

# The media9 Package, v0.43

Alexander Grahn  
a.grahn@web.de

25th March 2014

## Abstract

A L<sup>A</sup>T<sub>E</sub>X package for embedding interactive Adobe Flash (SWF) and 3D files (Adobe U3D & PRC) as well as video and sound files or streams (FLV, MP4/H.246, MP3) into PDF documents with Adobe Reader-9/X compatibility.

*Keywords:* embed flash movie LaTeX pdf 3d include sound swf mp3 video mp4 h.264 aac flv audio multimedia streamed media rtmp YouTube animation JavaScript pdfLaTeX dvips ps2pdf dvi2pdf XeLaTeX u3d prc Adobe Reader RichMedia annotation LuaLaTeX

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Requirements</b>	<b>3</b>
<b>3</b>	<b>Installation</b>	<b>3</b>
<b>4</b>	<b>Using the package</b>	<b>3</b>
<b>5</b>	<b>The user interface</b>	<b>5</b>
5.1	Media inclusion . . . . .	5
5.2	Command options . . . . .	6
5.3	Control buttons . . . . .	16
<b>6</b>	<b>Embedding Flash, video and sound (with examples)</b>	<b>18</b>
<b>7</b>	<b>Embedding 3D objects (with examples)</b>	<b>27</b>
7.1	Introduction . . . . .	27
7.2	3D quick-start guide . . . . .	33
<b>8</b>	<b>Caveats</b>	<b>34</b>
<b>9</b>	<b>Acknowledgements</b>	<b>35</b>

# 1 Introduction

This package provides an interface to embed, in the first place, interactive Flash (SWF) and 3D objects (Adobe U3D & PRC) into PDF documents. Video and sound files or streams in the popular MP4, FLV and MP3 formats can be embedded as well. However, a media player Flash component is required for playback, as will be explained shortly. Playback of multimedia files uses Adobe Flash Player, which was bundled with Adobe Reader 9 and 10 versions. Unfortunately, beginning with Adobe Reader 11, it must be installed as a separate plug-in.

Among the supported media types, video and sound files require an additional Flash (SWF) application for playback, which must be either embedded into the PDF or loaded at runtime from the internet. There are numerous such players, both open-source and commercial, available on the internet. One of them is the highly configurable open-source ‘StrobeMediaPlayback.swf’ [1], maintained by Adobe and hosted on SourceForge.net. Package ‘media9’ comes with an enhanced version of ‘StrobeMediaPlayback.swf’. In addition, two simple players for video and audio, ‘VPlayer.swf’ and ‘APlayer.swf’ are included, which can be used instead. They provide sufficient functionality for playing embedded files and streamed media.

Flash Player supports the efficient H.264 codec for video compression. MP4/H.264 video files can be encoded from existing video files and from numbered bitmap sequences using the `ffmpeg` (<http://ffmpeg.org>) or `avconv` (<http://libav.org>) command line tools (Libav is a fork from the FFmpeg code). In order to allow for precise seeking within video files it is necessary to encode them with a sufficient number of key frames. The command line for recoding an existing video file `video.avi` into `video.mp4` reads (`ffmpeg` can be substituted with `avconv`)

```
ffmpeg -i video.avi -c:v libx264 -g 30 -r 30 video.mp4
```

From a sequence `frame-0.png`, `frame-1.png`, ... of bitmap files, an MP4 video is produced by

```
ffmpeg -i frame-%d.png -c:v libx264 -g 30 -r 30 video.mp4
```

Both examples insert a key frame (option ‘-g’) at every second since the frame rate is set to 30 fps.

*Note:* ‘media9’ package replaces the now obsolete ‘movie15’ package. ‘media9’ is based on the RichMedia Annotation (Annotations are the interactive elements in a document, in PDF specification parlance.), an Adobe addition to the PDF specification [2], while ‘movie15’ uses the old multimedia framework (‘Screen Annotation’) of pre-9 Readers which depends on third-party plug-ins and which does not support recent media file formats.

Package ‘media9’ supports the usual PDF making workflows, i.e. `pdfLATEX`, `LuaLATEX`, `LATEX` → `dvips` → `ps2pdf`/Distiller and `(XY)LATEX` → `(x)dvipdfmx`.

The final PDF can be viewed in current Adobe Readers on MS Windows and other platforms. On Unix platforms including Linux, however, support of Flash, video and sound was discontinued at Reader version 9.4.2, probably for security reasons. PDF documents which target Adobe Reader 9.4.1 for Linux should

use ‘VPlayer9.swf’ and ‘APlayer9.swf’ (also included in the ‘media9’ package). These media player components are compatible with the older Flash Player 9 plugin that is bundled with the Reader for Linux. On tablets and phones running Android or iOS, ezPDF Reader was reported to play video and sound files embedded with ‘media9’.

## 2 Requirements

l3kernel (L<sup>A</sup>T<sub>E</sub>X package), version  $\geq$  2013/07/28

l3packages (L<sup>A</sup>T<sub>E</sub>X package), version  $\geq$  2013/07/28

pdfT<sub>E</sub>X, version  $\geq$  1.30

Ghostscript, version  $\geq$  8.31 or Adobe Distiller for PS to PDF conversion

dvipdfmx, version  $\geq$  20120420 for DVI to PDF conversion

Adobe Reader, version  $\geq$  9, but not greater than 9.4.1 on Linux

## 3 Installation

MiK<sub>T</sub><sub>E</sub>X and T<sub>E</sub>XLive users should run the package manager for installation and updates.

Otherwise, a manual installation into the *local* TeX-Directory-Structure (TDS) root directory is done along the following steps:

1. Download the TDS compliant package file ‘media9.tds.zip’ from CTAN.
2. Find the local TDS root directory by running  
`kpsewhich -var-value TEXMFLOCAL`  
on the command line. The local TDS root directory is intended for packages that are not maintained by the T<sub>E</sub>XLive package manager.
3. Unzip ‘media9.tds.zip’ into the local TDS root directory previously found. Depending on the location of this directory, you may need to be logged in as Root/Administrator.
4. After installation, update the filename database by running ‘texhash’ on the command line. Again, Root/Administrator privileges may be required.

For updating the package, repeat the steps given above.

## 4 Using the package

Invoke the package by putting the line

```
\usepackage[<package options>]{media9}
```

to the preamble of your document, i.e. somewhere between `\documentclass` and `\begin{document}`.

‘media9’ honours the package options:

```
dvipdfmx
xetex
bigfiles
draft
final
playbutton=...
noplaybutton
activate=...
deactivate=...
windowed=...
transparent
passcontext
3Dplaytype=...
3Dplaycount=...
3Dplayspeed=...
3Dtoolbar
3Dnavpane
3Dpartsattrs=...
3Dmenu
3Dbg=...
3Dlights=...
3Drender=...
```

Except for ‘dvipdfmx’, ‘xetex’ and ‘bigfiles’, the options above are also available (among others) as command options and will be explained shortly. However, if used as package options they have global scope, taking effect on all embedded media in the document. In turn, command options locally override global settings. Options without an argument are boolean options and can be negated by appending ‘=false’.

X<sub>Y</sub>L<sup>A</sup>T<sub>E</sub>X will be auto-detected. Therefore package option ‘xetex’ is optional. However, in the case of dvipdfmx, package option ‘dvipdfmx’ is mandatory because it cannot be auto-detected.

If PDF is generated via DVI and Postscript by the command sequence `latex` → `dvips` → `ps2pdf`, dvips option ‘-Ppdf’ should *not* be set when converting the intermediate DVI into Postscript. If you cannot do without, put ‘-D 1200’ *after* ‘-Ppdf’ on the command line. Users of L<sup>A</sup>T<sub>E</sub>X-aware text editors with menu-driven toolchain invocation, such as T<sub>E</sub>XnicCenter, should check the configuration of the dvips call.

Option ‘bigfiles’ is only relevant for the `latex` → `dvips` → `ps2pdf` workflow. It may be needed if large media files cause `latex` to abort with error ‘TeX capacity exceeded’. See Sect. 8.

## 5 The user interface

Package ‘media9’ provides commands for media inclusion (`\includemedia`) and insertion of media control buttons (`\mediabutton`). The latter is introduced in Sect. 5.3.

### 5.1 Media inclusion

```
\includemedia[<options>]{<poster text>}{  
  <main Flash (SWF) file or URL | 3D (PRC, U3D) file>}
```

The last argument, `<main Flash (SWF) file or URL | 3D (PRC, U3D) file>`, is the main interactive application to be inserted into the PDF. In the case of Flash, this can be a local SWF file, or a URL, such as a YouTube video player. A local file will become part of the final PDF file, while Flash content from a URL requires an internet connection when the user activates it in Adobe Reader. A URL must be fully qualified, i. e., starting with either ‘`http[s]://`’ or ‘`ftp://`’. As for 3D content, Adobe Reader only supports U3D or PRC files embedded in the PDF; they cannot be loaded or streamed during runtime. The most frequent use of `\includemedia` will likely be embedding video or sound files for playback in Adobe Reader. For this we need some media player, which is an SWF file we embed as our main application. It will be configured to load, upon activation, a particular video or sound file that was embedded as a resource into the PDF or is to be streamed from the internet. This will be shown later. Note that a local file (main application or resource) will only once be physically embedded in order to keep the final PDF file size small. If the same file (identified by MD5 checksum) appears in other `\includemedia` commands, only a reference will be inserted that points to the same storage location in the PDF.

Argument `<poster text>` defines the size of the rectangular region of the document page in which the media will be displayed. Moreover, `<poster text>` will be shown in case the media has not been activated. `<poster text>` can be anything that L<sup>A</sup>T<sub>E</sub>X can typeset, such as an `\includegraphics` command serving as a poster image, a PGF/TikZ/PSTricks inline graphics or just ordinary text. Alternatively, `<poster text>` can be left blank in which case the size of the media rectangle should be set with options ‘`width`’, ‘`height`’ and optionally with ‘`depth`’. If sizing options ‘`width`’ and ‘`height`’ are given *together* with `<poster text>`, `<poster text>` will be shrunk or stretched to fit exactly into the rectangle defined by the options, possibly changing the original aspect ratio of the poster text. On the other hand, if only one of ‘`width`’ or ‘`height`’ is given, the other dimension of `<poster text>` is scaled such that the original aspect ratio is preserved.

A list of directories where T<sub>E</sub>X searches for media and resource files can be set-up by means of

```
\addmediapath{<directory>}
```

This command appends one directory at a time to the search list. To specify more directories, just use it repeatedly. The path separator is always ‘/’, independent from the operating system.

The following section explains all command options provided. They are passed to the media inclusion command as a comma separated list enclosed in a pair of square brackets.

## 5.2 Command options

A subset of the command options (see Sect. 4) can also be used as package options, which lets them apply to all embedded media. Some of the options listed here are meaningful only for a specific media type (either Flash or 3D), which will be noted explicitly if not obvious. Dedicated sections covering Flash, video and sound as well as 3D inclusion will follow later on in this document.

```
width=<width>,
height=<height>,
depth=<depth>
```

Resize the media playback area, overriding the original dimensions of the `<poster text>` argument. Option `'depth'` specifies how far the playback area should extend below the base line of the running text. If only one of `'width'` or `'height'` is given, the other dimension is scaled to maintain the aspect ratio of `<poster text>`. Any valid T<sub>E</sub>X dimension is accepted as a parameter. In addition, the length commands `\width`, `\height`, `\depth` and `\totalheight` can be used to refer to the original dimensions of `<poster text>`.

```
label=<label text>
```

The media annotation is given a label, `<label text>`, which should be unique. Labelled media annotations can be targeted by the media actions of a control button (see description of the `\mediabutton` command in Sect. 5.3). Moreover, a reference to the RichMedia Annotation object (of type `'AnnotRichMedia'`) is assigned to the JavaScript variable `annotRM['<label text>']` in order to facilitate its access in JavaScript. Note that the JavaScript reference is known only after the first opening of the page containing the media.

```
scale=<factor>
```

Scales the playback area by `<factor>`.

```
addresource=<local file>,
addresource=<another local file>,
...
```

Every invocation of this option embeds another local file that is required to run the main Flash application or 3D file (last argument of `\includemedia`). Typically, this option is used to embed video files, media player skins, XML files (such as databases), additional objects to appear in a 3D scene etc. If an already embedded file is needed in another `\includemedia` command, this option must be given there again. However, the file in question will only once be physically embedded in order to keep the PDF file small.

```
flashvars={<some_var=some_val&another_var=another_val&...>}
```

(Flash only) Usually, Flash applications can be configured via ActionScript (AS) variables the programmer of the application has made visible from outside.

A typical use would be to set the video source of a media player to point to an embedded MP4 file or to a live stream, or to set the speaker volume for playback of an MP3 file. The argument of the `flashvars` option is a list of `<AS variable>=<value>` pairs separated by `'&'` and enclosed in a pair of braces `{...}`.

Note: If a variable is to be set to point to an embedded resource, the value of the variable must be given in exactly the same way as with the `'addressource'` option. Otherwise the name of the embedded file cannot be resolved. For example,

```
addressource=path/to/video.mp4
```

implies

```
flashvars={vid=path/to/video.mp4&...}
```

if, for a particular media player, the video source is set through ActionScript variable `'vid'`.

(Note for 3D) Resource files used in 3D scenes cannot be loaded by means of ActionScript variables. This must be done by 3D JavaScript during activation of the 3D scene in the Reader. 3D JavaScript can be attached using option `'add3Djscrip'`, see below.

```
activate=onclick | pageopen | pagevisible
```

Decides on how to activate the media annotation. `'activate=onclick'` is default behaviour and does not need be given explicitly; embedded media is activated when the user clicks on it or by a JavaScript. It is recommended to provide a poster image with the `<poster text>` argument in that case. `'pageopen'` and `'pagevisible'` automatically activate the media when the page becomes visible; `'pagevisible'` is better for two-up and continuous page display.

```
deactivate=onclick | pageclose | pageinvisible
```

Decides on how to de-activate the media annotation. `'deactivate=pageclose'` is default behaviour and does not need be given explicitly; media is automatically de-activated when the user leaves the page containing the media. `'pageinvisible'` is similar, but may be better for two-up and continuous page display. Setting `'deactivate=onclick'` requires user interaction for de-activating the media, either by right-click and choosing *'Disable Content'* or by a JavaScript.

```
draft
```

```
final
```

With `'draft'` the media is not embedded. Instead, a box is inserted that has the dimensions of `<poster text>`, subject to the resizing options `'width'`, `'height'`, `'depth'` and `'scale'`. Option `'final'` does the opposite as it forces the media to be embedded. Both options can be used to reduce compilation time during authoring of a document. To get the most out of them it is recommended to set `'draft'` globally as a package or class option and to set `'final'` locally as a command option of the media annotation that is currently worked on. After the document has been finished, the global `'draft'` option can be removed.

```
playbutton[= fancy | plain | none]
```

```
noplaybutton
```

By default, a transparent play button is laid over the inactive media annotation to draw the reader's attention to the embedded multimedia content. It is provided in two versions, 'fancy' and 'plain', but only 'plain' is available in the X<sub>Y</sub>L<sup>A</sup>T<sub>E</sub>X workflow. The default setting is to try the 'fancy' version. 'noplaybutton' or 'playbutton=none' disable the play button overlay.

`windowed[= false | [<width>x<height>][@<position>] ]`

The media is played in a floating window, instead of being played in an embedded fashion. The floating window size is specified via the optional argument `<width>x<height>`, where `<width>` and `<height>` are given in pixels (integer numbers without unit). If the size is not given, a default size is guessed from the annotation size. Optionally, the position of the floating window on the screen can be specified through `@<position>`, where `<position>` may assume one of 'tl', 'cl', 'bl', 'bc', 'br', 'cr', 'tr', 'tc' or 'cc'. The position specifiers have the following meaning:

tl	tc	tr
cl	cc	cr
bl	bc	br

Default window position is 'cc', that is, centred on the screen. 'false' can be set to override a global setting via package options.

#### **transparent**

Indicates whether underlying page content is visible through transparent areas of the embedded media. Default is 'transparent=false'; media artwork is drawn over an opaque background prior to composition over the page content.

#### **passcontext**

(Flash only) If set, user right-clicks are passed through to the context menu of the embedded Flash application, replacing the default Adobe Reader context menu. Useful for cases where the Flash programmer provided additional functionality through the context menu of his application.

#### **3Dtoolbar**

Indicates whether a 3D toolbar should be shown in the Reader on top of the embedded 3D model.

#### **3Dnavpane**

If set, the 3D navigation pane displaying the 3D Model Tree becomes visible in the Reader when the content is initially activated.

`3Dcoo=<x> <y> <z>`

`<x> <y> <z>` specify the positional vector  $\overrightarrow{COO}$  of the centre of orbit of the virtual camera. Real numbers in fixed and floating point notation are accepted.

`3Dc2c=<x> <y> <z>`

`<x> <y> <z>` specify a direction vector  $\overrightarrow{C2C}$  of arbitrary length, originating in the centre of orbit and pointing to the virtual camera. Real numbers in fixed and floating point notation are accepted.

`3Droll=<roll>`



Prescribes an initial camera roll around the optical axis (in clockwise direction, if <roll> is greater than zero); measured in degrees and given as fixed or floating point real number.

**3Dc2w=<12 element camera-to-world matrix>**

This option directly sets the camera-to-world transformation matrix according to the PDF specification. This is an expert option to be used *instead* of the ‘3Dc2c’, ‘3Dcoo’ and ‘3Droll’ options. Only fixed point real numbers are accepted.

**3Dpsob=Min | Max | W | H**

Expert option which directly sets either the /PS entry in the case of perspective projection or the /OB entry in the case of orthographic projection to one of the four possible values. Default value is Min.

**3Droo=<r>**

<r> is a positive fixed or floating point number specifying the radius of orbit *ROO* of the virtual camera. Good values can be found by means of the ‘3Dmenu’ option.











**3Daac=<angle>**

This option sets the aperture angle of the camera, measured in degrees, for the perspective view mode. Fixed and floating point real numbers between 0 and 180 are admissible. A sensible value of 30 is pre-set by default. Larger values can be used to achieve wide-angle or fish-eye effects. See example 8 in section 7.1. This option excludes the use of the ‘3Dortho’ option.

**3Dortho[=<orthographic scaling factor>]**

Switches from the default perspective to orthographic view mode. In orthographic view, the 3D object is parallelly projected onto the virtual camera chip. The projected image is scaled by <orthographic scaling factor> before reaching the camera chip; default value is 1. The optimal value for the scaling factor is given by  $1/D$ , where  $D$  is the diameter of the smallest enclosing sphere of the 3D object in World coordinate units. Fixed and floating point real numbers are accepted. The camera should be positioned outside the 3D object. For this, the radius of orbit (option ‘3Droo’) should be greater than  $D/2$ . Good values for orthographic scaling and orbital radius can easily be found by means of the ‘3Dmenu’ option. Option ‘3Dortho’ excludes the use of the ‘3Daac’ option.

**3Dmenu**

Mainly used during document authoring. Adds three entries, ‘*Generate Default View*’, ‘*Get Current View*’ and ‘*Cross Section*’ to the context (right-click) menu of an activated 3D annotation. Moreover, it allows single parts or part groups of the scene to be scaled, translated and rotated against the remaining scene objects using the keyboard. Their new position can be saved in the current view (‘*Get Current View*’). At first, parts to be modified must be highlighted by clicking either into the scene or into the 3D Model Tree (the part’s bounding box becomes visible). Then, arrow keys ,  let the part spin around the vertical axis, and ,  tilt against it. In order to spin parts around their local up-axis, keep  pressed while using  and . Keys ,  + ,

`[Y]`, `[↑]+[Y]`, `[Z]`, `[↑]+[Z]` translate the selected part along the World axes, and `[S]`, `[↑]+[S]` scale the part.

‘*Generate Default View*’ computes optimal camera settings such that the visible parts of the 3D scene fit tightly into the viewing area. The result is printed, formatted as a list of `\includemedia` options, into the JavaScript console. The calculation is based on the 3D object size and its position in the World coordinate system as well as the current viewing mode (perspective or orthographic).

‘*Cross Section*’ is a toggle switch to add or remove a cross section to or from the current view. If a part of the 3D scene was previously selected, the central rotating point of the section plane is put into the part’s centre, otherwise into the target point of the camera. The section plane can be rotated around the vertical axis and tilted against its upright position using the arrow keys `[←]`, `[→]`, `[↑]` and `[↓]`. Keys `[X]`, `[↑]+[X]`, `[Y]`, `[↑]+[Y]`, `[Z]`, `[↑]+[Z]` move the section plane along the World axes, and `[S]`, `[↑]+[S]` scale its size.

‘*Get Current View*’ writes camera settings, any part alterations, an optional cross section as well as part and scene rendering attributes of the current view into the JavaScript console. The output is a readily formatted **VIEW** section to be inserted into or appended to a file of predefined views. See option ‘3Dviews’. All settings reachable via the ‘*Part Options*’ and ‘*Viewing Options*’ context menu items are written to the **VIEW** section.

**3Dbg=<r> <g> <b>**

This option sets the background colour of the 3D scene. Only fixed point real numbers in the range from 0 to 1 are allowed for the colour components. Option ‘transparent’ may not be set at the same time.

**3Dlights=<lighting scheme>**

Sets the default lighting scheme. The following values are honoured: ‘None’, ‘White’, ‘Day’, ‘Night’, ‘Hard’, ‘Primary’, ‘Blue’, ‘Red’, ‘Cube’, ‘CAD’, ‘HeadLamp’. The default is to use the lighting scheme as specified within the 3D artwork.

**3Drender=<render mode>**

Sets the default render mode. The following values are honoured: ‘Solid’, ‘SolidWireframe’, ‘Transparent’, ‘TransparentWireframe’, ‘BoundingBox’, ‘TransparentBoundingBox’, ‘TransparentBoundingBoxOutline’, ‘Wireframe’, ‘ShadedWireframe’, ‘HiddenWireframe’, ‘Vertices’, ‘ShadedVertices’, ‘SolidOutline’, ‘Illustration’, ‘ShadedIllustration’.

**3Dpartsattrs=restore | keep**

When the user selects another view from the list of predefined views (see option ‘3Dviews’), attributes of individual parts, such as opacity, visibility, render mode, translation in space, which all can be set from within the Reader or by means of a file of predefined views, are reset to their original states as defined in the embedded 3D file, before any new part settings are applied. This default behaviour can be overridden by ‘3Dpartsattrs=keep’. This will preserve current part attributes when the user selects another predefined 3D view in the Reader.

**3Dviews=<views file>**

Instead of or in addition to the default view (options ‘3Dcoo’, ‘3Dc2c’, ‘3Droll’, ‘3Droo’, ‘3Daac’, ‘3Dortho’), further *named views* can be predefined in an auxiliary file <views file>. Besides the virtual camera position, it is possible to adjust the rendering attributes, such as visibility and transparency, as well as position and scaling of every single part in the 3D scene. Moreover, background colour and scene lighting can be set individually for every view. The additional views can later be selected either from a drop down list in the tool bar that is associated with the activated 3D object in the Reader or from the context menu of the 3D object.

The file <views file> is structured into view sections, one for every view:

```
VIEW[=<optional name>]
  C00=<x> <y> <z>
  C2C=<x> <y> <z>
  ROLL=<roll>
  % C2W=<camera-to-world matrix> % instead of C00, C2C and ROLL
  R00=<roo>
  AAC=<aac>
  % ORTHO[=<orthographic scaling factor>] % instead of AAC
  BGCOLOR=<r> <g> <b>
  RENDERMODE=<render mode>
  LIGHTS=<lighting scheme>
  CROSSSECT
    CENTER=<x> <y> <z>
    NORMAL=<x> <y> <z>
  END
  PARTSATTRS=keep
  PART=<part name as in the Model Tree (required, optional if UTF16NAME present)>
    UTF16NAME=<part name as hex encoded Unicode string>
    VISIBLE=true | false
    OPACITY=<part opacity>
    RENDERMODE=<part render mode>
    TRANSFORM=<12 element transformation matrix>
  END
  PART=<...>
  ...
  END
  etc.
END

VIEW
  ...
END

etc.
```

A view section starts with the keyword **VIEW**, optionally followed by a name for the view, and ends with the keyword **END**. If no name is given to the view, a default one is created, consisting of ‘View’ followed by the number of the current **VIEW** section in the file. A **VIEW** section may contain optional entries for setting

the camera position and global rendering attributes of the scene, a **CROSSECT** subsection as well as **PART** subsections for setting rendering and other attributes of parts individually. Table 1 lists the entries in a **VIEW** section.

Part sub-sections are opened by **PART=<part name>** and closed by **END**. There may be as many part subsections as there are parts in a 3D scene. Table 2 lists the possible entries in a **PART** sub-section. All entries are optional. However, a **UTF16NAME** entry is recommended, as the part name may contain non-ASCII characters. The value of the **UTF16NAME** key is the part name as a hex-encoded Unicode string. If **UTF16NAME** is not used, the part name in the 3D file must be entirely composed of ASCII characters. In that case, **<part name>** is mandatory and must match the part name as indicated in the 3D Model Tree of the 3D object (accessible via right-click onto the model in the Reader). The part can be scaled and repositioned by means of a **TRANSFORM** entry which takes a 12-element transformation matrix as its value. Remaining entries in a part sub-section control the visual appearance of the part.

A view section may contain at most one **CROSSECT** sub-section. It inserts a section plane at a definite position and orientation in the 3D space, controlled by optional **CENTER** and **NORMAL** entries. See Table 3 for explanation.

The views file can be commented. As usual, comments start with the percent sign.

To facilitate the creation of a views file, option ‘3Dmenu’ can be added to **\includemedia** (see above). It creates context (right-click) menu entry ‘Get Current View’ which outputs a complete **VIEW** section corresponding to the current view of the 3D object in the Reader, including camera position, an optional cross section, and all part and viewing options that can be modified via the 3D toolbar (option ‘3Dtoolbar’) or the context menu of the 3D object (entries ‘*Part Options*’, ‘*Viewing Options*’). Hence, apart from tweaking one or another entry, there should be no need for writing views files by hand.

**3Dplaytype=linear | oscillating**

According to the PDF specification, embedded keyframe animations can be played in two ways. If set to ‘**linear**’, keyframe animations are driven linearly from beginning to end, while ‘**oscillating**’ lets the animation play in a forth-and-back manner.

**3Dplaycount=<integer number>**

A non-negative **<integer number>** represents the number of times the animation is played. A negative integer indicates that the animation is infinitely repeated. This value is ignored if option **3Dplaytype** is not set.

**3Dplayspeed=<positive number>**

This option can be used to adjust the keyframe animation speed. A value of ‘1’ corresponds to the default speed defined in the 3D file.

**add3Djscript=<3D JavaScript file>,  
add3Djscript=<another 3D JavaScript file>,  
...**

Things like animation, lighting, background of 3D objects etc. may also be script driven. Every invocation of ‘add3Djscript’ associates another JavaScript file with the 3D object. Upon activation of the 3D object, the scripts are executed once in the order of their inclusion. Refer to the Acrobat 3D JavaScript Reference [3] for syntax details. The following 3D JavaScript loads an image file that was attached by ‘addresource=images/sunset.jpg’ and uses it as the scene background.

```

sunset = new Image(new Resource('pdf://images/sunset.jpg'));
reh = new RenderEventHandler();
reh.onEvent = function(event) {
    runtime.removeEventHandler(this);
    event.canvas.background.image=sunset;
}
runtime.addEventHandler(reh);

```

For convenience, subdirectory ‘javascript’ of the ‘media9’ installation contains three 3D JavaScript files which may come in handy at times: ‘animation.js’ enables embedded keyframe animation in 3D files; ‘3Dspintool.js’ enables the Spin tool of the 3D plugin for easier rotating the 3D object with the mouse; ‘asylabels.js’ adds ‘billboard behaviour’ to text labels in Asymptote ( $\geq v2.17$ ) generated PRC files for improved visibility; text labels always face the camera while rotating the 3D object with the mouse.

Table 1: Entries in a **VIEW** section.

key	type	remarks
C00	three numbers	centre of orbit, see option ‘3Dcoo’
C2C	three numbers	centre of orbit to camera vector, see option ‘3Dc2c’
R00	number	radius of orbit, see option ‘3Droo’
C2W	12 numbers	camera-to-world transformation matrix, see option ‘3Dc2w’
AAC	number	camera aperture angle, see option ‘3Daac’
ORTHO	number (optional)	enables orthographic view, see option ‘3Dortho’
PSOB	string	expert setting, see option ‘3Dpsob’
ROLL	number	camera roll, see option ‘3Droll’
BGCOLOR	three numbers	3D scene background colour (RGB), see option ‘3Dbg’
RENDERMODE	string	render mode of the 3D object, see option ‘3Drender’
LIGHTS	string	lighting scheme, see option ‘3Dlights’
PARTSATTRS	string	allowed values are ‘keep’ and ‘restore’; decides on whether to restore or not original part attributes before applying new ones from this view; see option ‘3Dpartsattrs’
PART (sub-section)	string	part name as in the 3D Model Tree; name argument is optional if a UTF16NAME entry is present in the sub-section opened by a PART keyword, otherwise required; see Table 2 for list of possible entries
CROSSSECT (sub-section)	–	see Table 3 for list of possible entries

Table 2: Entries in a **PART** sub-section.

key	type	remarks
<b>UTF16NAME</b>	hex string	part name in UTF-16 (aka Unicode), encoded as a hexadecimal string; optional, but useful for part names composed of non-latin characters;
<b>VISIBLE</b>	boolean	a flag ( <b>true</b> or <b>false</b> ) indicating the visibility of this part
<b>OPACITY</b>	number	a number between 0.0 and 1.0 specifying the opacity of this part
<b>RENDERMODE</b>	string	rendermode of this part, overrides global <b>RENDERMODE</b> value in parent <b>VIEW</b> section, see option <b>3Drender</b>
<b>TRANSFORM</b>	12 numbers	transformation matrix defining the part's position and scaling

Table 3: Entries in a **CROSSECT** sub-section.

key	type	remarks
<b>CENTER</b>	three numbers	central point coordinates of the section plane
<b>NORMAL</b>	three numbers	normal vector coordinates of the section plane pointing into the cut-off region

### 5.3 Control buttons

`\mediabutton[<options>]{<normal button text or graphic>}`

This command inserts a clickable button for media control. Actions to be performed are specified through options ‘mediacommand’, ‘3Dgotoview’ and ‘jsaction’. By using these options repeatedly and in any combination, several actions can be bound to one media button, and one media button can be used to control several media at the same time. Media actions are started in the given order but performed in parallel, because they do not wait for each other to finish. The target of an action is specified via the label that was also given to a particular media by the ‘label’ option of ‘\includemedia’. Individual button faces can be defined for the ‘mouse-over’ and ‘mouse-button-down’ events using the ‘overface’ and ‘downface’ options. Without options, the button produced does nothing. The options provided are as follows:

`overface=<mouse-over text or graphic>`

If specified, the media button changes its appearance when the mouse pointer is moved over it. Without this option, the button appearance does not change. An `\includegraphics` command may need to be enclosed in braces.

`downface=<mouse-button-down text or graphic>`

If specified, the media button changes its appearance when the mouse button is pressed while the pointer is over it. An `\includegraphics` command may need to be enclosed in braces.

`tooltip=<tip text>`

A box with `<tip text>` is shown when the mouse pointer is moved over the button.

`3Dgotoview=<label text>[:<view specification>]`

Selects a view from the list of predefined views associated with a 3D media inclusion (see option ‘3Dviews’). The target media is specified by `<label text>`, as defined by the ‘label’ option of ‘\includemedia’. `<label text>` alone without a view specification simply activates the 3D object if not yet activated. `<view specification>` which is separated from the label by a colon (:) can be one of the following: an integer specifying the zero-based index into the list of views in the 3D views file; one of ‘D’, ‘F’, ‘L’, ‘N’, ‘P’ indicating the default, first, last, next or previous view in the list of views; a string delimited by ‘(’ and ‘)’ matching the name of a view as specified by the ‘VIEW=...’ entry in the views file. The option can be given several times to simultaneously change the view in more than one 3D inclusion. However, it cannot be used to create an animation effect within the same 3D inclusion, because `3Dgotoview` actions are executed in parallel.

`mediacommand=<label text>[:<command> [<arg1> <arg2> ...]]`

A media command `<command>`, with arguments if required, is sent to a media inclusion identified by `<label text>`, as defined by the ‘label’ option of ‘\includemedia’. `<label text>` alone without a command specification simply activates the media, if not yet activated. The option can be multiply used within the same button to target different media inclusions at the same time or to



execute several commands for the same media. Depending on the type of the target media (3D or Flash), `<command>` is either the name of a JavaScript function defined in a 3D JavaScript file associated with the 3D media (see option `'add3Djavascript'`) or the name of an ActionScript function that was exposed by the embedded Flash file. ActionScript functions are exposed to the scripting context of the hosting document by using the `ExternalInterface` call within the Flash file. Arguments to be passed to `<command>` must be separated by spaces and the whole list be enclosed in '[' and ']'. Arguments can be of Boolean type (`true`, `false`), numbers (integer, reals) and strings. String arguments must be passed as (`string arg`), i.e. enclosed in parentheses, while numbers and Booleans are passed as they are. Of course, the number of arguments and their types must match the definition of the function to be called. Media players `VPlayer.swf` and `APlayer.swf` shipping with `media9` expose a number of ActionScript functions that can be used with this option (see Tab. 6). `<command> [<arg1> <arg2> ...]` must be enclosed in braces if there are embedded equals signs or commas.

```
jsaction=[<label text>:]{<JavaScript code>}
```

The JavaScript code is executed in the context of the document's instance of the JavaScript engine (there is one instance of the JavaScript engine per open document in Adobe Reader). `<JavaScript code>` is required and must be enclosed in braces. Unlike media actions defined with options `'mediacommand'` and `'3Dgotoview'`, the JavaScript action defined here is not targeted at a particular embedded media and can be used to run arbitrary code. Therefore, `<label text>` is optional. If provided, it must be separated from `<JavaScript code>` by a colon. However, it is recommended to provide a label text. It ensures that `annotRM['<label text>']` is a valid JavaScript reference to the `AnnotRichMedia` object. `annotRM['<label text>']` can be used to get access to the global context of the annotation's instance of the 3D JavaScript engine (there is one instance of the 3D JavaScript engine per activated RichMedia Annotation with 3D content). The 3D JavaScript context of a 3D model can be accessed as `annotRM['<label text>'].context3D`. Refer to the Acrobat 3D JavaScript Reference [3] for details on built-in JavaScript objects that are available in the 3D context. The `annotRM['<label text>'].callAS()` method may be used as an alternative to the `'mediacommand'` option in order run exposed ActionScript functions of an embedded Flash file. See [4] for details.

`draft`

`final`

See above.

## 6 Embedding Flash, video and sound

A YouTube video clip, as shown in Fig. 1, may serve as a basic example of loading Flash content from a URL to be displayed in an embedded fashion in a PDF document. Indeed, a YouTube clip is nothing more than a small SWF file which loads a video stream and other necessary resources, such as user controls and a player skin from a remote server. It can be configured via ActionScript variables to play several videos in a row, to play a video in a loop etc. Player parameters are documented on [http://code.google.com/apis/youtube/player\\_parameters.html](http://code.google.com/apis/youtube/player_parameters.html) and can be passed to the player using either the ‘flashvars’ option, as in the example, or appended to the URL string after the video ID. A question mark ‘?’ must be put between the video ID and the parameter string. Some of the documented parameters, such as ‘rel’, seem to have an effect only if they are passed as part of the URL.

```
\includemedia[
  width=0.6\linewidth,height=0.3375\linewidth,
  activate=pageopen,
  flashvars={
    modestbranding=1 % no YT logo in control bar
    &autohide=1       % controlbar autohide
    &showinfo=0       % no title and other info before start
    &rel=0            % no related videos after end
  }
]{}{http://www.youtube.com/v/w3f-WyDq0Uw?rel=0}
```

Figure 1: A YouTube video as an example of a Flash application loaded from a URL.

Video and sound files are always loaded and then played by a media player application. Three players are installed along with the ‘media9.sty’ package file: two simple players, ‘VPlayer.swf’ for video and ‘APlayer.swf’ for sound, and a fully blown one, ‘StrobeMediaPlayback.swf’, with some fixes to improve its usability. The simple ones are ‘chromeless’ players, that is, they do not have graphical user controls. Nevertheless, interactivity is provided through the keyboard, as summarized in Table 4, and through left mouse button press and release for playing, pausing and resuming media. ‘VPlayer.swf’ and ‘APlayer.swf’ were compiled, using the open-source Apache Flex SDK [5], from XML source files which reside in the doc/ folder of the package installation. For ‘StrobeMediaPlayback.swf’,

only a patch file is included, as the sources can be downloaded elsewhere.

The improvements of ‘StrobeMediaPlayback.swf’ in comparison to the original version on SourceForge.net are

- fix: video could not be restarted after end of playback if ActionScript variable ‘autoRewind’ is set to ‘false’
- new: first frame of video is shown as default poster instead of black stage
- new: play/pause video by clicking on the stage (as with ‘VPlayer.swf’), useful in a lecture situation

There is no need to copy the installed players into the directory of the document source for embedding. They will be found by L<sup>A</sup>T<sub>E</sub>X without taking any further action.

Like YouTube videos, media players are configured via ActionScript variables which are passed using option ‘flashvars’. Table 5 lists parameters available for ‘VPlayer.swf’ and ‘APlayer.swf’, table 7 for ‘StrobeMediaPlayback.swf’.

Playback of embedded video files is shown in Fig. 2. Besides embedded files, also video streamed from remote servers via HTTP and RTMP protocols is supported, as shown in Fig. 3.

‘VPlayer.swf’ and ‘APlayer.swf’ expose a number of ActionScript functions to the JavaScript engine of Adobe Reader, allowing for playback control of media through push buttons (see Sect. 5.3) and various trigger events. The functions and their calling convention are listed in Table 6. From within JavaScript, these functions can be called using the ‘callAS’ (= call ActionScript) method of the AnnotRichMedia object. As an example, a call to the ‘seek’ function looks like

```
annotRM.myvideo.callAS("seek", 12.3);
```

and a call to the parameter-less function ‘pause’ like

```
annotRM.myvideo.callAS("pause");
```

Both functions calls are sent to the media inclusion that was labelled ‘myvideo’ using the ‘label’ option of \includemedia. An example of playing and pausing a video clip and setting the video source via interactive push buttons is given in Fig. 2.

Sound files and streams in the MP3 format can be played with ‘APlayer.swf’. Fig. 4 contains examples of an audio live stream and a remote MP3 sound file. In one of the sound examples, the player is loaded from a CTAN mirror during runtime because an internet connection is required anyway for streaming the audio. If a local sound file is to be embedded into the PDF this would have to be done in the same way as with the video file in one of the previous examples using the ‘addresource’ option.

Table 4: Keyboard control of media players ‘VPlayer.swf’ and ‘APlayer.swf’. The media must have the focus to have effect. Click onto the media if necessary.















keys	action
	play/pause
 , 	go to start/end
 , 	seek backwards/forwards
 , 	decrease/increase speaker volume
	mute/unmute
 +  ,  +  ,  + 	(APlayer.swf only) change sound speaker balance

Table 5: Parameters (ActionScript variables) for media players ‘VPlayer.swf’ and ‘APlayer.swf’ shipping with media9. Parameters are passed as a ‘&’-separated string using ‘flashvars’ option.

parameter	description
<code>source=&lt;file path or URL&gt;</code>	(required) path to embedded media file (see option ‘ <code>addresource</code> ’), or URL (http, rtmp) to online media file
<code>autoPlay=true false</code>	if <code>=true</code> , automatically starts playback after activation (see option ‘ <code>activation</code> ’)
<code>autoRewind=true false</code>	(VPlayer.swf only) if <code>=true</code> , automatically rewind to the first frame after playback has finished; default is ‘ <code>false</code> ’
<code>loop=true false</code>	if <code>=true</code> , media is played in a loop
<code>stepping=true false</code>	(VPlayer.swf only) if <code>=true</code> , the video advances by roughly one frame per mouse click
<code>scaleMode=letterbox none stretch zoom</code>	default: <code>stretch</code> ; determines how to scale the video in order to fit into player
<code>hideBar=true false</code>	(APlayer.swf only) if <code>=true</code> , the progress bar indicating the play position is not shown
<code>volume=&lt;value between 0.0 and 1.0&gt;</code>	sets volume of the sound
<code>balance=&lt;value between -1.0 and 1.0&gt;</code>	(APlayer.swf only) sets balance of sound speakers

Table 6: Exposed ActionScript functions of media players ‘VPlayer.swf’ and ‘APlayer.swf’, that can be called from within media buttons (see Sect. 5.3) or from JavaScript using the ‘callAS’ method of the ‘AnnotRichMedia’ JavaScript object (see [4] for further information).

function	argument	description
<code>play</code>		play media
<code>pause</code>		pause media
<code>playPause</code>		toggle between play and pause
<code>stepping</code>		(VPlayer.swf only) toggle stepping mode (one frame per click)
<code>setSource</code>	string	load another media file (path to file, embedded using option ‘addresource’, or URL)
<code>seek</code>	number	move the play location to a time offset from the beginning of the media; argument measured in seconds
<code>rewind</code>		rewind media to the beginning (without pausing it)
<code>volume</code>	number between 0 and 1	set volume level
<code>balance</code>	number between -1 and +1	(APlayer.swf only) set speaker balance
<code>mute</code>		mute or unmute (toggle) the audio of the media
<code>currentTime</code>		returns current playhead position in seconds; only useful in JavaScript via ‘callAS’ method
<code>duration</code>		returns duration of the media video/sound file currently loaded, only useful in JavaScript via ‘callAS’ method
<code>playing</code>		returns boolean value ‘true’, if the media is currently playing, ‘true’ otherwise; only useful in JavaScript via ‘callAS’ method
<code>muted</code>		returns boolean value ‘true’, if the sound is currently muted, ‘false’ otherwise; only useful in JavaScript via ‘callAS’ method

Table 7: Parameters (ActionScript variables) for ‘StrobeMediaPlayback.swf’ shipping with media9. Parameters are passed as a ‘&’-separated string using ‘flashvars’ option.

parameter	description
<code>src=&lt;file path or URL&gt;</code>	(required) path to embedded media file (see option ‘ <code>addresource</code> ’), or URL (http, rtmp) of online media file
<code>autoPlay=true false</code>	default: <code>false</code> ; if <code>=true</code> , automatically starts playback after activation (see option ‘ <code>activation</code> ’)
<code>autoRewind=true false</code>	default: <code>true</code> ; if <code>=false</code> , keep last frame after end of playback
<code>loop=true false</code>	if <code>=true</code> , media is played in a loop
<code>scaleMode=letterbox none stretch zoom</code>	default: <code>letterbox</code> ; determines how to scale the video in order to fit into player
<code>controlBarMode=docked floating none</code>	default: <code>docked</code> ; determines position and visibility of control bar
<code>controlBarAutoHide=true false</code>	default: <code>true</code> ; automatically hide or not control bar
<code>controlBarAutoHideTimeout=&lt;number [s]&gt;</code>	default: 3; time span before auto-hide
<code>volume=&lt;value between 0.0 and 1.0&gt;</code>	sets volume of the sound
<code>audioPan=&lt;value between -1.0 and 1.0&gt;</code>	default: 0; sets balance of sound speakers
<code>muted=true false</code>	default: <code>false</code> ; mute or not sound

```

\includemedia[
    label=some_dice,
    width=0.6\linewidth,height=0.45\linewidth,
    addresource=random.mp4, %two video files
    addresource=cube.mp4,
    transparent,           %transparent player background
    activate=pageopen,
    flashvars={
        source=random.mp4
        &loop=true          % loop video
        &scaleMode=letterbox % preserve aspect ratio while scaling the video
    }
]{}{VPlayer.swf}

\mediabutton[
    mediacommand=some_dice:playPause,
    overface=\color{blue}{\fbox{\strut Play/Pause}},
    downface=\color{red}{\fbox{\strut Play/Pause}}
]{\fbox{\strut Play/Pause}}
\mediabutton[
    mediacommand=some_dice:setSource [(random.mp4)]
]{\fbox{\strut random.mp4}}
\mediabutton[
    mediacommand=some_dice:setSource [(cube.mp4)]
]{\fbox{\strut cube.mp4}}

```

Figure 2: Example of playing back two different embedded MP4 video files in the same video player instance. The player, ‘VPlayer.swf’, is also embedded in the PDF. Exposed ActionScript functions ‘playPause’ and ‘setSource’ of ‘VPlayer.swf’ (Table 6) are used to set-up media control buttons. Different button faces have been defined for the Play/Pause button.

```
\includemedias[
  width=0.6\linewidth,height=0.3375\linewidth, % 16:9
  activate=pageopen,
  flashvars={
    src=rtmp://streaming.music.indiana.edu:1935/onDemand/mp4:media/%
                                20090327_VarRussianTheme-h264-480.m4v
    &scaleMode=stretch
  }
]{StrobeMediaPlayback.swf}
```

Figure 3: Example of video streamed from an RTMP server. This example uses media player ‘StrobeMediaPlayBack.swf’, physically embedded in the PDF.



```

\includemedia[
  addresource=bird.mp3,
  flashvars={
    source=bird.mp3
    &autoplay=true
  },
  transparent
]{\color{blue}\framebox[0.4\linewidth][c]{Singing bird}}{APlayer.swf}

```

a) Singing bird

```

\includemedia[
  flashvars={
    source=http://mp3.live.tv-radio.com/franceculture%
      /all/franceculturehautdebit.mp3
    &autoplay=true
  },
  transparent
]{\color{blue}\fbox{Listen live to Radio France Culture}}{%
  http://mirrors.ibiblio.org/pub/mirrors/CTAN/macros/latex/%
  contrib/media9/players/APlayer.swf%
}

```

b) Listen live to Radio France Culture

```

\includemedia[
  flashvars={
    source=http://www.openbsd.org/songs/song49.mp3
    &autoplay=true
  },
  transparent
]{\color{blue}\fbox{Listen to OpenBSD 4.9 release song}}{APlayer.swf}

```

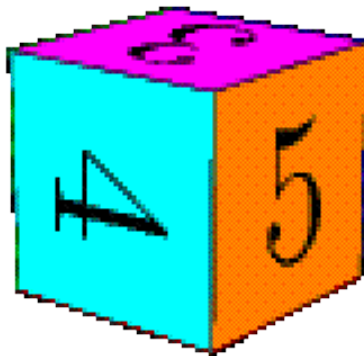
c) Listen to OpenBSD 4.9 release song

Figure 4: Example of (a) embedded sound file, (b) streamed audio and (c) progressively downloaded MP3. ID3 tags ‘title’, ‘artist’ and ‘album’ are displayed if contained in the MP3 stream or file. In (b), the sound player, APlayer.swf, is loaded from a CTAN mirror upon activation.

```

\includemedia[
    %activate=onclick,      % default
    addresource=cube.mp4,
    flashvars={
        source=cube.mp4
        &autoplay=true      % start playing on activation
        &loop=true
    }
]{\includegraphics[height=0.45\linewidth]{cubeposter}}{VPlayer9.swf}

```



```

\includemedia[
    addresource=bird.mp3,
    flashvars={
        source=bird.mp3
        &autoplay=true
    },
    transparent
]{\color{blue}\framebox[0.4\linewidth][c]{Singing bird}}{APlayer9.swf}

```

Singing bird

Figure 5: Video and sound examples that should run in Adobe Reader for Linux up to version 9.4.1. Here, players ‘VPlayer9.swf’ and ‘APlayer9.swf’ are used. Both are compatible with Adobe Flash Player 9 plugin that is bundled with the Reader. Also, the video player needs to be activated by mouse click (which is the default). We provide a poster image that is shown in the inactive state.

## 7 Embedding 3D objects

### 7.1 Introduction

Adobe Acrobat/Reader 7 was the first version to allow for embedding 3-dimensional graphic objects, such as CAD models or 3D scientific data, that can be manipulated interactively by the user. U3D was the first supported format and was mainly developed by Right Hemisphere and Adobe. U3D had some deficiencies and was later replaced by the PRC format after Adobe purchased the original developer, the French company ‘Trade and Technology France’. U3D is still supported, but PRC is preferred as it allows for exact representation of curved surfaces and better compression. Both, U3D and PRC specifications are public [6, 7].

Currently, two open-source software packages are known to export into the PRC file format. The first one is Asymptote [8], which is a descriptive 2D and 3D vector graphics language and interpreter and which uses  $\text{\TeX}$  to typeset labels and equations. It allows for high quality mathematical figures and technical drawings. An impressive gallery of examples can be found on its Web site. The second one is MathGL [9], a library for scientific data visualization. It provides interfaces to a number of programming and scripting languages as well as an interpreter for its own command language ‘MGL’.

MeshLab [11] is an open-source conversion and processing software for 3D mesh data which can import from and export to a number of file formats. Its U3D export filter is based on the open-source ‘Universal 3D Sample Software’ [10].

There are a few options to `\includemedia` which define how the 3D object is positioned within the view port of a virtual camera, or conversely, how the virtual camera is positioned and oriented within a coordinate system, called ‘The World’, which bears the 3D object at a fixed position. Fig. 6 should help to grasp the scenery: The virtual camera is orbiting at a distance of  $ROO$  (option ‘3Droo’) around the centre of orbit, specified by the position vector  $\overrightarrow{COO}$  (option ‘3Dcoo’);  $\angle AAC$  (option ‘3Daac’) is the camera’s aperture angle. The direction vector  $\overrightarrow{C2C}$  (option ‘3Dc2c’) is needed to specify the initial camera position. The camera may be given an initial roll angle (option ‘3Droll’) around its optical axis  $(-1) \cdot \overrightarrow{C2C}$ . Fig. 6 shows the camera parameters for the perspective view mode. Alternatively, the orthographic view mode may be chosen. In orthographic view, the 3D object is parallelly projected onto the virtual camera chip. Before reaching the camera chip, the projected image must be scaled in order to fit onto the chip. Orthographic view can be enabled using the ‘3Dortho’ option which takes the scaling factor as its argument.

Above options define the default view, i. e. the view that is shown initially after activating the 3D object in the Reader. Of course, once activated, the camera position can be changed using the mouse and one can change forth and back between perspective and orthographic viewing modes using the 3D tool bar.

By default, the virtual camera sits at the origin  $(0, 0, 0)$  of the World, looking in the positive  $Y$  direction, i. e. default settings of  $3Droo=0$ ,  $3Dcoo=0\ 0\ 0$  and  $3Dc2c=0\ -1\ 0$  are assumed. (Note that  $\overrightarrow{C2C}$  is the opposite of the optical axis

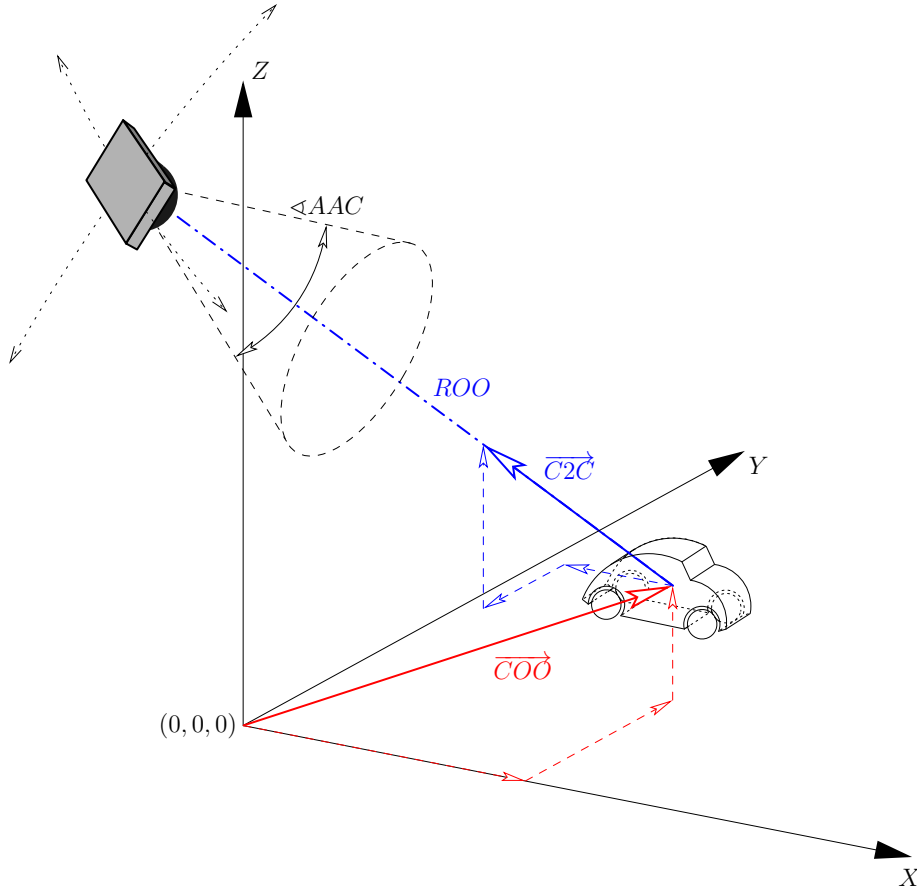


Figure 6: Camera and 3D object in the World System  $XYZ$ ; centre of orbit position vector  $\overrightarrow{COO}$ , centre of orbit to camera direction vector  $\overrightarrow{C2C}$ , radius of orbit  $ROO$ , aperture angle of camera  $\angle AAC$ .

vector.) Thus, in order to get a ‘front view’ of the 3D object it is sufficient to set the radius of orbit, i.e. the distance between camera and object appropriately. Sometimes you may want to adjust the orbital centre, i.e. the target of the camera as well, in particular, if the object is irregularly shaped or if it is not centred around the World origin. Fortunately, it is possible to let the values of the corresponding options be determined automatically. Choosing option ‘3Dmenu’ adds ‘*Generate Default View*’ to the context menu of the activated 3D scene. Selecting this entry calculates and outputs optimal camera settings which can be inserted into the option list of `\includemedia`.

Additional resource files that are needed to render the 3D scene can be embedded using the ‘`addresource`’ option. Typical resources are bitmaps and Flash files (even animated and interactive ones), to be used as materials or scene backgrounds, as well as additional 3D objects in the U3D or PRC file format. The allowed file formats of *bitmapped* image files depend on the  $\text{\LaTeX}$  workflow.  $\text{\LaTeX} \rightarrow \text{dvips} \rightarrow \text{ps2pdf}$ /Distiller accepts PS and EPS files; pdf $\text{\LaTeX}$  accepts

PNG, JPEG and JBIG2;  $(X_{\text{H}})\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} \rightarrow (\text{x})\text{dvipdfmx}$  accepts PNG and JPEG. 3D JavaScript is necessary to load these resources upon activation. 3D JavaScript files are attached using the ‘`add3Djscrip`t’ option.

Below, several examples of embedded 3D files are shown. The first one, Fig. 7 is a PRC file generated with Asymptote. Note the text labels always facing the camera thanks to the attached 3D JavaScript file ‘`asylabels.js`’. The second example, Fig. 8, demonstrates the use of a views file which defines additional named views of the 3D object. Moreover, the possibilities of the extended 3D context menu can be evaluated. They were enabled by adding the ‘`3Dmenu`’ option to `\includemedia`. All part and scene rendering attributes that can be changed via the ‘*Part Options*’ and ‘*Viewing Options*’ menu entries, as well as a cross section to be added with the ‘*Cross Section*’ menu entry can be saved into a new view (‘*Get Current View*’). Position, orientation and scaling of individual parts and of the cross section can be changed using the keyboard (keys `←`, `→`, `↑`, `↓`, `X`, `↑`+`X`, `Y`, `↑`+`Y`, `Z`, `↑`+`Z`, `S`, `↑`+`S`). The third example, Fig. 9, shows an animated 3D object. The animation itself and the functions called by pressing the controls are defined in a 3D JavaScript file attached to the model.

```

\includemedia[
  width=0.8\linewidth,height=0.8\linewidth,
  add3Djscrip=asylabels.js, %upright text labels
  add3Djscrip=3Dspintool.js, %let scene rotate about z-axis
  % 3Dcoo, 3Droo values found with 'Generate Default View' from
  % context menu
  3Dmenu,
  3Dc2c=4 2 3,
  3Dcoo=4.413303375244141 2.195653200149536 -0.000011444091796875,
  3Droo=429.49035778293376,
]{\includegraphics{epixposter}}{epix.prc}

```

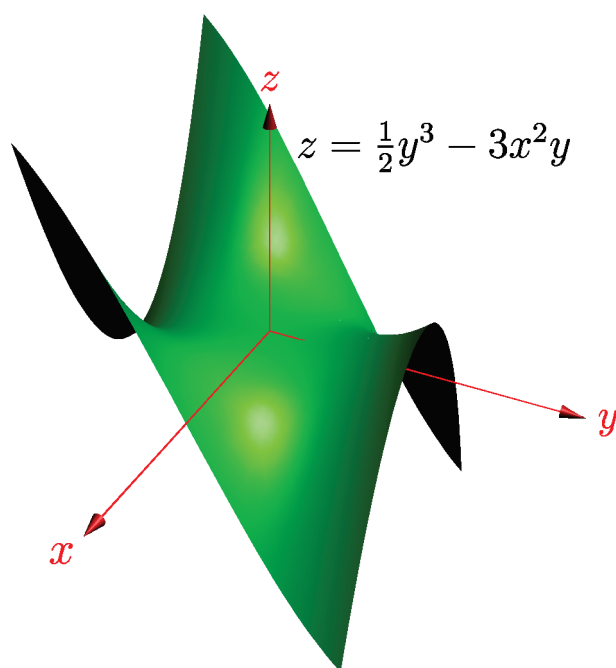


Figure 7: Embedded PRC file produced with Asymptote, making use of convenience 3D JavaScripts ‘asylabels.js’ and ‘3Dspintool.js’ mentioned above.

```

\includemedia[
  label=dice,
  width=0.5\linewidth,height=0.5\linewidth,
  activate=pageopen,
  3Dtoolbar, 3Dmenu,
  3Dviews=dice.vws,
]{}{dice.u3d}

\mediabutton[3Dgotoview=dice:N]{\fbox{Next view}}
\mediabutton[3Dgotoview=dice:(Back)]{\fbox{View 'Back'}}
\mediabutton[3Dgotoview=dice:5]{\fbox{6th view in the list}}

```

Contents of 'dice.vws':

```

VIEW=Front
  R00=27
END
VIEW=Back
  R00=27
  C2C=0 1 0
END
VIEW=Left
  R00=27
  C2C=-1 0 0
END
VIEW=Right
  R00=27
  C2C=1 0 0
END
VIEW=Top
  R00=27
  C2C=0 0 1
END
VIEW=Bottom
  R00=27
  C2C=0 0 -1
END
VIEW=Fish Eye at Centre
  AAC=130
END

```

Figure 8: Embedded U3D file, based on a VRML model by Peter Whitehouse, <http://www.wonko.info/vrml/index.htm>; conversion to U3D was done using DeepExploration<sup>®</sup>[12]. The file 'dice.vws' provides predefined views. Buttons are created with \mediabutton using the '3Dgotoview' option.

```

\includemedia[
  label=malte,
  width=0.5\linewidth,height=0.5\linewidth,
  activate=pageopen,
  3Dmenu,
  3Dc2c=1 1 1,
  3Dcoo=-0.001042630523443222 1.4577869224116568e-19 0.028235001489520073,
  3Droo=0.2604540212188131,
  add3Djscript=malte.js
]{\malte.u3d}

\mediabutton[
  jsaction=malte:{annotRM['malte'].context3D.cntrClockWise();}
]{\includegraphics[height=1.44em]{boutona}}
\mediabutton[
  jsaction=malte:{annotRM['malte'].context3D.pause();}
]{\includegraphics[height=1.44em]{boutonb}}
\mediabutton[
  jsaction=malte:{annotRM['malte'].context3D.clockWise();}
]{\includegraphics[height=1.44em]{boutonc}}
\hspace{1em}
\mediabutton[
  jsaction=malte:{annotRM['malte'].context3D.scaleSpeed(1/1.1);}
]{\includegraphics[height=1.44em]{boutond}}
\mediabutton[
  jsaction=malte:{annotRM['malte'].context3D.origSpeed();}
]{\includegraphics[height=1.44em]{boutone}}
\mediabutton[
  jsaction=malte:{annotRM['malte'].context3D.scaleSpeed(1.1);}
]{\includegraphics[height=1.44em]{boutonf}}

```

Figure 9: Animated U3D example of a Maltese drive contributed by Jean-Luc Chesnot. The animation and the functions called in the JavaScript actions of the media buttons are defined in the JavaScript file ‘malte.js’.



## 7.2 3D quick-start guide

1. Insert the 3D object with default camera settings and with extended context menu enabled (option '3Dmenu'):

```
\includemedia[  
    width=0.5\linewidth,height=0.5\linewidth,  
    activate=pageopen,  
    3Dmenu  
]{\myfile.u3d}
```

2. Compile the document.
3. Open the PDF document in Adobe Reader and go to the page containing the 3D object. Select '*Generate Default View*' from the 3D context menu (right mouse click) and wait for the JavaScript console to pop up. Optionally, drag the object with the mouse to change the viewpoint of the camera and select '*Generate Default View*' again. This will re-adjust the distance between camera and target to fit all visible parts tightly into the viewport. The options printed into the console are updated accordingly.
4. Copy the camera settings (3Droo=..., 3Dcoo=..., etc.) from the console into the option list of \includemedia.
5. Compile the document again.

*Optional steps* (option '3Dmenu' required):

6. Additional, named views; cross sections; rescaled, repositioned parts:
  - a) Open a text file, e.g. 'myviews.vws', to be populated with additional views of the 3D object.
  - b) Manipulate the 3D object using the mouse (camera position) and via 3D context menu items '*Part Options*' and '*Viewing Options*' (visibility, rendering attributes, background etc.); the camera target can be moved into the centre of a single part via '*Part Options*'→'*Zoom to Part*'.
  - c) Add a cross section plane (select '*Cross Section*' from the 3D context menu), adjust its position using the keyboard; keyboard keys are given here.
  - d) Adjust scaling and position of individual parts using the keyboard; keyboard keys are given here.
  - e) Re-adjust the camera distance using either '*Generate Default View*' or '*Part Options*'→'*Fit Visible*'.
  - f) When you are done, select '*Get Current View*' to get the VIEW section, readily formatted for insertion into the views file. Repeat steps (a)–(f) to get any number of views you want to define. The views file can be edited manually to give meaningful names to the views (change the value of the VIEW key), or to further tweak camera settings, opacity, part options etc.

g) Attach the views file with option ‘3Dviews’:

```
\includemedia[
  width=0.5\linewidth,height=0.5\linewidth,
  activate=pageopen,
  3Dviews=myviews.vws,
  3Dmenu
]{\myfile.u3d}
```

If you are satisfied with the predefined views in the views file, the default view first specified through the options of `\includemedia` can be deleted. The first view in the views file becomes the default view then.

7. Associate any number of 3D JavaScript files with the 3D object:

```
\includemedia[
  width=0.5\linewidth,height=0.5\linewidth,
  activate=pageopen,
  add3Djscript=somescript.js,
  add3Djscript=otherscript.js,
  3Dviews=myviews.vws,
  3Dmenu
]{\myfile.u3d}
```

A few 3D JavaScript files ready to be used are already installed along with ‘media9.sty’, see above.

## 8 Caveats

Large media files may cause  $\text{\TeX}$  to interrupt with error

```
! TeX capacity exceeded, sorry [main memory size=3000000].
```

when using `latex` in `dvips` mode. While writing the DVI file, media files in the current page that are about to be embedded are kept in  $\text{\TeX}$ ’s memory until shipping out of the readily typeset page. In the case of large or many files, this may be more than  $\text{\TeX}$  can cope with by default.

There are two options to handle such situations:

The first one is to increase  $\text{\TeX}$ ’s main memory. You may follow the steps in the Bugs section of the ‘animate’ package documentation. In  $\text{\TeX}$ Live-2012, the maximum value that can be set is `main_memory = 12435455`.

If increasing  $\text{\TeX}$ ’s main memory does not help, use the package option ‘`bigfiles`’ with `media9`. It defers file embedding from the DVI producing to the PS producing step.

## 9 Acknowledgements

This package was written using the new L<sup>A</sup>T<sub>E</sub>X3 syntax which was a lot of fun. Many thanks to the L<sup>A</sup>T<sub>E</sub>X3 team!

## References

- [1] Adobe Systems Inc.: *Strobe Media Playback*, 2010, available at [http://osmf.org/strobe\\_mediaplayback.html](http://osmf.org/strobe_mediaplayback.html)
- [2] Adobe Systems Inc.: *Adobe Supplement to ISO 32000, BaseVersion 1.7, ExtensionLevel 3*, 2008, available at [http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/adobe\\_supplement\\_iso32000.pdf](http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/adobe_supplement_iso32000.pdf)
- [3] Adobe Systems Inc.: *JavaScript for Acrobat 3D Annotations API Reference*, available at [http://livedocs.adobe.com/acrobat\\_sdk/10/Acrobat10\\_HTMLHelp/JS\\_3D\\_Intro.90.1.html](http://livedocs.adobe.com/acrobat_sdk/10/Acrobat10_HTMLHelp/JS_3D_Intro.90.1.html)
- [4] Adobe Systems Inc.: *JavaScript for Acrobat API Reference*, available at [http://livedocs.adobe.com/acrobat\\_sdk/10/Acrobat10\\_HTMLHelp/JS\\_API\\_AcroJSPreface.87.1.html](http://livedocs.adobe.com/acrobat_sdk/10/Acrobat10_HTMLHelp/JS_API_AcroJSPreface.87.1.html)
- [5] The Apache Software Foundation: *Apache Flex SDK*, available at <http://flex.apache.org>
- [6] ECMA International: *Universal 3D File Format (ECMA-363), 4th Edition*, 2007, available at <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-363%204th%20Edition.pdf>
- [7] Adobe Systems Inc.: *PRC Format Specification*, available at [http://livedocs.adobe.com/acrobat\\_sdk/10/Acrobat10\\_HTMLHelp/API\\_References/PRCReference/PRC\\_Format\\_Specification/index.html](http://livedocs.adobe.com/acrobat_sdk/10/Acrobat10_HTMLHelp/API_References/PRCReference/PRC_Format_Specification/index.html)
- [8] A. Hammerlindl, J. Bowman and T. Prince: *Asymptote: The Vector Graphics Language*, available at <http://asymptote.sourceforge.net>
- [9] A. A. Balakin: *MathGL - library for scientific graphics*, available at <http://mathgl.sourceforge.net>
- [10] T. O'Rourke, T. Strelchun: *Universal 3D Sample Software*, available at <http://sourceforge.net/projects/u3d>
- [11] P. Cignoni *et al.*: *MeshLab*, available at <http://meshlab.sourceforge.net>
- [12] RightHemisphere Inc.: *DeepExploration*, <http://www.righthemisphere.com/products/dexp/>