vorhandene Alias-Domain-Namen, die auf die gleiche Internetpräsenz verweisen.

MULTI-DOMAIN-LIZENZ

Eine Multi-Domain-Lizenz kann auf einer beliebigen Anzahl von Webseiten eingesetzt werden, auch auf Webseiten von Dritten.

Die Anzahl der Panoramen, die mit der lizenzierten Software auf einer Webseite angezeigt wird, ist nicht beschränkt.

3. FOLGENDE HANDLUNGEN SIND NICHT ERLAUBT:

- 3.1 Verwenden der Software außer in der durch die Lizenz erlaubten Weise.
- 3.2 Das Vermieten, Leasen, Verleihen oder Unterlizenzieren ohne vorherige schriftliche Genehmigung.
- 3.3 Die kommerzielle Nutzung einer NICHT lizenzierten Kopie auf einer Internetpräsenz nach Ablauf des Testzeitraumes von 30 Tagen.
- 3.4 Im Rahmen obiger Lizenzen ist es nicht gestattet, die Software für eine Hosting-Plattform zu verwenden, auf der Dritte Inhalte erstellen können, die mit der Software angezeigt werden.
Bei Interesse an einer Hosting-Lizenz Anfragen bitte an support@tshsoft.de

3.5 Jegliche Form von Rückentwickeln, Disassemblieren oder Dekompilieren der Software.

3.6 Ändern der Software oder abgeleitete Werke erstellen, welche auf der Software basieren.

4. Der Programmautor ist unter keinen Umständen haftbar für Schäden einschließlich aller entgangenen Gewinne und Vermögensverluste oder anderer mittelbarer Schäden, die durch den Gebrauch oder die Nichtverwendbarkeit dieser Software entstehen. Es kann keine absolut korrekte Funktionsweise der Software garantiert werden.

Teile der Software basieren auf 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.

Lizenz erwerben

Entsprechend obiger Lizenzbestimmungen erfordert der *PanoramaStudio Viewer* für kommerzielle Nutzung im Internet den Kauf einer Lizenz spätestens nach einer 30-tägigen Testphase. Für eine solche kommerzielle Nutzung können Sie eine Lizenz beim Online-Vertrieb ShareIt der Digital River GmbH erwerben. Durch den Kauf erwerben Sie einen Lizenzschlüssel passend zum Domain-Namen Ihrer Internetpräsenz.

Mit dem Lizenzschlüssel erhalten Sie:

- das Recht den *PanoramaStudio Viewer* auf kommerziellen Webseiten einzusetzen. Die Zahl der Panoramen und der Einsatz des Viewers ist für die jeweilige Webseite nicht beschränkt.
- die Möglichkeit ein eigenes Logo in den Viewer einzubinden.
- kostenlose Hilfe bei Fragen und Problemen

Lizenzschlüssel bestellen

Eine Lizenz kann bequem und vor allem sicher per Internet beim Online-Vertrieb Share-It bestellt werden. Der Kaufpreis kann per Kreditkarte, PayPal, Überweisung, Scheck oder Bar bezahlt werden. Die Lieferung eines Lizenzschlüssels für die gewünschte Domain erfolgt innerhalb von 24 Stunden nach Zahlungseingang per E-Mail. Bei Kreditkartenzahlung erfolgt die Lieferung meist innerhalb von 30 bis 60 Sekunden. Bestellen können Sie per Internet: https://www.tshsoft.de/de/order_de

Kundenservice

Sie haben Fragen zu Bestellung, Zahlung oder Auslieferung? Antworten finden Sie hier im Bereich Kundenservice von Share-It:

<https://secure.shareit.com/shareit/ccc/index.html?publisherid=20959&languageid=2>

2 Erstellen der Bilddaten für den Viewer

Der *PanoramaStudio Viewer* kann Bilder in beinahe beliebiger Größe darstellen. Dazu ist allerdings eine Konvertierung der Bilddaten in ein optimiertes Format nötig, bei dem das Originalbild in kleine Bildkacheln mit verschiedenen Auflösungen zerlegt wird. Der Viewer kann daraus dynamisch sehr schnell die nötigen Bildausschnitte laden und anzeigen. Für diese Konvertierung der Bilddaten ist die Ausgabe für den Viewer in PanoramaStudio nötig. Bestehende Bilder können dazu in PanoramaStudio mittels *Datei→Import/Export→Bestehendes Panoramabild importieren...* geladen und daraufhin für den Viewer ausgegeben werden. PanoramaStudio erzeugt dann automatisch die optimalen Bilddaten für den Viewer.

3 Einbinden des Viewers in eine Webseite

3.1 Viewer direkt in HTML einbinden

Um den PanoramaStudio Viewer direkt in HTML einzubinden, ist es nötig das JavaScript des Viewers (`panoStudioViewer.js`) in der Seite zu laden.

Über die Funktion `panoStudioViewer.insert(...)` wird dann der Viewer einem DIV in der Seite zugewiesen, die Parameterdatei des Panoramas geladen sowie optionale, weitere Parameter für die Einbettung eines Panoramas übergeben:

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

Diese Funktion ist in Kap. 5.3.1 (→ S.33) beschrieben.

Beispiel:

Der folgende HTML-Code wird in ähnlicher Form auch von PanoramaStudio beim Erstellen eines Panoramas für den Viewer automatisch erzeugt, so dass man beim Übertragen in eine andere Webseite diesen auch einfach kopieren kann.

Eine detaillierte Beschreibung des Aufbaus der Parameterdatei (hier `MeinPano.js`) finden Sie in Kap. 4 (→ S.9).

Die `id viewerID` im `div`-Element ermöglicht es zudem das Panorama aus der Webseite heraus mittels JavaScript anzusprechen (Kap. 5, S.30).

```
<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", "MeinPano.js", {html5: "disable_webgl_warning"});
  </script>
</div>
```

Hier das gleiche Beispiel noch einmal fensterfüllend im Browser:

```
<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", "MeinPano.js", {html5: "disable_webgl_warning"});
  </script>
</div>
```

3.2 Viewer mittels <iframe> einbinden

Eine von PanoramaStudio erstellte HTML mit einem Panorama kann auch einfach unverändert mit einem <iframe>-Element in eine andere, bestehende HTML-Seite integriert werden mittels:

```
<iframe src="/Pfad/zur/Panorama.html" width="[wie gewünscht]" height="[wie gewünscht]"
  scrolling="no" marginheight="0" marginwidth="0" frameborder="0"
  allowfullscreen mozallowfullscreen webkitallowfullscreen"></iframe>
```

Das Panorama am Besten mit der Option *An Seitengröße angepasst* abspeichern, dann füllt es das iframe immer exakt aus.

Beispiel:

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

3.3 Viewer dynamisch hinzufügen und entfernen

Ähnlich wie in Kap. 3.1 (→ S.6) können Viewer-Objekte auch dynamisch auf Nutzerinteraktion in zu einer Seite hinzugefügt und wieder entfernt werden.

Dazu muss das Viewer-JavaScript in der Seite enthalten sein. Z.B.:

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

Weiterhin muss ein HTML-DIV-Element als Container für das Viewer-Objekt in der Seite vorhanden sein. Z.B.:

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

Auf eine Nutzerinteraktion kann dann ein Panorama-Objekt dynamisch hinzugefügt werden. Etwa mittels:

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

Ebenso kann das Panorama wieder entfernt werden:

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

3.4 Nötige Dateien bei der Verwendung des Viewers

PanoramaStudio erzeugt beim Speichern eines interaktiven Panoramas/Zoom-Bildes für den Viewer bereits alle nötigen Dateien. Beim Kopieren oder Hochladen auf einen Webserver ist dabei immer darauf zu achten, dass jeweils alle erstellten Dateien weitergegeben werden.

Eine Webseite mit einem interaktiven Panorama/Zoom-Bild besteht neben der HTML-Datei noch aus weiteren Dateien. Das sind grundsätzlich der PanoramaStudio Viewer als JavaScript `panoStudioViewer.js`, ein Ordner mit den Bilddaten und eine oder mehrere Parameter-Dateien (`.js` oder `.json`), die alle Einstellungen und Parameter des Panoramas enthält.

Wenn möglich, sollten auch beim Weiterverarbeiten alle Dateien im gleichen Verzeichnis verbleiben, ansonsten müssen die entsprechenden Pfade im HTML-Code bzw. in den Parameter-Dateien angepasst werden.

4 Beschreibung der Parameter

4.1 Übersicht

Alle Parameter zur Beschreibung der Eigenschaften und der Darstellung des Panoramas werden in einer Parameter-Datei zusammengefasst, welche im Grunde aus einem JSON-Objekt (*JavaScript Object Notation*) besteht. Siehe auch

https://de.wikipedia.org/wiki/JavaScript_Object_Notation).

Der Aufbau der Parameter-Datei besteht dabei aus einer Zuweisung des JSON-Objekts zur globalen Variable `PanoramaStudioViewerParams`. Darin dann ein Objekt `panoramaStudioViewer`, welches schließlich alle Objekte zur Beschreibung des Panoramas enthält.

Zur besseren Übersichtlichkeit kann PanoramaStudio die Datei als .json-Datei ausgeben. Streng genommen handelt es sich aufgrund der Zuweisung jedoch um JavaScript. Für eine bessere Kompatibilität auf diversen, aktuellen Webservern sollten die Parameterdateien daher als .js erstellt werden.

Parameterdateien können Funktionen enthalten ([Function] im folgenden). Entsprechend dem deklarierten Datentyp der Parameterdatei, JSON oder JS, müssen Funktionen entweder als String (JSON) oder können direkt als JavaScript eingebettet werden.

Damit ein Funktion JSON genügt, muss die Funktion als String in folgender Form vorliegen:

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

Wenn die Parameterdateien hingegen als JavaScript deklariert werden, können Funktionen auch direkt eingebettet werden:

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

Unterstützte Objekte:

```
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": [...],
    "settings": [...],
    "textbox": [...],
    "toolbar": [...],
    "translate": [...],
    "video": [...],
    "view": [...]
  }
}

```

4.2 Objekte

"actions":

Array von Objekten, die jeweils eine Funktion (`func`) und eine ID (`id`) enthalten. Auf diese Weise können Funktionen, die von verschiedenen Objekten genutzt werden zentral abgelegt werden. Die Funktionen können mit der `action()`-Funktion aufgerufen werden (Kap. 5.3.2 (→ S.35)).

Eigenschaften eines Array-Elements:

- **func** - [Function] Eine Funktion mit einer beliebigen Anzahl von Argumenten.
- **viewer** - [Object], Zugriff auf die Viewer-Instanz Kap. 5.1 (→ S.30).
- von Objekt übernommene Eigenschaften: Kap. 4.3.1 (→ S.28)
- jedes Array-Element stellt die Funktionen `get()` und `tween()` bereit, wie in Kap. 5.3.3 (→ S.40) definiert.

Beispiel:

```

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

```

"audio":

Einbinden einer Audiodatei.

Eigenschaften:

- **src** - [String], URL auf eine MP3-Datei
- **src_alt** - [String], optionale URL auf eine OGG-Datei, als alternative Audio-Quelle für Browser, die kein MP3 unterstützen.
- **loop** - [Boolean], Audio-Datei in einer Endlosschleife abspielen (default: true)

- **autoplay** - [Boolean], Audio-Datei automatisch abspielen, wenn geladen (default: true)
- **volume** - [Number], Lautstärke (default: 1.0, min: 0.0, max: 1.0)
- **priority** - [Number], Priorität des Audio-Elements. Gibt es ein globales und ein lokales Audio-Element, dann wird das Element mit höherer Priorität gespielt, bei gleicher Priorität das lokale Element (default: 0)
- von Objekt übernommene Eigenschaften: Kap. 4.3.1 (→ S.28)

Beispiel:

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

"autoplay":

Parameter, die das Auto-Play betreffen, werden innerhalb eines autoplay-Objekts konfiguriert.

Eigenschaften:

- **status** - [String], mögliche Werte: 'on' oder 'off' für Auto-Play an bzw. aus.
- **reference** - [String], definiert die Einheit für die Werte in den Attributen **pan** und **tilt**. Zulässige Werte sind 'pano' und 'screen'. Bei 'pano' bewegen pan und tilt das Bild in Grad/Sekunde, bei 'screen' wird das Bild in Anteilen der Screenbreite/Sekunde bewegt.
- **pan** - [Number], bei Panoramen, horizontale Schwenkgeschwindigkeit (Werte <0 nach links, Werte >0 nach rechts). Einheit definiert in 'reference'.
- **tilt** - [Number], bei Panoramen, vertikale Neigungsgeschwindigkeit (Werte <0 nach oben, Werte >0 nach unten). Einheit definiert in 'reference'.
- **x** - [Number], bei Zoom-Bildern, horizontale Scrollgeschwindigkeit in Anteilen der Bildbreite/Sekunde (Werte <0 nach links, Werte >0 nach rechts).
- **y** - [Number], bei Zoom-Bildern, vertikale Scrollgeschwindigkeit in Anteilen der Bildhöhe/Sekunde (Werte <0 nach links, Werte >0 nach unten).
- **zoom** - [Number], Ein-/Auszoomen. Werte kleiner 1.0 führen zu einem Einzoomen, Werte größer 1.0 zu einem Auszoomen.
- **restart** - [Number], Neustart des Auto-Play nach Unterbrechung durch Nutzerinteraktion in Sekunden (default: 5.0, min: 0, max: 1000)
- **returnToLevel** - [Boolean], schwenkt bei horizontalem Auto-Play langsam wieder auf die ursprüngliche Höhe, wenn durch den Betrachter zuvor nach oben oder unten geschwenkt wurde (default: true)

"button":

Programmierbare Schaltfläche für Navigation und andere Funktionen.

Eigenschaften:

- **width** - [Number], Breite in Pixel.
- **height** - [Number], Höhe in Pixel.
- **type** - [Number], Vordefiniertes Verhalten bei gedrücktem Button für einige Funktionen (default: 0, kein vordefiniertes Verhalten). Mögliche Werte:
 - 1: Nach links schwenken
 - 2: Nach rechts schwenken
 - 3: Nach oben schwenken
 - 4: Nach unten schwenken
 - 5: Einzoomen
 - 6: Auszoomen
- **priority** - [Number], Dient dem *responsive* Design, wenn der Button in einer `buttonBox` enthalten ist. Wenn auf kleinen Bildschirmen der Platz nicht für alle Buttons der `buttonBox` ausreicht, werden Buttons mit geringerer Priorität ausgeblendet (z.B. die Richtungsschaltflächen). Geringste Priorität ist 0, die höchste vom PanoramaStudio-Export verwendete ist 3.
- **index** - [Number], Dient der Anordnung, wenn der Button in einer `buttonBox` enthalten ist. Buttons werden nach aufsteigendem `index` von links nach rechts angeordnet.
- **style** - [String], Entspricht dem CSS-Style-Attribut in HTML. Das Objekt kann auf diese Weise mittels CSS gestaltet werden.
- **stylehover** - [String], CSS-Style, der bei Maus-Hover über dem button angewendet wird. Überlagert evtl. in `style` definierte CSS-Anweisungen.
- **styleactive** - [String], CSS-Style, der bei gedrückter Maustaste über dem button angewendet wird. Überlagert evtl. in `style` und `stylehover` definierte CSS-Anweisungen.
- **skin** - [String], Definiert das Aussehen des Buttons. Siehe Kap. 4.3.3 (→ S.28).
- **skinhover** - [String], Definiert das Aussehen des Buttons, wenn sich der Mauszeiger über dem Button befindet. Siehe `skin`.
- **skinactive** - [String], Definiert das Aussehen des Buttons, wenn die Maustaste über dem Button gedrückt wird. Siehe `skin`.
- von Layer übernommene Eigenschaften: Kap. 4.3.2 (→ S.28)
- von Layer übernommene Events: Kap. 5.2.3 (→ S.32)

"buttonBox":

Stellt einen Container für Buttons dar. Es können u.a. einige `button`-Eigenschaften vorgegeben werden, die auf alle darin enthaltenen Buttons angewendet werden.

Eigenschaften:

- **width** - [String], Breite in Prozent (%) oder Pixel (px). Beispiel:

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

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

- **height** - [String], Höhe in Prozent (%) oder Pixel (px).
- **buttonWidth** - [Number], Breite der enthaltenen Buttons in Pixel. Kann von jedem Button individuell überschrieben werden.
- **buttonHeight** - [Number], Höhe der enthaltenen Buttons in Pixel.
- **spacing** - [Number], Abstand der enthaltenen Buttons in Pixel (default: 8).
- **style** - [String], Entspricht dem CSS-Style-Attribut in HTML. Das Objekt kann auf diese Weise mittels CSS gestaltet werden.
- **stylehover** - [String], CSS-Style, der bei Maus-Hover über der `buttonBox` angewendet wird. Überlagert evtl. in `style` definierte CSS-Anweisungen.
- **styleactive** - [String], CSS-Style, der bei gedrückter Maustaste über der `buttonBox` angewendet wird. Überlagert evtl. in `style` und `stylehover` definierte CSS-Anweisungen.
- **buttonstyle** - [String], `style`-Eigenschaft, die auf alle enthaltenen Buttons angewendet wird. Siehe `button`.
- **buttonstylehover** - [String], `stylehover`-Eigenschaft, die auf alle enthaltenen Buttons angewendet wird.
- **buttonstyleactive** - [String], `styleactive`-Eigenschaft, die auf alle enthaltenen Buttons angewendet wird.
- **button** - [Array], Array von `button`-Objekten, die in der `buttonBox` enthalten sind. Siehe Kap. 4.2 (→ S.11).
Enthält ein Eintrag das Attribut `"type": "compass"` oder `"type": "radar"` wird ein `orientation`-Elemente in die Button-Box eingefügt. Siehe auch Kap. 4.2 (→ S.24).
- von Layer übernommene Eigenschaften: Kap. 4.3.2 (→ S.28)

"camera":

Sichtfeld-Eigenschaften bei 3D-Panoramen. Hier wird der minimale und maximal mögliche Zoom eingestellt und bei partiellen Panoramen das vom Panorama erfasste Sichtfeld definiert.

Eigenschaften:

- **minpan** - [Number], Minimale Schwenkrichtung. Durch diesen Wert wird der linke Rand des Panoramas definiert (default: 0)
- **maxpan** - [Number], Maximale Schwenkrichtung. Durch diesen Wert wird der rechte Rand des Panoramas definiert (default: 360)
- **mintilt** - [Number], Minimale Neigungsrichtung. Durch diesen Wert wird der obere Rand des Panoramas definiert (default: -90)
- **maxtilt** - [Number], Maximale Neigungsrichtung. Durch diesen Wert wird der untere Rand des Panoramas definiert (default: 90)
- **minhfov** - [Number], Minimales, horizontales Sichtfeld. Durch diesen Wert wird das Einzoomen in das Bild begrenzt (default: 0.000000001)
- **maxhfov** - [Number], Maximales, horizontales Sichtfeld. Durch diesen Wert wird das Auszoomen aus dem Bild begrenzt (default: 140)
- **maxzoom** - [Number], Maximal mögliche Vergrößerung der Pixel beim Einzoomen (alternativ zum Parameter `minhfov`) (default: 10)

"contextmenu":

Konfigurierbares Kontextmenü.

Eigenschaften und Funktionen:

- **highlightColor** - [String] Hintergrundfarbe des aktiven Menu-Eintrags. Farbangabe im CSS-Format, z.B. blue, #ff0000 oder rgb(255,0,0).
- **highlightTextColor** - [String] Textfarbe des aktiven Menu-Eintrags. Farbangabe im CSS-Format, z.B. blue, #ff0000 oder rgb(255,0,0).
- **style** - Entspricht dem CSS-Style-Attribut in HTML. Das Kontextmenü kann auf diese Weise mittels CSS gestaltet werden.
- **viewer** - [Object], Zugriff auf die Viewer-Instanz Kap. 5.1 (→ S.30).
- **items** - [Array] Array der Menü-Items. Eigenschaften der Items siehe unten.
- **update()** - [Function] Sichtbarkeit und Titel der Items im Kontextmenü aktualisieren.
- **/*Object*/ getItem(/*String*/ id)** - [Function] Menü-Item aus items anhand seiner id finden.
- **/*Object*/ get(/*String*/ id)** - [Function] Ein Objekt anhand seiner id finden.
- von Layer übernommene Events: Kap. 5.2.3 (→ S.32)

Eigenschaften der Kontext-Menu-Items:

- **caption** - [String], Beschriftung des Items.
- **id** - [String], ID des Items.
- **visible** - [Boolean], Sichtbarkeit des Items.
- **viewer** - [Object], Zugriff auf die Viewer-Instanz Kap. 5.1 (→ S.30).
- **oninit()** - [Function], Wird beim Initialisieren des Items aufgerufen.
- **onclick()** - [Function], Wird beim Klick auf das Item aufgerufen.

"control":

Dient der Konfiguration der Touch- und Maussteuerung sowie dem Erfassen von Events auf dem Panorama-Objekt.

Eigenschaften:

- **mousemode** - [String], Art der Steuerung bei Mausbedienung. Mögliche Werte: 'move', die Kamera schwenkt in die Richtung, in die die Maus gezogen wird. 'drag' das Bild/Panorama kann mit dem Mauszeiger gegriffen und verschoben werden.
- **touchmode** - [String], Art der Steuerung bei Touchbedienung. Mögliche Werte: 'move', die Kamera schwenkt in die Richtung, in die der Finger bewegt wird. 'drag' das Bild/Panorama kann mit dem Finger gegriffen und verschoben werden.
- **zoomtocursor** - [Boolean], Mausposition als Mittelpunkt beim Ein- und Auszoomen verwenden (default: true)

- `mwheel` - [Boolean], Zoomen per Mousrad erlauben (default: true)
- Events auf dem globalen Panorama-Layer werden dem `control`-Objekt gesendet. Folgende Events werden unterstützt: Kap. 5.2.3 (→ S.32)
- `propagatela`st - [Boolean], Standardmäßig werden Maus-Events zunächst dem globalen Panorama-Layer gesendet, erst dann allen darüberliegenden Objekten beginnend beim obersten. Setze dieses Attribut auf true, um die Events zuerst dem Objekt im Vordergrund zu senden (default: false)
- `enablevrcontroller` - [Boolean], Nutzt bei aktivem VR-Modus einen Controller bzw. ein Gamepad, falls vorhanden. (default: true)
- `vrdisplay` - [String], Art des VR-Displays. Mögliche Werte: 'auto', nutzt WebXR oder das ältere WebVR, wenn vom Browser unterstützt, ansonsten erfolgt die VR-Darstellung automatisch Emulation mittels Split-Screen-Darstellung, etwa für Smartphones, die mit Google Cardboard genutzt werden. 'splitscreen', immer eine emulierte Split-Screen-Darstellung nutzen, kein WebXR oder WebVR. (default: auto)

"data":

Im `data`-Objekt können beliebige Key-Value-Paare, auch dynamisch, gespeichert werden. So können Daten an einer Stelle zentral abgelegt werden und von verschiedenen, anderen Objekten kann darauf zugegriffen werden.

Eigenschaften:

- beliebige Key-Value-Paare im JSON-Format.
- von Objekt übernommene Eigenschaften: Kap. 4.3.1 (→ S.28)

Beispiel:

```
"data": {  
  "id": "globalData",  
  "messageText": "Hello, World!"  
}  
  
//Zugriff auf die Daten innerhalb einer Funktion:  
console.log(this.viewer.get('globalData').messageText);
```

"events":

Im `events`-Objekt können einer Reihe von Ereignissen Funktionen zugewiesen werden. Auf diese Weise können eigene Funktionen bei bestimmten Ereignissen ausgeführt werden.

Eigenschaften:

- [Events] - Allen Ereignissen (Events), die in Kap. 5.2.2 (→ S.30) definiert sind, können hier Funktionen zugewiesen werden.
- von Objekt übernommene Eigenschaften: Kap. 4.3.1 (→ S.28)

Beispiel:

```
"events": {
  "id": "mainEvents",
  "onplay": function(){ console.log('Auto-Play gestartet'); }
}
```

"gallery":

Das gallery-Objekt ist ein Array mit Einträgen für eine Galerie aus Miniaturbildern, die jeweils auf andere Panoramen oder andere Ansichten im gleichen Panorama verweisen.

Das galleryElement-Objekt kann diese Galerie darstellen.

Ein gallery-Objekt kann auch genutzt werden, um die Positionen der Panoramen mit einem mapElement auf einer Karte darzustellen.

Eigenschaften und Funktionen:

- `/*int*/ current()` - [Function] Liefert den Index des aktuell selektierten Galerie-Eintrags.
- `/*int*/ previous()` - [Function] Liefert den Index des Galerie-Eintrags zur Linken. Liefert den Index des Eintrags ganz rechts, wenn der aktuelle Index 0 ist.
- `/*int*/ next()` - [Function] Liefert den Index des Galerie-Eintrags zur Rechten. Liefert 0, wenn der aktuelle Index bereits der bereits ganz rechts ist.
- `set(/*int*/ i)` - [Function] Selektiert den Eintrag mit Index i.
- `viewer` - [Object], Zugriff auf die Viewer-Instanz Kap. 5.1 (→ S.30).
- `/*Object*/ get(/*String*/ id)` - [Function] Ein Objekt anhand seiner id finden.

Eigenschaften eines Array-Elements:

- `href` - [String] Url der Parameter-Datei des Ziel-Panoramas (.js oder .json).
- `thumbnail` - [String] Url eines JPG Vorschau-Bildes.
- `view` - [Object] Optionales view-Objekt, um die initiale Ansicht im Ziel zu definieren. Siehe Kap. 4.2 (→ S.27).
- `location` - [Object] Optionales location-Objekt, um Standortdaten des Ziels zu hinterlegen. Siehe Kap. 4.2 (→ S.21).
- `text` - [String] HTML-formatierter Text, der als Beschreibung auf der Vorschau gezeigt wird.
- `tooltip` - [Object] Ein textbox-Objekt als Tooltip über dem Galerie-Eintrag an (siehe textbox-Objekt). Lediglich die Position des Tooltips wird nicht durch textbox-Eigenschaften vorgegeben, sondern folgt immer der Maus über dem Galerie-Eintrag.

"galleryElement":

Ein galleryElement ist eine Miniaturbild-Übersicht bzw. Galerie, die Verknüpfungen auf verschiedene Ansichten im gleichen Panorama oder weitere Panoramen enthalten kann.

Es kann ein freistehendes Element im Viewer oder Teil eines buttonBox-Elements sein. Ein galleryElement visualisiert den Inhalt eines gallery-Objekts.

Eigenschaften:

- `thumbnailWidth` - [int] Breite der Miniaturbilder (default: 96).
- `thumbnailHeight` - [int] Höhe der Miniaturbilder (default: 96).
- `thumbnailPadding` - [int] Padding der Miniaturbilder in Pixel (default: 8).
- `framewidth` - [int] Breite des Rahmens um ein selektiertes Miniaturbild (default: 3).
- `selectioncolor` - [String] Farbe des Rahmens ein selektiertes Miniaturbild (default: #fff).
- `selectionstyle` - [String] Entspricht dem CSS-Style-Attribut in HTML. Der Rahmen um den selektierten Eintrag kann auf diese Weise per CSS gestylt werden.
- `collapsed` - [Boolean] Status des `galleryElement`. Falls `true`, dann ist die Galerie ausgeblendet.
- `style` - [String] Konfiguriert den CSS-Style des `galleryElement`, zum Beispiel Rahmen oder Hintergrundfarbe.
- `thumbstyle` - [String] Konfiguriert den CSS-Style der Miniaturbilder in der Galerie.
- `thumbstylehover` - [String] Analog `thumbstyle`. Wird angewendet, wenn sich der Mauszeiger über dem Miniaturbild befindet.
- `thumbstyleactive` - [String] Analog `thumbstyle`. Wird angewendet, wenn die Maustaste über dem Miniaturbild gedrückt wird.
- `contentstyle` - [String] Konfiguriert den CSS-Style des Inhalts über den Miniaturbildern in der Galerie. Der Inhalt wird jeweils im `text`-Attribut der Galerie-Einträge definiert.
- `contentstylehover` - [String] Analog `contentstyle`. Wird angewendet, wenn sich der Mauszeiger über dem Miniaturbild befindet.
- `contentstyleactive` - [String] Analog `contentstyle`. Wird angewendet, wenn die Maustaste über dem Miniaturbild gedrückt wird.
- von Layer übernommene Eigenschaften: Kap. 4.3.2 (→ S.28)
- von Layer übernommene Events: Kap. 5.2.3 (→ S.32)

"gfx":

Ermöglicht das Laden von Bitmap-Bildern, die dann zur Gestaltung von Buttons zur Verfügung stehen.

Eigenschaften:

- `src` - [String], Url des Bildes.
- von Objekt übernommene Eigenschaften: Kap. 4.3.1 (→ S.28)

"hotspot":

Ein Hotspot im Panorama/Zoom-Bild

Eigenschaften:

- **anchor** - [Number,Number], Bei schwebenden Hotspots die Pixelkoordinaten im Hotspot-Bild, an denen der Hotspot im Panorama verankert werden soll. [0,0] ist die linke obere Ecke des Bildes. Standardmäßig ist der Anker die Mitte des Hotspot-Bildes.
- **bitmap** - [String], URL auf eine Bitmap-Datei, die als Bitmap-Hotspot dient (JPG, GIF, PNG, WEBP).
- **bitmaptype** - [String], Darstellungsart des Bitmap-Hotspots. Mögliche Werte sind: `floating`, die Grafik wird schwebend über der Position im Panorama dargestellt oder `embedded`, die Grafik wird räumlich fest verbunden in das Panorama eingebettet (default: `floating`).
- **bitmapscale** - [Number;Number(optional)], Skalierung eines schwebenden Bitmap-Hotspots im Normalzustand und beim Berühren mit der Maus, z.B. `'1.0;1.2'`, um den Hotspot bei Berührung mit der Maus 20% größer darzustellen.
- **displaytext** - [String], Darstellungstyp eines evtl. vorhandenen Tooltips `'permanent'` zeigt den Tooltip immer an, `'onhoverstatic'` zeigt den Tooltip beim Überfahren des Hotspots an `position` an, `'onhoverfollow'` zeigt den Hotspot beim Überfahren mit der Maus am Mauszeiger an.
- **embeddedwidth** - [Number], Breite eines eingebetteten Bitmap-Hotspots (siehe auch `unit`)
- **href** - [String], Ziel-URL. Das kann eine Webseite oder ein weiteres Panorama durch einen Verweis auf die Parameterdatei (`.js` oder `.json`) sein (siehe `target`)
- **opacity** - [Number;Number(optional)], Deckkraft eines Hotspots im Normalzustand und beim Berühren mit der Maus in Werten von 0 (unsichtbar) bis 1 (undurchsichtig), z.B. `'0.5;1.0'`, um den Hotspot bei Berührung mit der Maus von halbtransparent auf komplett undurchsichtig umzuschalten.
- **position** - [Number,Number], Position eines Bitmap-Hotspots (siehe auch `unit`).
- **rotate** - [Number], Drehung eines eingebetteten Bitmap-Hotspots in Grad.
- **target** - [String], HTML-Target, beim Keyword `myself` wird als href ein Verweis auf eine Parameterdatei (`.js` oder `.json`) eines weiteren Panoramas erwartet, den der Viewer dann im gleichen Viewerfenster öffnet
- **targetview** - [String], Befehlskette, mit der eine initiale Ansicht beim Wechsel in ein weiteres Panorama vorgegeben werden kann (siehe dazu auch `cmd` in `openPanorama` in Kap. 5.3.2 (→ S.38))
- **transition** - [String], Befehlskette, mit der ein Übergangseffekt beim Wechsel in ein weiteres Panorama vorgegeben werden kann. Beschreibung durch Beispiel: `'zoomin,1.0 ; blend,2.0'` Dabei führt `zoomin,1.0` zu einem Zoom auf den angeklickten Hotspot mit Dauer von 1,0 Sekunden und einem anschließenden Überblenden ins neue Panorama (`blend`) mit einer Dauer von 2,0 Sekunden.
- **unit** - [String], Einheit der Koordinaten im `position`-Attribut. Mögliche Werte: `deg`, Angabe der Werte in Grad, `norm` Angabe der Werte normiert auf Werte von 0 bis 1. Für Panoramen ist nur die Angabe in `deg` möglich, für planare Zoom-Bilder nur die Angabe in `norm`.
- von Layer übernommene Eigenschaften: Kap. 4.3.2 (→ S.28)
- von Layer übernommene Events: Kap. 5.2.3 (→ S.32)

Enthaltene Objekte:

- **textbox** - Zeigt ein `textbox`-Element als Tooltip zum Hotspot an (siehe `"textbox"`-Element). Lediglich die Position des Tooltips wird nicht durch `textbox`-Eigenschaften vorgegeben, sondern in diesem Fall durch das `displaytext`-Attribut im Hotspot.
- **animation** - Spezifiziert einen animierten Hotspot. Die Einzelbilder der Animation werden als Streifen bzw. Matrix im Bild des `bitmap`-Attributs des Hotspots hinterlegt. Die Einzelbilder (Frames) müssen darin in Zeilen von links nach rechts und von oben nach unten angeordnet sein.

- **width** - [int], Breite der Frames in Pixel.
- **height** - [int], Höhe der Frames in Pixel.
- **frames** - [int], Gesamtzahl der Frames.
- **delay** - [Number], Verzögerung in Sekunden bis die Animation nach dem Laden der Szene startet.
- **iterations** - [int], Spezifiziert wie oft die Animation abgespielt wird. 0 für unbegrenzte Anzahl von Wiederholungen.
- **frameOffset** - [int], Ersten Bild der Animation. Die Animation stoppt auch an diesem Bild nach einem **mouseover** oder nachdem eine begrenzte Anzahl Wiederholungen (**iterations**) abgespielt wurden.
- **duration** - [Number], Zeit in Sekunden, um die Animation einmal abzuspielen.
- **mouseover** - [Boolean], Die Animation wird nur abgespielt, während sich der Mauszeiger über dem Hotspot befindet.

"hotspots":

Eigenschaften der Hotspots

Eigenschaften:

- **visible** - [Boolean], bei **true** sind Hotspots immer sichtbar, ansonsten nur, wenn der Mauszeiger sich über einem Hotspot befindet (default: **true**)

"image":

Beschreibung der Bilddaten. Das **image**-Objekt und die Bilddaten sollten automatisch von PanoramaStudio erzeugt und nicht manuell bearbeitet werden.

Eigenschaften:

- **projection** - [String], Projektionstyp des Bildes
mögliche Werte: **'cubic'**, **'planar'**
'planar' bedeutet, dass das Panorama als flaches Zoom-Bild dargestellt wird. **'cubic'** zeigen das Bild entsprechend perspektivisch als 3D-Panorama.
- **multilevel** - [Boolean], **true**, falls mehrere Auflösungsstufen vorliegen, sonst **false**
- **baseindex** - [Number], Basisindex bei der Nummerierung der Bildkacheln
- **resolutionadjust** - [Number], legt fest, bei welchem Zoom von einer Auflösungsstufe zur nächsten umgeschaltet wird. Mögliche Werte sind 0.5 bis 1.0, wobei 0.5 früh zur nächsthöheren Auflösung wechselt und ein schärferes Bild zeigt, jedoch mehr Ressourcen und Bandbreite für die Bilddaten benötigt. (default: 0.82, min: 0.5, max: 1.0)
- **mipmapping** - [Boolean], Mip-Mapping verwenden um die Darstellungsqualität von Texturen in WebGL zu verbessern. (default: **false**)

Enthaltene Objekte:

- **bitmap** - Bilddaten des Panoramas/Zoom-Bildes. Bei **"multilevel": true** werden mehrere Bitmap-Elemente des Bildes in verschiedenen Größen erwartet.

Eigenschaften:

- **width** - [Number], Breite des Gesamtbildes in diesem bitmap-Element
- **height** - [Number], Höhe des Gesamtbildes in diesem bitmap-Element
- **tilesize** - [Number], Breite und Höhe einer Bildkachel in diesem bitmap-Element, falls das Bild in Kacheln aufgeteilt ist.
- **src** - [String]. URL auf eine JPG-Bilddatei oder ein Template auf eine Reihe von JPG-Bildkacheln. Falls das Bild in Kacheln geteilt ist, wird als Platzhalter in der URL für die x- und y-Koordinate der Bildkachel %x und %y verwendet. Führende Nullen können z.B. für eine dreistellige Zahl mittels %00x angegeben werden.

Beispiel:

```
"bitmap" : {
  "width": 22705,
  "height": 7918,
  "tilesize": 841,
  "src": "pano1/pano1.tiles/l4\_\\%0y\_\\%0x.jpg"
}
```

Enthaltene Objekte von bitmap:

- **"left", "right", "front", "back", "up", "down"** - In einem kubischen Panorama werden im bitmap-Objekt die 6 Würfelseiten in diesen Unter-Objekten erwartet. Im Attribut **src** [String]. Wird das Bild oder die Bildkacheln für jede Würfelseite analog wie ansonsten im **src**-Attribut des bitmap-Objekt erwartet. **width** und **height** im übergeordneten bitmap-Objekt müssen gleich sein und die Größe einer Würfelseite beschreiben.
- **preview** - Vorschaubild

Eigenschaften:

- **src** - [String], Pfad auf eine JPG-Bilddatei
 - **rotate** - [String], Bei kubischen Panoramen, muss das Vorschaubild ein horizontaler Streifen aus den 6 Würfelseiten sein. Das **rotate**-Attribut enthält optional eine codierte Anweisung die Würfelseiten nach dem Laden nochmal einzeln zu drehen. Es wird ein String aus 6 Zeichen erwartet. Dabei bedeutet 0 keine Drehung, 1: 90°, 2: 180°, 3: 270°.
- Beispiel:

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

"include":

include ermöglicht das Einbinden von weiteren Parameterdateien. Die Include-Dateien haben dabei das gleiche Format wie die Haupt-Parameterdatei. Innerhalb des **"panoStudioViewer"**-Objekts können dort wiederum Einstellungen und Objekte definiert werden, die in das gerade angezeigte Panorama übernommen werden. So lassen sich beispielsweise Panorama-übergreifend eigene Bedienelemente oder Parameter und Einstellungen in eine Include-Datei auslagern, die dann in verschiedene Panoramen eingebunden werden kann.

Eigenschaften:

- **src** - [String], URL auf eine Include-Datei

Beispiel:

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

"key":

Element zur Eingabe des Lizenzschlüssels einer Single-Domain-Lizenz. Die Lizenz einer Multi-Domain-Lizenz hingegen wird von PanoramaStudio im Viewer selbst eingetragen.

Beispiel:

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

"lensflare":

Eine künstliche Linsenreflektion (*Lens Flare*) in einem 3D-Panorama.

Eigenschaften:

- **pan** - [Number], horizontale Position des Lens Flares in Grad (min: 0, max: 360)
- **tilt** - [Number], vertikale Position des Lens Flares in Grad (min: -90, max: 90)
- **type** - [Number], Typ des Lens Flares. Es gibt hier 7 vordefinierte Typen (min: 0, max: 6)
- **brightening** - [Number], Aufhellung beim Blick in die Lichtquelle (default: 0.8, min: 0, max: 1)
- **scale** - [Number], Skalierung der Reflexionseffekte (default: 1, min: 0.1, max: 10)
- von Objekt übernommene Eigenschaften: Kap. 4.3.1 (→ S.28)

"location":

In **location** kann optional Ortsinformation der Szene abgelegt werden. Das **location**-Objekt kann Bestandteil eines **node**-Objekts oder **gallery**-Eintrags sein.

Eigenschaften:

- **heading** - [Number] Richtung der Mitte des Panoramabildes in Grad. 0 ist Norden. Die Richtung wird für die Kompass/Radar-Funktion und für die Radar-Option auf der Karte verwendet. (default: 0).
- **lat** - [Number] Breitengrad als Dezimalzahl (default: 0).
- **lon** - [Number] Längengrad als Dezimalzahl (default: 0).

"logo":

Bei Offline-Nutzung oder falls ein Lizenzschlüssel für gewerbliche Nutzung vorhanden ist (siehe key), kann mit dem logo-Objekt ein eigenes Logo in den Viewer anstelle des PanoramaStudio-Viewer-Logos eingeblendet werden.

Eigenschaften:

- **src** - [String], URL auf eine Bilddatei für das Logo (GIF, PNG oder JPEG).
- **href** - [String], Ziel-URL der Webseite, die bei Klick auf das Logo geöffnet werden soll.
- **target** - [String], HTML-Target.
- von Layer übernommene Eigenschaften: Kap. 4.3.2 (→ S.28)
- von Layer übernommene Events: Kap. 5.2.3 (→ S.32)

Enthaltene Objekte:

- **textbox** - Zeigt ein textbox-Objekt als Tooltip über dem Logo an (siehe textbox-Objekt). Lediglich die Position des Tooltips wird nicht durch textbox-Eigenschaften vorgegeben, sondern beim Logo immer an der Maus über dem Logo dargestellt.

"mapElement":

Ein mapElement enthält eine Karte, auf der die Orte einer virtuellen Tour dargestellt werden können. Auf dem mapElement werden die Einträge des gallery-Objekts mit Markern visualisiert, sofern diese die nötigen Ortsdaten enthalten (location). Eine Karte kann ein freistehendes Element im Viewer oder Teil eines buttonBox-Elements sein.

Als Kartendienst wird *Google Maps* verwendet und die Karte mittels der *Google Maps JavaScript API* eingebettet.

Um diesen Google-Dienst zu nutzen, benötigen Sie einen *API-Key*, den Sie bei Google anfordern können. Je nach Nutzungsumfang bzw. bei sehr vielen Aufrufen der Karte können bei Google Kosten für die Nutzung der *Maps JavaScript API* anfallen. Die meisten Anwendungsfälle werden jedoch kostenfrei sein. Sie finden unter folgendem Link bei Google weitere Informationen und die Möglichkeit einen API-Key anzufordern:

<https://developers.google.com/maps/documentation/javascript/get-api-key?hl=de>

Eigenschaften:

- **gmapstype** - [String], Darstellungsart der Karte.
Mögliche Werte sind "roadmap" für (Straßen-)Karte, "satellite" für Satellitenansicht, "hybrid" für Satellitenansicht mit Beschriftungen, "osm" zur Verwendung von OpenStreetMap-Daten in der Google Maps API.
- **gmarker** - [String], Vorgegebenen Marker durch eigenen Marker ersetzen. gmarker enthält den Wert für die Markereigenschaft icon der Google Maps API. Siehe https://developers.google.com/maps/documentation/javascript/markers?hl=de#complex_icons

Beispiel:

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

- **gmarkeractive** - [String], Vorgegebenen aktiven Marker durch eigenen Marker ersetzen. Siehe `gmarker`.
- **apikey** - [String], Ein *Google Maps JavaScript API-Key* damit Google Maps auf der eigenen Webseite bzw. in den Panoramen eingebettet werden kann (nötig).
Siehe <https://developers.google.com/maps/documentation/javascript/get-api-key?hl=de>
- **zoom** - [int], Initiale Zoom-Stufe der Karte. Der Zoom kann zwischen 0 (Welt) und 18 (maximaler Zoom) eingestellt werden. Für manche Regionen ist ein Zoom bis 21 möglich. (default: 18, min: 0, max: 21)
- **zoomcontrols** - [Boolean], Zeigt Bedienelemente für das Zoomen. (default: true)
- **typecontrols** - [Boolean], Zeigt Bedienelemente zum Umschalten des Kartentyps. (default: true)
- **onmarkerclick** - [Function], Wird beim Klick auf einen Marker aufgerufen und ermöglicht, neben dem automatisch ausgeführten Aufruf des Ziel-Panoramas, weitere Aktionen.
- **width** - [String], Breite der Karte in Prozent (%) oder Pixel (px). Beispiel:

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

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

- **height** - [String], Höhe der Karte in Prozent (%) oder Pixel (px).
- **style** - [String] Konfiguriert den CSS-Style des `mapElement`, zum Beispiel Rahmen und Hintergrundfarbe.
- **contentstyle** - [String] Konfiguriert den CSS-Style (z.B. Position und Größe) der Google-Karte innerhalb des Karten-Elements.
- von Layer übernommene Eigenschaften: Kap. 4.3.2 (→ S.28)
- von Layer übernommene Events: Kap. 5.2.3 (→ S.32)

Enthaltene Objekte:

- **radar** - Zeigt in der Karte auf dem aktiven Marker ein `orientation`-Element im radar-Stil an. Siehe `orientation` (Kap. 4.2 (→ S.24)).

"node":

In einem `node` kann eine Szene bzw. ein Panorama abgelegt werden. Liegen die einzelnen Panoramen einer virtuellen Tour jeweils in einem `node`, ist es möglich zum Beispiel Bedienelemente szenenübergreifend zu nutzen.

Folgende Objekte können in einem `node` abgelegt werden:

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

"orientation":

Das `orientation`-Objekt zeigt die Blickrichtung in Form eines Radar- oder Kompass-Elements.

Properties:

- `type` - [String], `compass` für die Darstellung eines Kompass-Elements, `radar` für die Darstellung eines Radar-Elements (default: `compass`).
- `radius` - [Number], Größe des Radars innerhalb des Elements (default: 0.9, min: 0, max: 1).
- `rotatedial` - [Boolean], `true`, damit sich in der Kompass-Darstellung die Scheibe dreht (die Kompass-Nadel steht fest), `false`, damit sich die Kompass-Nadel bewegt (default: `false`).
- `skindial` - [String], Definiert das Aussehen der Kompass-Scheibe oder des Radar-Hintergrunds. Siehe Kap. 4.3.3 (→ S.28).
- `skinframe` - [String], Definiert das Aussehen eines feststehenden Rahmens um den Kompass oder das Radar-Element. Siehe Kap. 4.3.3 (→ S.28).
- `skinneedle` - [String], Definiert das Aussehen der Kompass-Nadel. Siehe Kap. 4.3.3 (→ S.28).
- `radarcontext` - [Function], Funktion, die mit dem 2D-Canvas-Kontext des Radars als Argument aufgerufen wird. Ermöglicht vor dem Zeichnen des Radars Attribute wie `lineWidth`, `fillStyle`, `strokeStyle` des Kontexts zu modifizieren.

"settings":

Hier ist es möglich das Verhalten beim Laden der Bilddaten zu beeinflussen

Eigenschaften:

- **loadqueues** - [Number], Maximale Zahl von Bildkacheln, die gleichzeitig geladen werden (default: 4, min: 1, max: 10)
- **crossOrigin** - [String], Ermöglicht optional das crossOrigin-Attribute in den vom Viewer verwendeten `<img>`-Elementen des DOM zu setzen (default: undefined, wenn Viewer lokal läuft, sonst Anonymous)
- **backgroundColor** - [#RRGGBB oder #RGB], Hintergrundfarbe als hexadezimaler Farbcode. Genutzt in Zoom-Bildern, wenn das Bild nicht den gesamten Viewer ausfüllt und Hintergrund sichtbar wird (default: #000000).
- **safeareamargin** - [String], Korrekturwert in Pixel, der im Falle der Anwendung einer Erweiterung der Displayfläche (siehe `viewport_fit_cover`, Kap. 5.3.1 (→ S.33)) noch auf die Position von Elementen an den Bildrändern addiert oder subtrahiert wird. Reihenfolge und Anwendung identisch mit padding in CSS (oben, rechts, unten, links). So kann mit negativen Werten verhindert werden, dass Elemente etwa allzu weit eingerückt werden. Beispiel:

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

"textbox":

Dient der Darstellung von Text und anderen Inhalten. Wird u.a. vom `hotspot`- und `logo`-Objekt als Tooltip verwendet, kann aber auch direkt zur Darstellung von Text ins Panorama eingebunden werden.

Eigenschaften:

- **text** - [String] Kann beliebiges HTML für Text und andere Elemente enthalten.
- **alpha** - [Number], Transparenz der Textbox, 0: durchsichtig, 1: undurchsichtig (default: 1.0, min: 0.0, max:1.0)
- **background** - [Boolean], Text vor Hintergrund anzeigen (default: false)
- **backgroundColor** - [Number], Hintergrundfarbe hexadezimal oder in HTML-Schreibweise
- **width** - [String], feste Breite der Textbox vorgeben, auch wenn der Text z.B. nur eine Zeile enthält, die kürzer ist. Angabe in Prozent (%) oder Pixel (px). Beispiel:

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

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

- **maxwidth** - [String], ist `width` nicht definiert kann mit `maxwidth` eine maximale Breite vorgegeben werden. Vor allem bei einzeiligen Texte mit Hintergrund, die kleiner sind, wird die Größe der Textbox hier angepasst. Bei längeren Texten erfolgt automatisch ein Zeilenumbruch. Angabe in Prozent (%) oder Pixel (px).
- von Layer übernommene Eigenschaften: Kap. 4.3.2 (→ S.28)
- von Layer übernommene Events: Kap. 5.2.3 (→ S.32)

"toolbar": (veraltet)

Das alte toolbar-Element wird ersetzt durch `buttonBox` und `button`. Wird vom aktuellen Viewer in einer alten Parameterdatei (PanoramaStudio Viewer-Version < 4) ein toolbar-Element gefunden, dann wird stattdessen eine Variante aus vorgegeben Schaltflächen basierend auf `buttonBox` und `button` eingeblendet.

"translate":

Das `translate`-Objekt ermöglicht es Texte entsprechend der Spracheinstellung des Browsers zu übersetzen. Dazu kann es Objekte enthalten, die den Names des zu übersetzenden Language Codes tragen, z.B. `en-US`, `de` oder `de-AT`.

Diese Language-Code-Objekte können dann Key-Value-Paare enthalten, bei denen der Key der zu übersetzende Text und Value den übersetzten Text enthält.

Zum Übersetzen stellt das Viewer-Objekt die Funktion `tr()` bereit, welche die Einträge im `translate`-Objekt nutzt, um Texte entsprechend der Browsersprache anzuzeigen. Siehe `tr()`, S. 39.

Beispiel:

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

"video":

Einbinden eines Videos.

Eigenschaften:

- `src` - [String], URL auf eine MP4-Videodatei
- `src_alt` - [String], optionale URL auf eine Videodatei im WebM-Format, als alternative Video-Quelle für Browser, die kein MP4 unterstützen.
- `poster` - [String], Spezifiziert ein Bild, das anstelle des Videos gezeigt wird, bevor das Video gestartet wird.
- `loop` - [Boolean], Videodatei in einer Endlosschleife abspielen (default: false)
- `autoplay` - [Boolean], Starte das Abspielen sofort nach dem Laden des Panoramas (default: false)
- `videowidth` - [Number], Breite des Quell-Videos in Pixel
- `videoheight` - [Number], Höhe des Quell-Videos in Pixel
- `width` - [Number], Breite des Video-Elements im Panorama (siehe auch `unit`)
- `position` - [String], Position des Video-Elements im Panorama (siehe auch `unit`)
- `rotate` - [Number], Drehung des Videos in Grad

- **unit** - [String], Einheit der Koordinaten im **position**-Attribut. Mögliche Werte: **deg**, Angabe der Werte in Grad, **norm** Angabe der Werte normiert auf Werte von 0 bis 1. Für Panoramen ist nur die Angabe in **deg** möglich, für planare Zoom-Bilder nur die Angabe in **norm**.
- **volume** - [Number], Lautstärke des Videos (default: 1.0, min: 0.0, max: 1.0)
- von Layer übernommene Eigenschaften: Kap. 4.3.2 (→ S.28)
- von Layer übernommene Events: Kap. 5.2.3 (→ S.32)

Beispiel:

```
"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":

Einstellung der initialen Ansicht auf das Panorama/Zoom-Bild

Eigenschaften:

- **pan** - [Number], horizontale Blickrichtung im Panorama in Grad (default: 0)
- **tilt** - [Number], vertikale Blickrichtung im Panorama in Grad (default: 0)
- **hfov** - [Number], horizontaler Bildwinkel im Panorama in Grad (default: 90)
- **x** - [Number], horizontale Position des Betrachters bei einem Zoom-Bild im Bereich von 0 (links) bis 1 (rechts) (default: 0.5)
- **y** - [Number], vertikale Position des Betrachters bei einem Zoom-Bild im Bereich von 0 (oben) bis 1 (unten) (default: 0.5)
- **zoom** - [Number], Vergrößerung bei Zoom-Bildern, ein Wert von 0 zeigt das Bild komplett ausgezoomt, ein Wert von 1 zeigt das Bild in Originalgröße (default: 0)
- **mode** - [String], Modus bei Zoom-Bildern, **fillWindow**: Bild kann nur soweit ausgezoomt werden, dass es das Fenster gerade noch ausfüllt oder **fitInWindow**: Bild kann soweit ausgezoomt werden, dass es komplett ins Fenster passt (default: **fitInWindow**)
- **shortdescription** - [String], Kurze Beschreibung, die für die Markierung in der Karte und die Galerie genutzt wird (analog zu **text** im **gallery**-Eintrag).

4.3 Universaleigenschaften

Einige Eigenschaften haben alle Objekte und Layer gemeinsam:

4.3.1 Eigenschaften aller Objekte

- **id** - [String], Objekt eine eindeutige ID zuweisen, um z.B. mittels `get()`-Funktion darauf zuzugreifen.

4.3.2 Eigenschaften aller Layer-Objekte

Eigenschaften auf Layer-Objekten und allen davon abgeleiteten:

- **viewer** - [Object], Zugriff auf die Viewer-Instanz Kap. 5.1 (→ S.30).
- **visible** - [Boolean], Sichtbarkeit des Layers (default: true)
- **checkHover** - [Boolean], Layer erfasst Mausbewegungen und meldet entsprechende Ereignisse.
- **captureMouse** - [Boolean], wenn die Maus über einem Layer gedrückt wird, dann werden alle folgenden Maus-Ereignisse bis zum Loslassen der Maustaste von diesem Layer erfasst, auch wenn sich die Maus außerhalb des Layers befindet.
- **style** - [String], Entspricht dem CSS-Style-Attribut in HTML. Layer können auf diese Weise mittels CSS gestaltet werden.
- **align** - [String], Ausrichtung eines Elements im Fenster, mögliche Werte: `left`, `top`, `right`, `bottom`, `lefttop`, `leftbottom`, `righttop`, `rightbottom` und `center`
- **xoff** - [Number], Versatz nach links (<0) oder rechts (>0) in Pixel.
- **yoff** - [Number], Versatz nach oben (<0) oder unten (>0) in Pixel.
- **padding** - [Number], Padding analog CSS in Pixel (default: 0)
- von Objekt übernommene Eigenschaften: Kap. 4.3.1 (→ S.28)

4.3.3 Eigenschaften von Button und Orientation

- **skin** - [String], Eine Folge von Instruktionen, die es ermöglicht Grafiken auf den Layer zu kopieren und mit einigen Filtern zu modifizieren. Folgende Instruktionen werden unterstützt:
 - `dim(String dstx,String dsty,String dstw,String dsth,int srcw,int srch)`
 Von der Größe des Layers abweichende Größe der Grafik festlegen.
`dstx, dsty`: Position linke obere Ecke der Grafik auf dem Layer in Pixel (px) oder Prozent (%).
`dstw, dsth`: Breite/Höhe der Grafik auf dem Layer in Pixel (px) oder Prozent (%).
`srcw, srch`: Breite/Höhe der internen Zeichenfläche (Canvas) für die Grafik.
 - `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])`
 Einen Ausschnitt aus einem gfx-Objekt auf den Layer kopieren.
`gfxId`: ID eines gfx-Objekts, das als Quelle verwendet wird. Der Viewer enthält unter `defaultSkin` bereits eine integrierte Grafik für die Standard-Bedienelemente.
`srcx, srcy`: Position linke, obere Ecke in der Grafik.
`srcw, srch`: Breite/Höhe des Ausschnitts in der Grafik.
`dstx, dsty`: Position linke, obere Ecke in der Zeichenfläche des Layers.
`srcw, srch`: Breite/Höhe des Zielbereichs im Layer.
`tintColor`: Optional kann die zu kopierende Grafik mit einer Farbe eingefärbt werden. Farbangabe im CSS-Format, z.B. `blue`, `#ff0000` oder `rgb(255,0,0)`
`actionId`: Optional kann der Canvas-Kontext direkt bearbeitet werden. `actionId` ist dabei die Id einer Aktion die aufgerufen wird, wenn der Canvas initialisiert wird. Der erste Parameter beim Aufruf enthält den Canvas-Kontext.

- `comp(Number alpha,String composite[optional])`: Alpha bzw. Transparenzwert und Kompositionsmodus für folgende `copy`-Instruktionen einstellen.
alpha: Wert zwischen 0.0 (komplett transparent) und 1.0 (undurchsichtig) (default: 1.0).
composite: Kompositionsmodus, der bei `copy` angewandt wird. Entspricht `globalCompositeOperation` des HTML-Canvas-Elements (default: `source-over`). Mögliche Werte siehe z.B.:
http://www.w3schools.com/tags/canvas_globalcompositeoperation.asp.
- `shadow(Number blur,Number xoff,Number yoff,String color)` Auf nachfolgende `copy`-Instruktionen einen Schatteneffekt anwenden.
blur: Unschärfe in Pixel.
xoff: X-Versatz des Schattens in Pixel.
yoff: Y-Versatz des Schattens in Pixel.
color: Schattenfarbe. Farbangabe im CSS-Format, z.B. `blue`, `#ff0000` oder `rgb(255,0,0)`

Beispiel:

```
"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)"
```

- von Layer übernommene Eigenschaften: Kap. 4.3.1 (→ S.28)

5 JavaScript API

5.1 Zugriff auf ein Panorama

Enthält ein DIV-Element ein PanoramaStudio Viewer-Objekt, dann enthält es die Variable `panoStudioViewer`. Entsprechend erhält man das Viewer-Objekt z.B. mittels:

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

5.2 Events

5.2.1 Übersicht

Der PanoramaStudio Viewer stellt eine Reihe von Events zur Verfügung, die genutzt werden können, um Aktionen auszuführen, wenn ein bestimmtes Ereignis eintritt.

Events können auf mehrere Arten erfasst werden.

- Extern in der Webseite:

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

- Globale Events einer Viewer-Instanz im `events`-Objekt der Parameterdatei. Funktionen werden in die Parameter-Dateien des Viewers als String eingefügt:

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

- Lokale Events eines Layers im zugehörigen Objekt. Zum Beispiel onclick auf einem button:

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

5.2.2 Events einer `panoStudioViewer`-Instanz

Folgende Ereignisse werden vom `panoStudioViewer`-Objekt bereitgestellt:

```
onenterfullscreen()
```

Wird aufgerufen, wenn der Viewer in den Vollbildmodus umgeschaltet wurde.

```
onentervrmode()
```

Wird aufgerufen, wenn der Viewer in den VR-Modus schaltet.

`onexit()`

Wird aufgerufen, bevor ein Panorama verlassen wird.

`onexitfullscreen()`

Wird aufgerufen, wenn der Viewer den Vollbildmodus verlassen hat.

`onexitvrmode()`

Wird aufgerufen, wenn der Viewer in den VR-Modus verlassen hat.

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

Wird für jedes Frame aufgerufen.

Optionale Parameter:

frameTime, Zeit in Millisekunden seit letztem Frame

viewChanged, hat sich die Ansicht im Panorama geändert

Beispiel:

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

`ongyroscopetoggle(enabled/*Boolean*/)`

Wird aufgerufen, wenn sich der Status der Steuerung per Gyroskop ändert. Dabei signalisiert enabled, ob die Gyroskop-Steuerung aktiviert oder deaktiviert ist.

`oninactive(ms/*Number*/)`

Wird nach 2 Sekunden ohne Nutzerinteraktion aufgerufen. Aufruf wird alle 2 Sekunden ohne Nutzerinteraktion wiederholt.

ms, Zeit in Millisekunden seit letzter Nutzerinteraktion.

`oninit()`

Wird aufgerufen, wenn ein neues Panorama initialisiert wurde.

`onplay()`

Wird aufgerufen, wenn das Auto-Play gestartet wird.

`onresize()`

Wird aufgerufen, wenn sich die Größe des Viewer-Elements geändert hat.

`onscenechanged()`

Wird aufgerufen, wenn die Ansicht in ein anderes Panorama gewechselt hat, etwa durch Klick auf einen Hotspot oder auf einen Eintrag in der Galerie.

`onstartaudio(id/*String*/)`

Wird aufgerufen, wenn das Abspielen eines Audio-Objekts gestartet wird.
id, ID des Audio-Objekts.

```
onstop()
```

Wird aufgerufen, wenn das Auto-Play gestoppt wird.

```
onstopaudio()
```

Wird aufgerufen, wenn das Abspielen eines Audio-Objekts gestoppt wird.
id, ID des Audio-Objekts.

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

Wird nach dem Initialisieren des Viewers aufgerufen und meldet, ob WebGL verwendet wird. Optionale Parameter:

available, enthält *true*, falls WebGL für die Darstellung genutzt wird, ansonsten wird CSS3D verwendet.

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

Wird aufgerufen, wenn sich die Ansicht im Viewer geändert hat.

Optionale Parameter:

mouseX, mouseY, Position des Cursors.

```
onvisibilitychange()
```

Wird aufgerufen, wenn der Viewer bzw. Browser-Tab ein- oder ausgeblendet wurde.

5.2.3 Lokale Events auf jedem Layer

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

Wird aufgerufen, wenn die linke Maustaste geklickt oder ein Touch-Event stattfand.

mouseX, mouseY sind die Mauskoordinaten relativ zum Viewer-Fenster. Liefert zurück, ob das Ereignis vom Objekt verarbeitet wurde, dann wird es an keine weiteren Objekte mehr geschickt.

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

Wird aufgerufen, wenn ein Doppelklick stattfand.

mouseX, mouseY sind die Mauskoordinaten relativ zum Viewer-Fenster. Liefert zurück, ob das Ereignis vom Objekt verarbeitet wurde, dann wird es an keine weiteren Objekte mehr geschickt.

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

Wird aufgerufen, wenn die linke Maustaste gedrückt oder ein Touch-Event startet.

mouseX, mouseY sind die Mauskoordinaten relativ zum Viewer-Fenster. Liefert zurück, ob das Ereignis vom Objekt verarbeitet wurde, dann wird es an keine weiteren Objekte mehr geschickt.

```
onexit()
```

Wird aufgerufen, bevor das Objekt gelöscht wird.

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

Wird für jedes Frame aufgerufen.

Optionale Parameter:

frameTime, Zeit in Millisekunden seit letztem Frame.

viewChanged, hat sich die Ansicht im Panorama geändert.

```
oninit()
```

Wird aufgerufen, wenn ein das Objekt initialisiert wurde.

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

Wird aufgerufen, wenn die Maus über ein Objekt bewegt wird.

mouseX, mouseY sind die Mauskoordinaten relativ zum Viewer-Fenster. Liefert zurück, ob das Ereignis vom Objekt verarbeitet wurde, dann wird es an keine weiteren Objekte mehr geschickt.

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

Wird aufgerufen, wenn der Mauszeiger ein Objekt verlässt.

mouseX, mouseY sind die Mauskoordinaten relativ zum Viewer-Fenster.

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

Wird aufgerufen, wenn der Mauszeiger erstmals über ein Objekt bewegt wird.

mouseX, mouseY sind die Mauskoordinaten relativ zum Viewer-Fenster.

```
onresize()
```

Wird aufgerufen, wenn sich die Größe des Viewer-Elements geändert hat.

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

Wird aufgerufen, wenn die linke Maustaste losgelassen oder ein Touch-Event ended und wenn das Objekt die Maus gerade erfasst.

mouseX, mouseY sind die Mauskoordinaten relativ zum Viewer-Fenster.

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

Wird aufgerufen, wenn sich die Ansicht im Viewer geändert hat.

Optionale Parameter:

mouseX, mouseY, Position des Cursors.

5.3 Funktionen

5.3.1 Funktionen des globalen, statischen panoStudioViewer-Objekts

Wenn der PanoramaStudio Viewer mittels panoStudioViewer.js in einer Seite geladen ist, dann steht zum Hinzufügen und Entfernen von Panoramen ein globales, statisches Objekt panoStudioViewer zur Verfügung.

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

Fügt dem DIV-Element mit der eindeutigen id ein Panorama definiert durch die Parameter-Datei parameterFile hinzu.

inlineParameters kann optional eine Liste mit Parametern enthalten, die dem Viewer beim Start übergeben werden. Mögliche Werte sind:

- {html5:disable_webgl_warning} , zeige keine Warnmeldung, wenn der Browser kein WebGL zur Verfügung stellt
- {html5:nowebgl} , nutze WebGL nicht, stattdessen CSS-3D-Transformationen
- {html5:viewport_fit_cover}, wenn der Viewer so konfiguriert ist, dass er das Browserfenster ausfüllt oder im Vollbildmodus angezeigt wird, dann wird das Panorama mit dieser Option weiter ausgedehnt, um auch die Ränder auf Geräten mit abgerundeten Displayecken oder einer Aussparung im Display auszufüllen, die sonst nicht abgedeckt wird.

Oder jede Kombination der obigen Optionen:

- {html5:nowebgl|disable_webgl_warning}

startupCallback kann optional eine Funktion enthalten, die nach dem Start vom Viewer aufgerufen wird. Die Funktion muss als String in folgender Form vorliegen:

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

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

Entfernt ein Panorama anhand der eindeutigen id des enthaltenden DIV-Elements.

5.3.2 Funktionen der Viewer-Instanz

Enthält ein DIV-Element ein Panorama, dann kann über die lokale Variable panoStudioViewer des Elements auf diese Viewer-Instanz zugegriffen werden (nicht zu verwechseln mit dem globalen, statischen panoStudioViewer-Objekt).

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

Folgende Funktionen stehen zur Verfügung:

- 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(id, arg1, arg2, ...)
```

Sucht in den actions-Objekten (Kap. 4.2 (→ S.10)) eine Funktion mit entsprechender *id* und ruft die Funktion mit den Argumenten (*arg1*, *arg2*, ...) auf. Falls die aufgerufene Funktion einen Wert zurückgibt, dann liefert *action()* diesen Wert zurück.

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

Fügt einen Layer vom Typ *button*, *buttonBox*, *audio*, *video* oder *textbox* hinzu.
json enthält ein JSON-Objekt mit den für den Typ nötigen Parametern.

Beispiel, Info-Box bei Klick auf einen Button einblenden:

```

"button" : {
    [...]
    /* eine textbox hinzufügen; JSON-Objekt ist als 'textElement' hier ebenfalls enthalten */
    "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()
```

Liefert ein Objekt mit den Parametern des camera-Objekts (minpan, maxpan, ...).

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

Liefert true, wenn Bewegungssensoren zur Steuerung des Panoramas zur Verfügung stehen.

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

Liefert die Laufzeit der aktuellen Szene in Millisekunden.

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

Rufe mit true oder false auf, um die Bewegungs-Steuerung zu aktivieren oder zu deaktivieren. Liefert den neuen Status zurück.

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

Schaltet den Viewer in den VR-Modus. Liefert, ob der VR-Modus erfolgreich aktiviert wurde.

```
exitVRMode()
```

Verlässt den VR-Modus.

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

Gibt true zurück, wenn das aktuelle Panorama ein 360x180°-Panorama ist, ansonsten false.

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

Liefert true, wenn der Vollbildmodus vom Browser unterstützt wird.

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

Liefert das gallery-Array, falls definiert. Siehe Kap. 4.2 (→ S.16).

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

Ein Objekt anhand seiner id finden.

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

Liefert ein Objekt mit den Winkeln für Blickrichtung und Sichtfeld (pan, tilt, hfov).
Bei einem planaren Bild sind die Attribute Bildmitte und Zoom (x, y, zoom).

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

Liefert true, wenn die Bewegungs-Steuerung aktiv ist.

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

Aktuelle Höhe des Viewers in Pixel.

```
hideCursor()
```

Blendet den Mauscursor aus.

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

Liefert ein Array der Hotspots im Panorama zurück.
Das Hotspots-Array stellt einige zusätzliche Funktionen bereit:

- `/*Object*/ get(/*int*/ i)`, Liefert das JSON-Objekt des Hotspots an Index-Position *i*.
- `remove(/*int*/ i)`, Entfernt den Hotspot an Index-Position *i*.
- `insert(/*int*/ i, /*Object*/ h)`, Fügt einen Hotspot *h* als JSON-Objekt (siehe Kap. 4.2 (→ S.19)) an Index-Position *i* ein.
- `enableDrag(/*Boolean*/ b)`, Aktiviert oder Deaktiviert das Verschieben von Hotspots per Drag&Drop.

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

Liefert die Zeit seit der letzten Nutzerinteraktion in Millisekunden.

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

Liefert true, wenn der Viewer auf einem Android- oder iOS-Mobilgerät läuft.

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

Liefert true, wenn das Auto-Play aktiv ist.

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

Liefert, ob sich der Viewer im VR-Modus befindet.

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

Liefert ein Array der Lensflares im Panorama zurück.
Das Lensflares-Array stellt einige zusätzliche Funktionen bereit:

- `/*Object*/ get(/*int*/ i)`, Liefert das JSON-Objekt des Lensflares an Index-Position *i*.
- `remove(/*int*/ i)`, Entfernt das Lensflare an Index-Position *i*.

- `insert(/*int*/ i,/*Object*/ h)`, Fügt ein Lensflare `h` als JSON-Objekt (siehe Kap. 4.2 (→ S.21)) an Index-Position `i` ein.

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

Liefert `location`-Objekt, falls definiert. Siehe Kap. 4.2 (→ S.21).

```
lock()
```

Sperrt den Viewer für Nutzereingaben.

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

Liefert das `mapElement`-Objekt, falls definiert. Siehe Kap. 4.2 (→ S.22).

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

Liefert den `maxzoom`-Wert wie in `camera` gesetzt.

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

Öffnet ein anderes Panorama im bestehenden Viewer-Fenster. Dem Parameter `parameterFile` muss dabei der Name der Parameterdatei des zu öffnenden Panoramas zugewiesen werden.

Optional sind zudem die Parameter `cmd`, `blend`, `time` und `nodeOnly`. Wird `blend` auf `true` gesetzt, blendet der Viewer sanft ins neue Panorama über. Die Dauer in Sekunden kann in diesem Fall zusätzlich noch mittels `time` eingestellt werden.

Der Parameter `cmd` erlaubt eine einfache Befehlskette, um an dieser Stelle z.B. ein Sichtfeld vorzugeben mit dem das neue Panorama geöffnet wird. Dazu können für 3D-Panoramen die Parameter `view.pan`, `view.tilt` und `view.hfov` verwendet werden. Bei 2D-Zoom-Bildern entsprechend die Parameter `view.x`, `view.y` und `view.zoom` (siehe dazu auch `setView`).

Hier ein Beispiel für das Öffnen eines Panoramas in JavaScript mit Blick in Richtung 100° horizontal und 5° vertikal bei einem Bildwinkel von 45°:

```
<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;');" />
```

Falls `nodeOnly` gleich `true` ist, lädt der Viewer nur den Inhalt des `node`-Objekts in der Parameterdatei und hält Bedienelemente und andere Objekte bei.

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

Liefert 0, falls es sich um ein 3D-Panorama handelt, 1, wenn es ein planares 2D-Zoom-Bild ist.

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

Führt einen Schwenk auf eine neues Sichtfeld mit Ziel `pan`, `tilt`, `hfov` durch. Bei einem planaren Bild sind die Argumente Bildmitte und Zoom (`x`, `y`, `zoom`).

`time` gibt die Dauer des Schwenks in Sekunden an, `delay` die Verzögerung bis zum Beginn des Schwenks. `transition` ist der Tween-Typ. Vorgabe ist `easeInOutQuint` (siehe *Tween-Typen*, S. 40).

Mittels `lock` können Nutzereingaben bis zum Ende des Schwenks blockiert werden, ansonsten bricht der Schwenk bei einer Nutzereingabe ab. `onComplete` ist eine optionale Funktion, die am Ende des Schwenks aufgerufen wird.

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

Einen Layer anhand seiner id finden und löschen.

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

Liefert ein Objekt mit den Werten left, top, right, bottom in Pixeln für die Einrückung, die aktiv ist, wenn der Viewer mit der Option viewport_fit_cover konfiguriert ist und im Vollbildmodus ausgeführt wird oder das Browserfenster füllt. Die Werte entsprechen in dem Fall den CSS-Umgebungsvariablen safe-area-inset-*.

```
/*Object*/ screenToPano(/*Number*/ x,/*Number*/ y)
```

Liefert ein Objekt mit den Winkeln (pan, tilt) für die Blickrichtung an der Screen-Koordinate x, y. Bei einem planaren Bild liefert die Funktion die x-, y-Koordinaten im Bild im Intervall [0;1].

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

Setzt die Ansicht im Panorama auf die Blickrichtung pan, tilt und das Sichtfeld auf hfov. Bei einem planaren Bild werden als Parameter Bildmitte und Zoom erwartet (x, y, zoom).

```
showCursor()
```

Zeigt den Mauszeiger an.

```
startAutoPlay()
```

Startet das Auto-Play.

```
stopAutoPlay()
```

Stoppt das Auto-Play.

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

Liefert Browser- und Betriebssystem-Information.

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

Übersetzt text. Verwendet dazu das translate-Objekt (Kap. 4.2 (→ S.26)).

Liefert eine Übersetzung des Strings text, wenn es ein entsprechendes Key-Value-Paar für den aktuellen Language Code des Browsers im translate-Objekt gibt, ansonsten liefert es text zurück.

```
toggleFullscreen()
```

Schaltet zwischen Vollbildmodus und Fenstermodus um.

```
unlock()
```

Gibt den Viewer für Nutzereingaben frei.

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

Liefert Versionsinformation zum PanoramaStudio Viewer.

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

Aktuelle Breite des Viewers in Pixel.

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

Liefert den aktuellen Zoom-Wert. Wobei 1.0 einer 100%-Ansicht entspricht. Kleinere Werte ausgezoomt, größere eingezoomt.

5.3.3 Lokale Funktionen von Layern

Die gemeinsame Basis aller Elemente in einem Panorama (Buttons, Logos, Hotspots, Textboxen, ...) ist ein sog. Layer-Element. Jedes Layer-Element stellt folgende Funktionen zur Verfügung:

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

Aufrufen mit true setzt Layer sichtbar, ansonsten unsichtbar.

```
updateAlign()
```

Aufrufen, um Änderungen an Positions-Attributen (xoff, yoff, align, ...) anzuwenden.

```
updateStyles()
```

Aufrufen, wenn ein Style-Attribut (style, stylehover, styleactive) geändert wurde.

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

Einen Layer oder Objekt anhand der id finden.

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

Konvertiert die Eigenschaften x und y des coordinate-Objekts von Viewport-Koordinaten in Koordinaten relativ zum Layer.

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

Liefert die linke Position des Layers relativ zum Viewport.

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

Liefert die obere Position des Layers relativ zum Viewport.

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

Liefert die Breite des Layers.

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

Liefert die Höhe des Layers.

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

Mittels tween können Variablen zwischen zwei Werten über die Zeit sanft interpoliert werden, um Animationen zu erstellen.

Das parameters-Objekt enthält dabei alle Parameter des *Tweens* als Key/Value-Paare. Folgende Parameter können gesetzt werden:

- 'time', Zeit für den Tween in Sekunden (default: 1.0)
- 'transition', Tween-Typ (default: easeOutExpo)
- 'delay', Zeit bis zum Start des Tweens (default: 0)

- 'onInit', Funktion, die beim Start des Tweens aufgerufen wird (default: null). Die Funktion erhält als Argumente jeweils das JSON-Objekt des Layers sowie das Objekt mit allen Parametern des Tweens als Key/Value-Paare, so können die Zielparameter beim Initialisieren nochmal dynamisch geändert werden (onInit(layer,parameters)).
- 'onUpdate', Funktion, die bei jedem Update des Tweens aufgerufen wird (default: null). Die Funktion erhält als Argument jeweils das JSON-Objekt des Layers.
- 'onComplete', Funktion, die am Ende des Tweens aufgerufen wird (default: null). Die Funktion erhält als Argument jeweils das JSON-Objekt des Layers.
- 'breakOnUser', Boolean. Falls true, wird der Tween bei einer Nutzerinteraktion vorzeitig abgebrochen und onComplete aufgerufen (default: false).

Als Variablen können alle numerischen Attribute des zum Layer des Tweens gehörenden JSON-Objekts eingesetzt werden.

Beispiel für eine Tween-Animation, welche die Bedienelemente (id: toolbar) innerhalb von 0,5s durch Verschieben von 70 Pixeln nach unten ausblendet.

Es wird dabei die yoff-Eigenschaft des Layers durch den Tween sanft interpoliert. Die onUpdate-Funktion sorgt dabei jeweils dafür, dass die Änderung der Position yoff auf die Darstellung angewendet wird:

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

Folgende Tween-Typen (auch *Easing-Funktionen*) werden aktuell unterstützt:

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

Eine visuelle Darstellung der Tween-Typen finden Sie z.B. hier:
<http://easings.net/de>

5.3.4 Lokale Funktionen von Buttons

Neben den Layer-Funktionen (Kap. 5.3.3, S.40) stellt ein `button` noch folgende Funktionen bereit:

```
updateSkins()
```

Aufrufen, wenn ein Skin-Attribut (`skin`, `skinhover`, `skinactive`) geändert wurde.

5.3.5 Lokale Funktionen von Textboxen

Neben den Layer-Funktionen (Kap. 5.3.3, S.40) stellt eine `textbox` noch folgende Funktionen bereit:

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

Ändert den Inhalt der Textbox. Kann beliebiges HTML enthalten.

5.3.6 Lokale Funktionen von Audio-Objekten

Neben den Layer-Funktionen (Kap. 5.3.3, S.40) stellt ein audio-Objekt noch folgende Funktionen bereit:

```
play()
```

Startet das Abspielen des Audio-Objekts.

```
pause()
```

Pausiert das Abspielen des Audio-Objekts.

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

Liefert true, wenn das Audio-Objekt gerade abgespielt wird.

```
/*Boolean*/ isPlaying()     /*Veraltet*/
```

Liefert true, wenn das Audio-Objekt gerade abgespielt wird.

5.4 Eigenschaften

Auf alle Eigenschaften der panoStudioViewer-Instanz, die in der Parameterdatei definiert sind, kann zur Laufzeit per JavaScript zugegriffen werden (siehe Kap. 5.1 (→ S.30)), ebenso auf alle Eigenschaften von Layer-Objekten mittels `get(...)` (siehe Kap. 5.3.2 (→ S.36)).