Visual Technology Services Ltd.

Conversion Automation Toolkit - Technical documentation and User Manual

Visual Technology Services
1/25/2019

Table of Contents

1. What is pdf3d.io?

pdf3d.io is a suite of programs, dynamic libraries, RESTful server and examples that facilitates conversion of 3D models into 3D PDF file format. The key characteristics are file-based conversion and a rich XML-based API. Apart from supporting a large number of 3D file formats it provides vast number of customizable parameters and configurations including changing scene lighting conditions, metadata and mesh quality to direct PDF document modification. Automated batch production processing is yet another powerful feature for converting large number of 3D models. **pdf3d.io** can be easily deployed through automation scripting or to an online web server environment. It is not a just a single application, and does not include a user-interface layer, consequently is intended for developers and system integrators. **pdf3d.io** is "self-hosted", meaning it installs and runs locally on client systems. The conversion engine can be configured on a "server", but also bundled directly into local client applications if required.

2. Who is it for?

pdf3d.io is a software development toolkit, designed to be used by programmers with skills in scripting, web backend stacks, XML manipulation and use of command-line and dynamic library services. Integration into existing software can be much faster than traditional compiled program development. It is used where an unattended batch script is used to convert a number of 3D files into PDF using a standard template. It is used for on-demand PDF generation as a service behind a website. It is also used for desktop applications where it can be called to convert a temporary intermediate format to 3D PDF without any involvement of a remote service. Unlike other applications such as PDF3D ReportGen, it does not provide an end-user graphical interface layer.

Currently **pdf3d.io** customers include large multinational industrial manufacturing companies, remote data collection feed report production, as well as small internal-use or equipment-coupled application software.

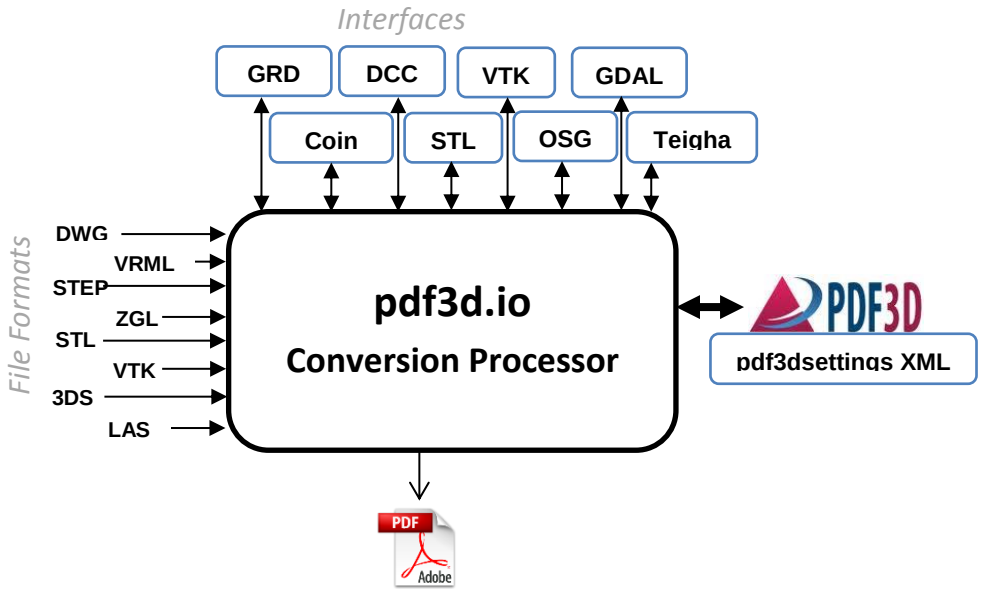
3. What does it do?

pdf3d.io can:

- Automatically and intelligently process 3D model data, converting it into 3D PDF reports, applying a preset style or layout.
- Hook into existing systems, including ERP and PLM, through C++, C# .NET, Visual Basic, Python and PHP
- Be controlled by multiple programming and scripting languages, and is fully compatible with Windows, Linux and macOS.
- Utilize and customise the PDF Templates included in the library
- Enhance your content further with reporting features which include Comments, Measurements, Annotations and External Links
- Enable key security features within your 3D PDFs including password protection, watermarking and strong 256-bit encryption
- Supports over 70 different industry files and formats
- Import settings created by the PDF3D ReportGen desktop application

4. Product overview

pdf3d.io is ideal for server-side deployment, batch automation, and integration into larger work-flow systems. The following figure shows a bird's eye view of how **pdf3d.io** works. It inputs a wide variety of 3D file formats which are forwarded to appropriate interfaces for identification and parsing. After the interface has processed the 3D data and stored it into the memory Pdf3d.io invokes processing services to perform operations on that data and finally export it to PDF file format.



5. Three-Way Access Points

- **Command-Line for simple no-coding operations**
- **Dynamic Library for tighter integration using in-memory XML**
- **RESTful API using a remote server**

pdf3d.io comes in three variations: a command line application program, a dynamic link library, and an always-live RESTful server. The command line version can be run through a standard console within a supported operating system without programming. The dynamic link library provides direct access to **pdf3d.io** API functions and can be interfaced with many common languages such as C++, C#, VB, PHP, Python, Ruby, Java and Delphi. In this document however, we will primarily be concentrating on the command line version of the **pdf3d.io**. The pdf3d.io.rest server provides http communication to upload models, convert and download PDFs from a remote or self-hosted local server.

6. Quick Start – show me the code!

If you are in a rush, here is the smallest possible way to get a conversion from STL to 3D PDF, replace input.stl with your own file, specify output PDF filename, with everything else left to defaults:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<pdf3d:InputParameters xmlns:pdf3d="http://www.pdf3d.com/ns/2012/03/pdf3d.io"
                       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                       xsi:schemaLocation="http://www.pdf3d.com/ns/2012/03/pdf3d.io pdf3d.io.xsd"
                       xsi:noNamespaceSchemaLocation="pdf3d.io.xsd">
    <Assembly>
        <InputFileName value="input.stl"/>
    </Assembly>
    <OutputFileName value="output.pdf"/>
</pdf3d:InputParameters>
```

Save this to *my.pdf3dsettings*, open a command window and type:

pdf3d.io.exe my.pdf3dsettings

See "Minimal Input File" below for description of these basic parts.

7. pdf3d.io Simple Command-Line Input Syntax

The command line program version of the Pdf3d.io takes a maximum of 2 parameters as input:

`pdf3d.io.exe <input file> [configuration file]`

<input file> is mandatory. It points to the pdf3dsettings XML file containing settings for model conversion, animation, data table generation, multimedia objects insertion, drawing commands and document security etc.

[configuration file] is optional. It specifies which of the various conversion interfaces to enable. If no argument is given, *"pdf3d.io-Win.xml"*, *"pdf3d.io-Linux.xml"* or *"pdf3d.io-Apple.xml"* is assumed (depending on the operating system) to be in the current directory.

a. Input File

The input file contains configuration parameters in the form of XML tags. The file begins with the standard xml tag:

```
<?xml version="1.0" encoding="utf-8"?>
```

The <pdf3D:InputParameters> tag is listed immediately after <?xml> tag. It is the root tag for all sub-elements which optionally specifies link to the *PDF3DXmlServer.xsd* file. The *PDF3DXmlServer.xsd* file is a huge collection of schema tags which define what elements and attributes may appear and how they might appear in the input xml file. Essentially it checks the syntax of the input.xml file. All other configuration tags are enclosed within <pdf3D:InputParameters> block. Following code listing shows the barebones structure of input.xml:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<pdf3d:InputParameters xmlns:pdf3d="http://www.pdf3d.com/ns/2012/03/pdf3d.io"
                       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                       xsi:schemaLocation="http://www.pdf3d.com/ns/2012/03/pdf3d.io pdf3d.io.xsd"
                       xsi:noNamespaceSchemaLocation="pdf3d.io.xsd">
...
```

Please note that specifying the pdf3d.io.xsd is optional. We can omit it and hence simplifying <InputParameters> tag as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<InputParameters>
    ...
    <!-- Various configuration tags -->
    <PDF Document manipulation tags>
    <3D Model import and configuration tags>
    <Animation related tags>
    ...
</InputParameters>
```

However, it is highly encouraged to specify the xsd schema file. See section 6 below for step by step explanation on how to create the input file.

b. Interface Configuration File

Specifying configuration file as an input to Pdf3d.io is optional. If no configuration file is specified, one of the three defaults is automatically chosen depending on the operating system: "`pdf3d.io-Win.xml`", "`pdf3d.io-Linux.xml`" and "`pdf3d.io-Apple.xml`". Configuration file contains a list of interfaces (or plugins in the new version) which Pdf3d.io uses to identify and read input 3D file format and load it into the memory. Once model is read **pdf3d.io** uses PDF3D the conversion engine to export the model into PDF format.

General structure of the configuration file is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<InterfacesConfiguration>
    ...
    <Interface filename=" ... "/>
    ...
</InterfacesConfiguration>
```

Interfaces to be included are enclosed within `<InterfacesConfiguration>` tags. The `<Interface>` tag results in loading the specified interface used by **pdf3d.io**. Depending on the operating system interface plugin file names might differ. For example in order to load the coin interface on windows, following name will be used:

```
<Interface filename="PDF3DCoinPlugin.dll"/>
```

On a Linux system however, coin interface will be loaded as follows:

```
<Interface filename="libPDF3DCoinPlugin.so"/>
```

On Mac OS:

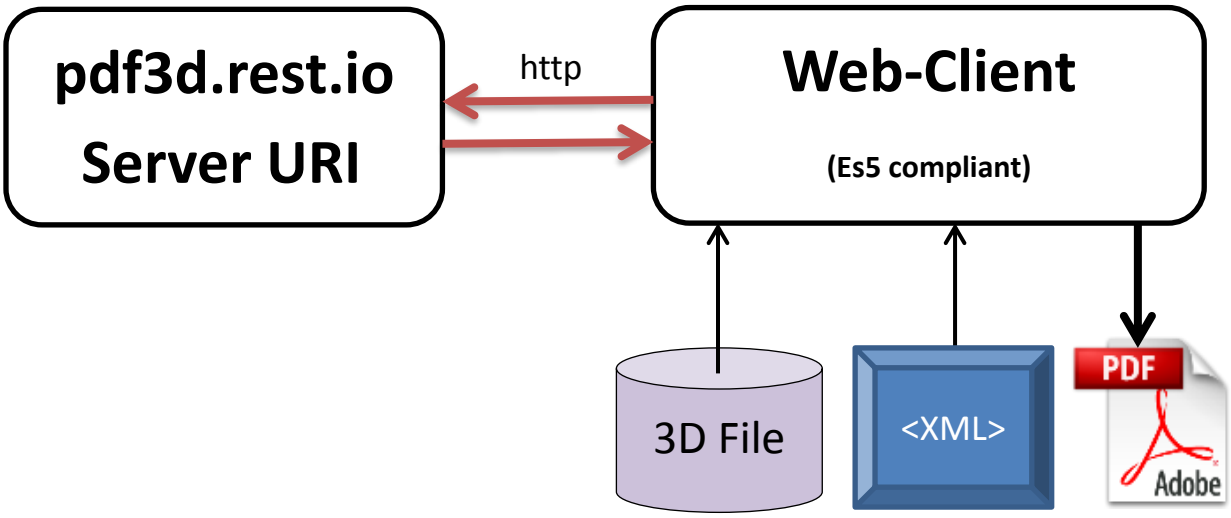
```
<Interface filename="libPDF3DCoinPlugin.dylib"/>
```

Following table lists the interfaces and their operating system dependent library file names (Note the actual Plugin list is much larger, this is only for illustration.):

Plugin	Windows	Mac OSX	Linux
Coin	PDF3DCoinPlugin.dll	libPDF3DCoinPlugin.dylib	libPDF3DCoinPlugin.so
GRD	PDF3DGrdPlugin.dll	libPDF3DGrdPlugin.dylib	libPDF3DGrdPlugin.so
STL	PDF3DStlPlugin.dll	libPDF3DStlPlugin.dylib	libPDF3DStlPlugin.so
VTK	PDF3DVtkPlugin.dll	libPDF3DVtkPlugin.dylib	libPDF3DVtkPlugin.so
OSG	PDF3DOsgPlugin.dll	libPDF3DOsgPlugin.dylib	libPDF3DOsgPlugin.so
...	...	...	...

8. pdf3d.io.rest RESTful server for backend or remote deployment

The **pdf3d.io** can be installed on your server making it possible to upload and convert 3D model data. **pdf3d.io** online server is a light weight package that can be easily configured onto a standard webserver. Once operational it provides the user a dynamic webpage. The user can then upload his desired 3D data, set configuration parameters using detailed interface provided by the webpage and finally download the output PDF file.



The process is initiated by **pdf3d.io.rest.exe**, the RESTful server providing a JSON+XML service over http, without the need of a webserver such as Apache, IIS, Tomcat, etc. The server may run continuously, providing conversions on demand. A sample html client application is provided at examples/html/ which can be loaded directly into a web browser (Chrome, Edge, Firefox, Safari). Through this interface, 3D model files can be uploaded, conversion settings specified and the conversion initiated. The HTML client (HTML documents + embedded JavaScripts) uses typical HTTP methods for communication with **pdf3d.io.rest.exe.**

The small resource file **pdf3d.io.rest.xml** specifies the server configuration details, including the server communication socket (typically 80 or 443), within the *<ServerConfiguration>* tag.

a. REST Server Functions Offered

The **pdf3d.io.rest** server provides an always-on service to convert 3D models to PDF. Because it runs in a separate process, or even a separate machine, there is strong isolation from the client. This eliminates direct library loading, or calling out to local executable programs. The actual 3D PDF conversion services are identical to using local resources, adding a communication and file transfer layer.

b. Typical REST Session Outline

1. Establish a connection using server URI, receive a valid session token
2. Upload input files to server, using create assembly file request
3. Request Conversion action by passing XML conversion settings to server and trigger process
4. On completion, receive completed PDF from server, write to local file

c. How to Access pdf3d.io.rest Server

The **pdf3d.io.rest** server is accessed by using http POST and GET requests, directly by a web browser or another integrated backend web application.

d. REST Server Protocols for communication

The server uses normal common practice TCP/IP internet socket communications, with default port 80 for http requests or port 443 for https URI/URL requests, with arguments passed through the URI to the server.

e. How conversion settings are specified to server

The request for conversion is invoked using a POST request containing an XML specification. This specifies the assembly input 3D model files, output file and an extensive set of optional tags for adjusting the 3D PDF content and 2D PDF page layout. A separate reference guide documents each tag, and the pdf3d.io.xsd schema provides a formal reference.

f. REST Session Tokens

The token is an alpha-numerical key issued by the server when a connection is established. Subsequent communication sent to the server uses this key to identify a unique conversion session. The Tokens are persistent and allow for follow-up data management and caching of previous files.

g. Security Considerations

Firewall may block operations by default, http port (80), https port (443) and pdf3d.io.rest.exe may need security exception or white-list allowance settings. Uploaded files are not encrypted, all have the same owner-group permissions, and may be accessible to any account with direct server access.

h. How to change REST Server Port Number?

The small configuration file pdf3d.io.rest.xml has setup configuration values including port number.

i. How to add SSL (secure https)?

The small configuration file pdf3d.io.rest.xml has setup configuration values including protocol type. It could be HTTP or HTTPS. By default, pdf3d.io.rest uses HTTP, so, you need to change this value to HTTPS. We recommend that administrators replace the default SSL certificate (.pem files) with your own. To enable secure communications, change the configuration file specification from <ServerProtocol>HTTP</ServerProtocol> to <ServerProtocol>HTTPS</ServerProtocol>.

j. How to check for server errors?

HTTP responses contain some error codes, as described in formal API. Also, you can get pdf3d.io conversion log information back from server. It is possible to make dynamic progress requests (polling) messages during a conversion if required.

k. Multiple Simultaneous Connections or Conversions and Queuing?

The configuration file has a maximum count of threads parameter. So, use one thread per one user. Simultaneous conversion limits are also constrained and set by the license manager.

l. Can uploaded files be used again during session?

Yes. The uploaded files are stored on server. For the second and next conversions you can skip the uploading step, and request conversion by passing XML conversion settings containing references to the existing input files. Moreover, resulting 3D PDF files could be also reused. If XML conversion parameters are not changed since the last conversion of the same output 3D PDF file then pdf3d.io.rest returns the previously created 3D PDF file without actual conversion, implementing caching of resulting 3D PDF files.

m. Are uploaded files removed from server after session?

No. They are stored in clients/{token_id} folder where {token_id} is the actual client token id. So, if the next session uses the same token id as before, this new session could access the existing input files. However, each new session has a unique token id, making impossible to reuse existing uploaded input files between different sessions.

n. What processes require license check-out?

Only the actual pdf3d.io.exe process requires license check-out. The pdf3d.io.rest.exe itself doesn't require any license check-out, pdf3d.io.exe is executed only at the conversion step. So uploading input files and establishing a new session can be done without any valid license. A log file records conversion activity for audit and license.

o. How to check if previously uploaded file is available?

You can send HTTP GET request the status with file name, like http://server_name/check_files/input_file_name?token={token_id} and server will respond returning True or False.

p. How to delete 3D model or PDF from server?

Use the HTTP request to remove an individual file or purge all files identified with a token.

q. Can other users with a different Token access my files?

No. Other users with different token cannot access files managed under a different token. However, if another user steals a token then files could be exposed. Only the system administrator can normally access files directly on the server. The files are unencrypted on the server.

r. How to get a list of supported file format types?

Use the HTTP request to receive a list of supported formats.

s. How to check if conversion license is available, trial-time left?

Use the HTTP request to receive license status. If no license is available the conversion requests will fail.

t. What is the maximum upload file size, and how is it configured?

Server doesn't limit size of the input files. However, it could be some reasonable practical limit, introduced by available free space on server, and waiting time for uploading. HTML JavaScript example uses 10MB limit, but this limit could be changed.

u. How to configure the Html client example?

The sample html client can be built using **Node.js.** This process uses **webpack** is a *static module bundler* for modern JavaScript applications. When webpack processes the application, it internally builds a dependency graph which maps every module your project needs and generates one or more *bundles*. The “production” folder contains the output, ready to load into a web browser. This system supports all browsers that are ES5-compliant (IE8 and below are not supported).

9. PDF3D.IO Rest API – Functional Introduction

PDF3D.IO Rest server is a HTTP or HTTPS (depending on parameters in the configuration file) Server which allows creating 3D PDF documents from various input file formats. Usually PDF3D.IO Rest server uses JSON format in HTTP responses, however, in some particular cases it uses other data formats.

1. Initiating connection

Before communication with PDF3D.IO Rest server you need to obtain a valid client token. The token is used for client identification. So, you need to send a POST request to */clients* URI, for instance:

```
POST http(s)://localhost/v1.0/clients
```

We assume that PDF3D.IO Rest server is running on 80(HTTP) or 443(HTTPS) port on localhost. The returned response will be in JSON format and will be something like:

```
{ "Token": "8a9a75cb-5c82-4b28-9843-c003111d19f9" }
```

You need to use this token for all other subsequent PDF3D.IO Rest server invocations, by specifying the *token* HTTP(s) request parameter. You can check your token for validity by sending GET request to */v1.0/clients/${token}* URI. Where ${token} is a valid client token. For instance:

```
GET http(s)://localhost/v1.0/clients/8a9a75cb-5c82-4b28-9843-c003111d19f9
```

Again the response will be in JSON format and it will be a true or false value:

```
{ "Result" : "true" }
```

in the case the token is valid, otherwise returning false is the token is not valid. Also you can close the connection with the server by sending a DELETE request to */v1.0/clients/${token}* URI. After this all use of this token value will fail. For instance:

```
DELETE http(s)://localhost/v1.0/clients/8a9a75cb-5c82-4b28-9843-c003111d19f9
```

Also the response will be in JSON format and it will be a true or false value:

```
{ "Result" : "true" }
```

2. Uploading files

Before conversion you will need to upload the all input files to PDF3D.IO Rest server. Each input file we call *assembly*. To upload some input file to PDF3D.IO Rest server use POST request to */v1.0/assemblies/${file_name}?token=${token}* URI. Where ${file_name} is name of the input file and ${token} is valid client token. The request body should contain the actual file content which could be binary (for binary input formats), text (for text input formats), XML (for XML input formats), etc. Also don't forget to specify the *token* parameter. For instance:

```
POST http(s)://localhost/v1.0/assemblies/tetrahedron.wrl?token=8a9a75cb-5c82-
4b28-9843-c003111d19f9
```

The request body should contain the actual file content, so, for *tetrahedron.wrl* it will be the following:

```
#VRML V2.0 utf8
#
#  tetrahedron.wrl  10 August 1999
#
#  Rikk Carey and Gavin Bell,
#  The Annotated VRML 2.0 Reference Manual,
#  pages 198-199
#

Viewpoint { description "Initial view" position 0 0 9 }

NavigationInfo { type "EXAMINE" }

#
#  A tetrahedron, with 4 vertices and 4 faces and a color at each vertex.
#
Transform {
  translation 1.5 -1.5 0.0
  children Shape {
    appearance DEF A Appearance { material Material { } }
    geometry IndexedFaceSet {
      coord Coordinate {
        point [
           1.0  1.0  1.0,
           1.0 -1.0 -1.0,
          -1.0  1.0 -1.0,
          -1.0 -1.0  1.0,
        ]
      }
      coordIndex [
        3, 2, 1, -1,
        2, 3, 0, -1,
        1, 0, 3, -1,
        0, 1, 2, -1,
```

```
        ]
        color Color {
          color [
            0.0 1.0 0.0,
            1.0 1.0 1.0,
            0.0 0.0 1.0,
            1.0 0.0 0.0
          ]
        }
        colorPerVertex TRUE
      }
    }
  }
```

In case of success PDF3D.IO Rest server will respond in JSON format:

```
{ "Result" : "true" }
```

and returning false if some error has occurred.

Also you can get information about input files (hash and size of file). To getting information about input file you need to send GET request to */v1.0/assemblies/${file_name}?token=${token}* URI. For instance:

```
GET http(s)://localhost/v1.0/assemblies/tetrahedron.wrl?token=8a9a75cb-5c82-
4b28-9843-c003111d19f9
```

The request body should contain actual information about file, so for *tetrahedron.wrl* it will be the following:

```
    {
      "Result" : "true",
      "Hash" :
"6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b",
      "Size" : "10435"
    }
```

If the input file does not exist on the server, then PDF3D.IO Rest server will respond by JSON format:

```
{ "Result" : "false" }
```

3. Conversion and parameters

After uploading all the input files you can perform a conversion. To perform a conversion please send HTTP POST request to */v1.0/files?token=${token}* URI. Where ${token} is the valid client token. The request body should contain conversion

parameters in XML format. XML conversion parameters should satisfy XSD schema from *http(s)://www.pdf3d.com/ns/2012/03/pdf3d.io* namespace. For instance:

```
 POST http(s)://localhost/v1.0/files?token=8a9a75cb-5c82-4b28-9843-
c003111d19f9
```

with the following request body:

```
    <?xml version="1.0" encoding="iso-8859-1"?>

    <pdf3d:InputParameters
xmlns:pdf3d="http://www.pdf3d.com/ns/2012/03/pdf3d.io"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.pdf3d.com/ns/2012/03/pdf3d.io pdf3d.io.xsd"
xsi:noNamespaceSchemaLocation="pdf3d.io.xsd">

    <OutputFileName value="sample.pdf"/>
    <U3DOutputFormat value="false"/>
    <U3DQualityFactor value="1"/>
    <ZoomFactor value="0.8"/>
    <ModelUnitType value="ModelUnit_is_cm"/>

    <Assembly>
        <InputFileName value="tetrahedron.wrl"/>
        <NodeName value="Coin Interface imported scene - sample"/>
        <AssemblyProperties>
            <CcwMode       value="CCW 0"/>
            <SolidMode     value="SOLID 1"/>
        </AssemblyProperties>
    </Assembly>
    </pdf3d:InputParameters>
```

Previously uploaded input files should be referenced by *value* attribute of *InputFileName* tag. Incorrect input file names will cause failures during conversion. In case of success the PDF3D.IO Rest server will respond by returning the actual generated PDF file content:

```
    %PDF-1.7
    %в??У
    1 0 obj
    ...
```

In case of any failure PDF3D.IO Rest server will respond by JSON format:

```
{ "Result" : "false" }
```

You can also get the generated PDF file later by sending GET request to */v1.0/files/${output_file_name}?token=${token}* URI. Where ${output_file_name} is a valid output file name which was used in conversion parameters as *value* attribute of *OutputFileName* tag and ${token} is valid client token. The response will be the same as for the corresponding POST request.

4. Obtaining Progress Logs During Conversion

If conversion of file last for long time, you can receive logs during conversion. To getting logs during conversion please send GET request during conversion to: */v1.0/logs/${token}* URI. For instance:

```
GET http(s)://v1.0/logs/8a9a75cb-5c82-4b28-9843-c003111d19f9
```

In case of success PDF3D.IO Rest server will respond by actual logs of conversion, for example:

```
Job Started:
Job Progress: 50% completed. Entered Group
Job Progress: 50% completed. Entered Geode face
Job Progress: 50% completed. Entered Geometry
Job Progress: 50% completed. Adding Triangles
Job Progress: 50% completed. Left Geode
...
```

5. Working with the server filesystem

Beside conversion, you can easily work with files associated with the current token. For example, you can read and delete files in your client folder. For deteting some file, you need to send DELETE request to: */v1.0/files/${file_name}?token=${token}* URI. For instance:

```
DELETE http(s)://localhost/v1.0/files/tetrahedron.wrl?token=8a9a75cb-5c82-
4b28-9843-c003111d19f9
```

The response will be in JSON format and it will be a true or false value:

```
{ "Result" : "true" }
```

Also, you can purge all your files in your directory. For it, you need DELETE request to: */v1.0/files/?token=${token}* URI. For instance:

```
DELETE http(s)://localhost/v1.0/files?token=8a9a75cb-5c82-4b28-9843-
c003111d19f9
```

The response will be in JSON format and it will be a true or false value:

```
{ "Result" : "true" }
```

But also you need read file from the server, you can send GET request to: */v1.0/files/${file_name}?token=${token}* URI. For instance:

```
GET http(s)://files/tetrahedron.wrl?token=8a9a75cb-5c82-4b28-9843-
c003111d19f9
```

The response will be in JSON format and it will be a true or false value:

```
{ "Result" : "true" }
```

10. PDF3D.IO Rest API Reference

The following table specifies the individual actions to send and receive information with the server.

URI	HTTP(s) Method	Request Parameters	Request Body	Description	Response	Example Response
/v1.0/clients	POST			Creates a new session	Created session token in JSON	`{ "Token": "8a9a75cb-5c82-4b28-9843-c003111d19f9" }`
/v1.0/clients/${token}	GET			Checks the passed token for validity	Boolean flag in JSON, true means valid token, false otherwise	`{ "Result" : "true" }`
/v1.0/clients/${token}	DELETE			Closes session with specified token	Boolean flag in JSON, true means success, false otherwise	`{ "Result" : "true" }`
/v1.0/online	GET			Checks if PDF3D.IO Server is available here	Boolean flag in JSON, true means available server, no response means absent server or offline	`{ "Result" : "true" }`
/v1.0/assemblies/ ${file_name}	POST	token= ${token}	Request body should contain the actual file content which could be binary (for binary input formats), text (for text input formats), XML (for XML input formats), etc.	Uploads the input file for conversion	Boolean flag in JSON, true means success, false otherwise	`{ "Result" : "true" }`

/v1.0/assemblies/ ${file_name}	GET	token= ${token}		Returns information about the previously uploaded input file	Result, Hash and Size parameters in JSON. For Result true means success, false otherwise. Hash contains SHA256 hash of the input file, Size contains size in bytes of the input file. Hash and Size parameters are missing in case of failure.	`{ "Result" : "true", "Hash" : "6b86b273ff34fc9d6b80…", "Size" : "10435" }`
/v1.0/plugins/ ${token}	GET			Returns information about available plugins for conversion different formats.	Names of availbale plugins, their supported formats and extensions for conversion.	`{ "plugins": [ { "name": "Coin Interface", "formats":[ { "name": "VRML","description": "VRML Uncompressed Format" } ], "extensions": [ { "name": ".vrml", "formatIndex": "0" } ] } ] }`
/v1.0/files	DELETE	token= ${token}		Purges client folder with all files associated with token.	Boolean flag in JSON, true means success, false otherwise	`{ "Result" : "true" }`
/v1.0/files/ ${file_name}	DELETE	token= ${token}		Removes selected file from the client folder.	Boolean flag in JSON, true means success, false otherwise	`{ "Result" : "true" }`

/v1.0/files/ ${file_name}	GET	token= ${token}		Returns file content from specified file.	Data from file AS-IS.	%PDF-1.7 %â??Ó 1 0 obj ...
/v1.0/files	POST	token= ${token}	Request body should contain the actual data with settings for conversion(file ine .pdf3dsettings format).	Converts file by settings from request body.	Data of conversion file in PDF fromat.	%PDF-1.7 %â??Ó 1 0 obj ...
/v1.0/logs/ ${token}	GET			Returns logs during conversion of file.	Logs during conversion process.	Dump(): Start dump [TInputParameters] Dump(): OutputFileName:OBJECT = Dump(): Start dump [TStringParameter]

11. Converting 3D models to PDF with XML Instructions

In this section we will go through the detailed process of generating input xml file manually or programmatically.

a. Minimal Input File

The following is the minimum possible input file which converts a 3D model into 3D PDF format:

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <pdf3d:InputParameters xmlns:pdf3d="http://www.pdf3d.com/ns/2012/03/pdf3d.io"
3                         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4                         xsi:schemaLocation="http://www.pdf3d.com/ns/2012/03/pdf3d.io pdf3d.io.xsd"
5                         xsi:noNamespaceSchemaLocation="pdf3d.io.xsd">
6
7      <Assembly>
8          <InputFileName value="input.stl"/>
9      </Assembly>
10
11 </pdf3d:InputParameters>
```

Line 1: Contains the mandatory `<xml>` tag which must be placed at the beginning of every xml document. All conversion parameters are children of the root node "`InputParameters`". Lines 3-7 lists beginning of `<pdf3d:InputParameters>` tag which specifies the location of XML schema file. As mentioned above specifying a xsd file is optional. Line 13 is the closing tag for `<pdf3d:InputParameters>`. All other tags, parameters and configurations must reside between these tags.

Lines 9-11: Specify the `<Assembly>` tag. Every 3D model related setting and configuration must be enclosed within these tags. A single input file can contain multiple `<Assembly>` tags pointing to multiple 3D models which will be merged in a single output PDF file by the Pdf3d.io. `<InputFileName>` occurs on line 10 between the `<Assembly>` tags. It specifies filename and complete path to the 3D model. Specifying only "`input.stl`" will tell the **pdf3d.io** to read the model from the current directory. Assuming we are using Windows, we can replace

the above file name with "`c:\models\input.stl`" in the above example (assuming that the path is valid). For other OS appropriate path syntax has to be observed.

If we run the **pdf3d.io** command-line program with the above file as input:

```
C:\..\>pdf3.io.exe input.xml
```

pdf3d.io generates output PDF file named "`Output.pdf`" in the current directory. Since no configuration file is specified `pdf3d.io-Win.xml` is assumed by default.

b. 3D Model Configuration

There are many parameters with which we can tweak the 3D model in the final output. These include setting the opacity of the scene, diffuse color, position, orientation, visibility and many more. For a complete list of parameters please refer to the Input XML Specifications documentation. In this section we are going to look at the most important tags that directly affect the 3D scene.

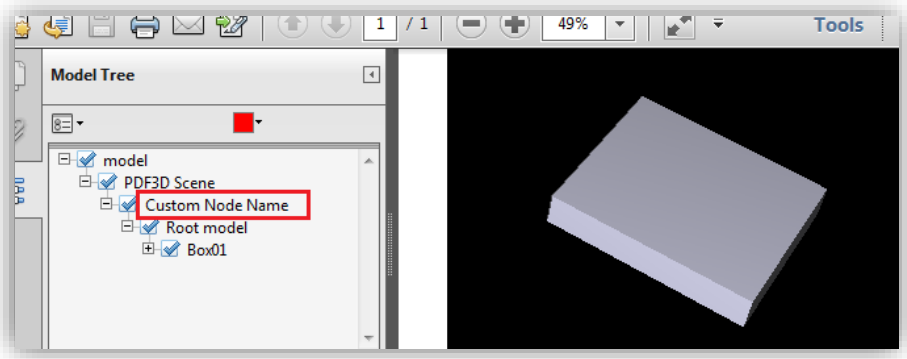
i. Setting Input File Name

Consider the following input.xml file (Note that in order to save space extra details have been omitted from the `<pdf3d:InputParameters>` tag (line 2)):

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <pdf3d:InputParameters ... >
03
04     <OutputFileName value="xml-sample.pdf"/>
05     <Assembly>
06          <NodeName value="Custom Node Name"/>
07          <InputFileName value="box.3ds"/>
08     </Assembly>
09
10 </pdf3d:InputParameters
```

Line 4: Specifies the output filename. File name is given as a string value to the `<OutputFileName>` tag. If this tag is not specified then default value of "output.pdf" is assumed.

Line 6: Node name of the parent node of that 3D model. This tag occurs inside the `<Assembly>` as it directly affects a particular mesh. If this tag is not present the system assumes a default node name. Node names can be viewed by toggling the "Model Tree" button in Adobe Reader. Following figure shows the result of the above code:

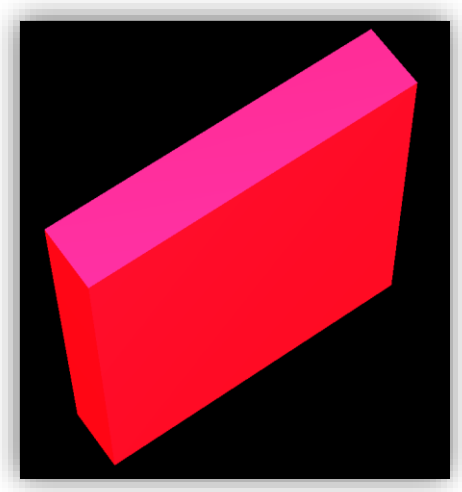


ii. Setting Lighting Conditions

Specific settings that manipulate the model directly are given inside the <AssemblyProperties> block, which in turn resides in the <Assembly> block as shown in the following code listing:

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <pdf3d:InputParameters ...>
03    <OutputFileName value="xml-output.pdf"/>
04      <Assembly>
05         <NodeName value="Custom Node Name"/>
06         <InputFileName value="box.3ds"/>
07         <AssemblyProperties>
08             <DiffuseColor  red="255" green="0"  blue="0"  alpha="255"/>
09             <AmbientColor  red="255" green="0"  blue="0"  alpha="255"/>
10             <SpecularColor red="50"  green="50" blue="150" alpha="255"/>
11
12         </AssemblyProperties>
13      </Assembly>
14 </pdf3d:InputParameters>
```

Lines 8,9,10: Lighting conditions and material properties of the model specified inside the <AssemblyProperties> block. For example the diffuse color is specified by four parameters red, green, blue and alpha: <DiffuseColor red="255" green="0" blue="0" alpha="255"/>. **pdf3d.io** produces following output after processing above code listing:



iii. Changing Scale and Opacity

Similarly, we can also set the opacity and scale of the model with <Opacity> and <Scale> tags respectively as shown in the following code listing:

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <pdf3d:InputParameters ... >
03  <OutputFileName value="xml-output.pdf"/>
04     <Assembly>
05         <InputFileName value="box.3ds"/>
06         <AssemblyProperties>
07            <Opacity value="0.4"/>
08            <Scale autoScale="false" generalScale="1.0">
09                <ScalePerComponent x="1.0" y="0.2" z="0.1"/>
10            </Scale>
11         </AssemblyProperties>
12     </Assembly>
13 </pdf3d:InputParameters>
```

Line 7: `<Opacity>` tag controls the opacity of the model. Valid values lie in the range 0.0 to 1.0. 0.0 means model is completely transparent and 1.0 means the model is completely opaque. Figure below shows the result of opacity value 0.4.

Line 8: `<Scale>` controls the amount of scaling in the scene.

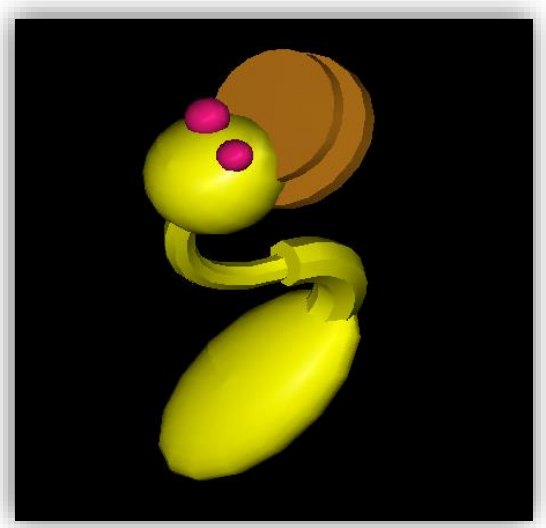
Line 9: `<ScalePerComponent>` tag can be enclosed in the `<Scale>` block. It controls scaling of the model on individual axis. For example, in the above code listing the model is scaled on the y and z axis while it remains unaffected on the x axis essentially elongating the geometry as shown below:

iv. Changing Orientation

Orientation of the model can be changed by specifying the angle and axis along which to orient in the `<Orientation>` tag as shown below:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <pdf3d:InputParameters... >
3     <Assembly>
4         <InputFileName value="duck.iv"/>
5         <AssemblyProperties>
6             <Orientation angle="80" x="1" y="0" z="1"/>
7         </AssemblyProperties>
8     </Assembly>
9 </pdf3d:InputParameters>
```

Line 6: <Orientation> tag takes four parameters. The angle specifies how much to orient and x,y,z specify which axis to orient against. In this case the model is rotated at 80 degrees angle against x and z axis. Following figure shows the result of above code:



v. Model Visibility

Following tag controls the visibility of the model. If it is set to false the model will be completely invisible:

```
...
<AssemblyProperties>
    <Visible value="false"/>
</AssemblyProperties>
...
```

vi. Overriding Interfaces

We can tell the **pdf3d.io** to use a specific interface instead of automatically picking one and it will try to use our specified interface whenever possible. Following code shows how to override interfaces:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <pdf3d:InputParameters ... >
3     <Assembly>
4         <InputFileName value="duck.iv"/>
5         <AssemblyProperties>
6             <PreferredInterfaceName value="Coin Interface"/>
7         </AssemblyProperties>
8     </Assembly>
9 </pdf3d:InputParameters>
```

Line 6: Preferred interface is specified via `<PreferredInterfaceName>` tag. Possible values are “Coin Interface”, “Grd Interface”, “Stl Interface”, “Vtk Interface” and “Osg Interface”. Refer to section 5b for further information.

c. Global Configurations

This section details the most important tags and configurations which affect scene globally. For a complete list of parameters please refer to Input XML Specifications documentation. Consider following code snippet:

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <pdf3d:InputParameters ... >
03  <PDFECompliance value="false" />
04  <U3DOutputFormat value="false" />
05  <CompressionFormat value="U3D" />
06  <ZoomFactor value="0.8" />
07  <PDF3DSceneOptions lightingScheme="CAD" renderMode="Solid" />
08  <UseBackgroundColor>
09      <Color alpha="255" blue="255" green="255" red="190" />
10  </UseBackgroundColor>
11  <U3DQualityFactor value="1.0" />
12  <Assembly>
13      <InputFileName value="iflamigm.3ds"/>
14  </Assembly>
15  </pdf3d:InputParameters
```

Line 3: Specifies whether PDF/E compliance is to be enabled.

Line 4: If this value is set to be true then the Pdf3d.io produces U3D file as output instead of a PDF file. Default value is set to be false.

Line 5: Specifies internal compression format to be applied to the input models while exporting to PDF. Acceptable values include “U3D”, “U3D_RHC”, “PRC” and “PRC_HCT”. However “U3D_RHC” is assumed to be default value.

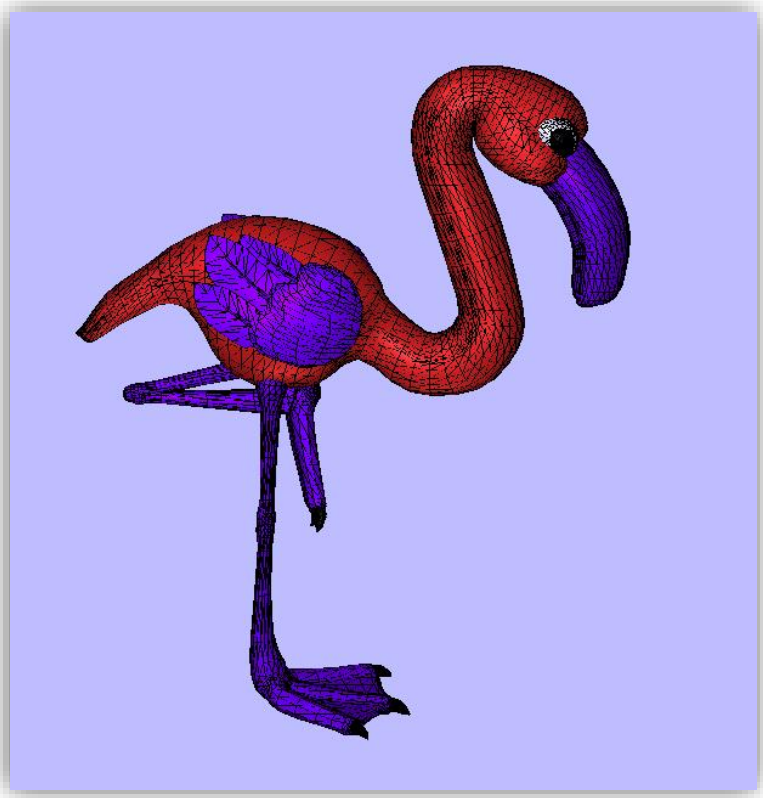
Line 6: Specifies the relative size of the scene relative to the viewport. For example, a zoom factor of 0.2 will result in a very small sized model being displayed on the screen as compared to the PDF viewport. It can be further zoomed in or out via mouse scrolling.

Line 7: Specifies overall scene lighting scheme and model rendering mode. For further details see the “Input Specifications” document or xsd.

Line 8-10: Background color of the scene. Actual color is specified by `<Color>` tag (line 9) which in turn is enclosed between `<UseBackgroundColor>` tags. Background color is composed of the standard red, green, blue and alpha components.

Line 11: Specifies the quality of the output U3D model. Value ranges from 0.0 (lowest quality) to 1.0 (highest quality)

Following figure shows the final output of the above XML code snippet:



d. Logging

Often there is a need to view detailed output messages during a conversion process. You can enable logging by using the <Logging> tag. When enabled it outputs very detailed information on the screen or in a separate log file:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <pdf3d:InputParameters ... >
3     <Logging file="log.txt" toFile="true" toScreen="true" verbose="true" />
4     <Assembly>
5         <InputFileName value="box.3ds"/>
6     </Assembly>
7 </pdf3d:InputParameters>
```

Line 3: Turns on the message logging system. The "file" parameter points to physical file to which all the messages will be redirected. If a file with same name already exists it will be overwritten. "toScreen" parameter if set to true tells the **pdf3d.io** to dump all the message information onto the screen. "verbose" directs Pdf3d.io to enable detailed message logging.

12. Template Design Layouts

The optional tag

```
<TemplateFileName value="MyTemplate.pdf"/>
```

can be used to reference an existing PDF containing design, graphics, text elements as static background for the new 3D PDF. Almost any PDF generating software may be used. In the simplest mode, a 3D view can be added onto an existing page by specifying 4 margin parameters for the view window location.

The accompanying tag

```
<MergeMode value="Placeholder">
     <ReplaceMode value="Number"/>
     <AnnotationNumber value="1"/>
 </MergeMode>
```

Is used to reference a particular page (using “AtPage” mode) or may reference an existing placeholder definition already present (created by the PDF3D with OFFICE plugins or similar process) or in Replace mode to swap out an existing dummy placeholder 3D scene with an actual conversion.

13. Animating 3D models

pdf3d.io provides simple mechanisms for creating 3D animations and exporting them to PDF format. Currently the **pdf3d.io** supports two types of animation systems: “sequence animations” and “rigid body animations” as described in the sections below.

a. Sequence Animation

Sequence animation consists of putting together different meshes and switching between them after short intervals. Consider following example:

```
1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <pdf3d:InputParameters ...>
3     <AnimationFromFolder animationName="anim1" inputFileNameFilter="*.wrl"
                                                 inputFolderName="animation">
4         <FrameDelays>
5             <FrameDelay value="0.1" />
6         </FrameDelays>
7     </AnimationFromFolder>
8     <InsertAnimationControls ... />
9 </pdf3d:InputParameters>
```

Line 3: `<AnimationFromFolder>` contains sequence animation related tags. `inputFileNameFilter` tag causes the Pdf3d.io to read only specific types of files while filtering or ignoring others. In the above example only `*.wrl` files are considered for animation. `inputFoldername` points to the location of the folder containing 3D model files.

Line 4-6: `<FrameDelays>` specify delay between frames. `<FrameDelay>` represents actual delay between two frames. To add variable delays between frames just add multiple `<FrameDelay>` tags inside `<FrameDelays>.`

See the input specification document for detailed list of related tags.

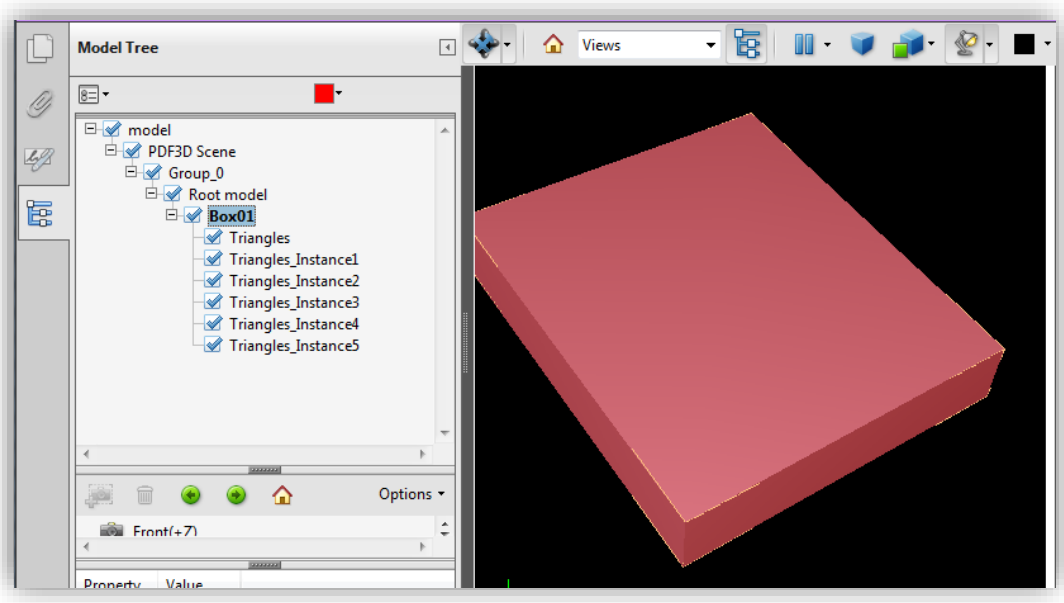
b. Rigid Body Animation

Before we can animate a 3D scene we must add animation controls in the document. Following code shows the minimal way to do it:

```
<InsertAnimationControls animationName="anim1"/>
```

Note that “animationName” is the only required parameter. All other parameters are optional. For detailed information about the animation control parameters see the XML Input specification documentation.

Every scene is composed of 3D models. Each of the 3D models is made up of a tree of nodes sorted in a hierarchical order. You can easily view nodes a model consists of and their names via the “Model Tree Control Panel” present in adobe reader as shown in the picture below:



Any sub node of a 3D model can be animated. Currently position, rotation and opacity animations are supported. You can easily attach as many key frames to a node as you like. **pdf3d.io** automatically interpolates animation between these key frames. A key frame represents the state of a model at a given time (more details below). Consider following code snippet:

```
01 <?xml version="1.0" encoding="iso-8859-1"?>
02 <pdf3d:InputParameters ... >
03     <Assembly>
04         <InputFileName value="box.3ds"/>
05     </Assembly>
06     <RigidAnimation animationName="Box-Animation" fps="20"
                                                hardTiming="false">
07         <NodeMovement localRepeatMode="Smooth" nodeName="Box01">
08             <KeyFrame>
09                 <Position y="-1.0" x="80.0" z="-20.0"/>
10                 <Delay value="3.0"/>
11             </KeyFrame>
12             <KeyFrame>
13                 <Position y="10.0" x="-80.0" z="20.0"/>
14                 <Delay value="1.0"/>
15             </KeyFrame>
16         </NodeMovement>
17     </RigidAnimation>
18     <InsertAnimationControls ... />
19 </pdf3d:InputParameters>
```

Line 6: All of the animation related tags are enclosed between `<RigidAnimation>`. Parameter "animationName" sets name of the current animation to the specified value for current and future reference. "fps" specify how many frames per second are to be run. "hardTiming" if set to true will force the Pdf3d.io to skip animation frames on a slow system.

Line 7: `<NodeMovement>` specifies the node to be animated. "nodeName" specifies the exact name of the node to be animated.

Line 8: Attaches a key frame to the animation. Pdf3d.io will automatically interpolate between two key.

Line 9: Specifies position of the node at current key frame. `<Position>` must be enclosed inside `<Keyframe>` tags. Similarly you specify current orientation via `<Rotation>` tag or even scale by using the `<Scale>` tag. Please refer to the input specification document for further details.

Line 10: Time interval between the current and next frame. Longer the delay, lengthier the animation between the key frames will be.

Lines 12-15: These lines specify another key frame representing a separate state of the node. Animation will be interpolated between this and previous key frames.

14. Manipulating PDF Document

a. Changing Document Appearance

i. Document creation modes

While exporting 3D scenes the PDF files can be merged, replaced, appended or insert the model at a specified page. Consider following input.xml file:

```
1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <pdf3d:InputParameters ... >
3    <MergeMode value="NewOrReplace" />
4    <Assembly>
5        <InputFileName value="box.3ds"/>
6    </Assembly>
7 </pdf3d:InputParameters>
```

Line 3: <MergeMode> tag is used to specify the PDF creation mode. In above code value "NewOrReplace" means that if target pdf file doesn't exist in the current directory then Pdf3d.io will create a new one. But if the pdf file having the same name already exists then it will be overwritten or replaced with the new one.

Other possible <MergeMode> values are "Prepend", "Append" and "AtPage". Prepend inserts the current 3D scene at the start of the existing pdf file. Append inserts the current 3D scene at the end of the existing pdf file. AtPage inserts 3D scene at a specified page. When AtPage is specified then <PageNumber> tag is used to indicate the page number at which to insert the scene as shown in the code below:

```
1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <pdf3d:InputParameters ... >
3   <MergeMode value="AtPage" />
4   <OnExistingPage value="true">
5   <PageNumber value="30" />
6     <Assembly>
7         <InputFileName value="box.3ds"/>
8     </Assembly>
9 </pdf3d:InputParameters>
```

Line 3: Specifies the AtPage value. This will direct the **pdf3d.io** to insert the 3D scene at a page given by the <PageNumber> tag (Line 5). If page number value is greater than the number of pages in the current pdf the **pdf3d.io** will insert appropriate number of pages in between.

Line 4: <OnExistingPage> tells the Pdf3d.io to inject the 3D view into the current page specified by <PageNumber> tag. Note that "MergeMode" must be selected to "AtPage" in order to use this tag.

ii. Document Page Settings

Consider the following code snippet:

```
01 <?xml version="1.0" encoding="iso-8859-1"?>
02 <pdf3d:InputParameters ... >
03  <PageSize height="800" width="500" />
04  <Margins adjustBottomMarginForControls="true" left="50.0"
                        right="50.0" bottom="150.0" top="50.0" />
05  <TitleString fontFamily="Arial" fontFile="" value="PDF3D SDK" />
06  <CaptionBox fontFamily="Times New Roman" fontFile=""
                                          value="Caption" />
07  <ShowToolBar value="true" />
08  <ShowUserInterface value="true" />
09  <Assembly>
10      <InputFileName value="box.3ds"/>
11  </Assembly>
12 </pdf3d:InputParameters>
```

Line 3: Page dimensions are specified via <PageSize>. The height and width parameters control the page size of the final output pdf document.

Line 4: <Margin> tells the Pdf3d.io how much white space to leave from each of the page edges before placing the viewport. This variable indirectly controls the size of the 3D viewport.

Line 5: `<TitleString>` specifies the main title of the document. “fontFamily” specifies the family of the font to render the title string, the famous “Arial” font for example. You can also use external font by setting the “fontFile” parameter. The “value” parameter is the actual title text to be displayed on the top of the document.

Line 6: `<CaptionBox>` adds a bordered read only text field at the bottom of the page. You can add text description about the 3D scene or any other explanation in this area.

Line 7: `<ShowToolBar>` can be used to enable or disable 3D scene viewing toolbar. You can navigate the scene or change its lighting conditions and rendering modes etc. When enabled this toolbar will hover over the viewport as soon as the mouse pointer enter scene boundaries.

Line 8: `<ShowUserInterface>` toggles the scene model tree panel on or off. By default it is hidden but you can override this parameter to “true” to make the panel visible at the start. Model tree panel can also be toggled from adobe reader interface directly by going to View→Show/Hide→Navigation Panes→Model Tree.

For further information and more advanced tags please see the pdf3d.io input specifications documentation.

b. Security

You can apply passwords, disable printing, make the document read-only and select encryption scheme with a single xml tag as shown below:

```
01 <?xml version="1.0" encoding="iso-8859-1"?>
02 <pdf3d:InputParameters ... >
03     <EnableDocumentSecurity allowAccessible="false"
                               allowAssemble="false"
                               allowFillInExistingForm="false"
                               allowHighPrint="false"
                               allowModify="false"
                               allowModifyAnnotation="false"
                               allowPrint="false"
                               allowTextExtraction="false"
                               encryption="RC4-128"
                               ownerPassword="PDF3DSDK"
                               userPassword="PDF3DSDK" />
04     <Assembly>
05         <InputFileName value="box.3ds"/>
06     </Assembly>
07 </pdf3d:InputParameters>
```

Line 3: `<EnableDocumentSecurity>` parameter is responsible for enabling the security options. This single tag manages a wide variety of options. Most of the parameters are binary true/false variables except for encryption and passwords. Possible values for “encryption” parameter are “RC4-40”, “RC4-80”, “AES-128” and “AES-256”.

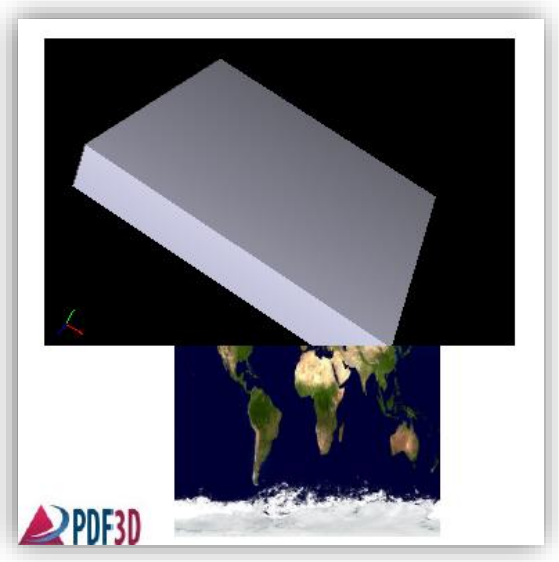
15. Importing External Media

Single `<DrawImage>` tag allows the user to import a wide variety of media formats into the PDF file ranging from static picture files to flash animations. However please note that all of the media will be placed underneath the 3D viewport (except flash, please see below). So, if a picture intersects with the viewport, a portion of the picture will be hidden by the viewport unless if its background is set to transparent (as shown in the figure below):

To import picture files into the current document:

```
1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <pdf3d:InputParameters ... >
3     <PageSize height="500" width="500" />
4     <Margins top="20" bottom="190" left="26" right="29"/>
5     <Assembly>
6         <InputFileName value="box.3ds"/>
7     </Assembly>
8     <DrawImage file="earthmap.jpg" bottom="10" left="150"
                                width="250" height="250" />
9 </pdf3d:InputParameters>
```

Line 8: Displays a standard jpeg file on the page. "`bottom`" and "`left`" fields specify how far the image is to be placed relative to the bottom and left edges of the document page respectively. If the specified value of "`width`" or "`height`" parameter turns out to be less or greater than the actual image width and height then Pdf3d.io will stretch the original image to the required dimensions. Currently JPEG, PNG, TIFF, BMP formats are supported (32-bit BMP is currently not supported). Following figure shows the output of the above code:



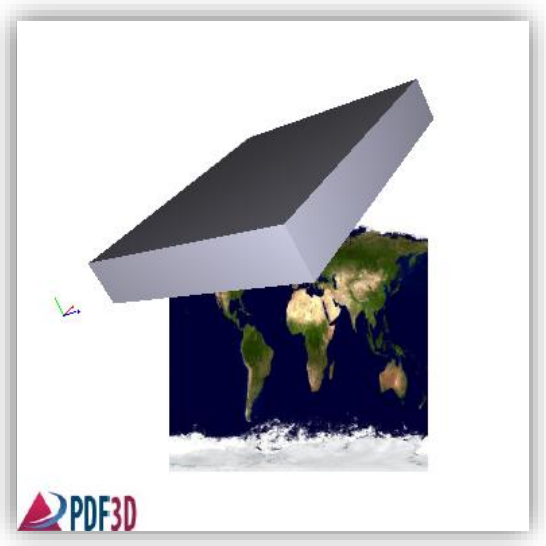
You can also set the viewport background to transparent. Hence any image file that is drawn underneath the viewport will show through:

```
01 <?xml version="1.0" encoding="iso-8859-1"?>
02 <pdf3d:InputParameters ... >
03     <SetTransparentBackground value="true" />
04     <UseBackgroundColor>
05         <Color alpha="0" blue="255" green="255" red="190" />
06     </UseBackgroundColor>
07     <PageSize height="500" width="500" />
08     <Margins top="20" bottom="190" left="26" right="29"/>
09     <Assembly>
10         <InputFileName value="box.3ds"/>
11     </Assembly>
12     <DrawImage file="earthmap.jpg" bottom="60" left="150"
                            width="250.0" height="250" />
13 </pdf3d:InputParameters>
```

Line 3: Sets the background transparent mode to true.

Line 5: Set the alpha in the background color to "0" in order to actually make the background fully transparent.

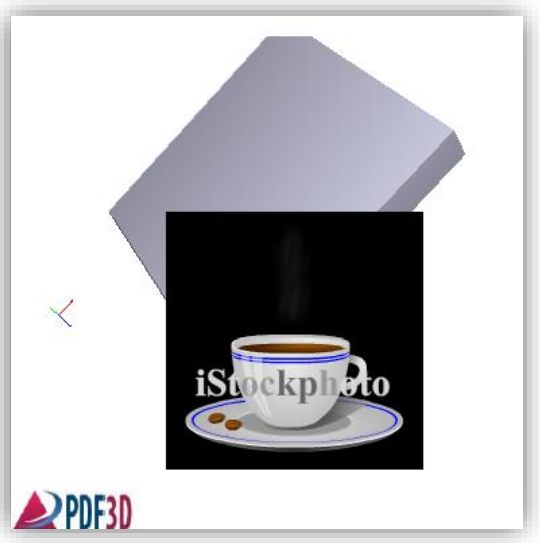
Above code produces following output:



Similarly importing animated flash files, sounds and videos that are embedded in swf file can be easily imported via same `<DrawImage>` tag:

```
01 <?xml version="1.0" encoding="iso-8859-1"?>
02 <pdf3d:InputParameters ... >
03     <SetTransparentBackground value="true" />
04     <UseBackgroundColor>
05         <Color alpha="0" blue="255" green="255" red="190" />
06     </UseBackgroundColor>
07     <PageSize height="500" width="500" />
08     <Margins top="20" bottom="190" left="26" right="29"/>
09     <Assembly>
10         <InputFileName value="box.3ds"/>
11     </Assembly>
12     <DrawImage file="istock-coffee.swf" bottom="60" left="150"
                            width="250.0" height="250" />
13 </pdf3d:InputParameters>
```

However the media contained in the swf will always be rendered on TOP of the viewport as shown in below:



16. Drawing Elements

In this section we will use transparent viewport backgrounds for the purpose of visual clarity.

a. Text and Hyperlinks

Consider the following code for input.xml:

```
01 <?xml version="1.0" encoding="iso-8859-1"?>
02 <pdf3d:InputParameters ... >
03     <SetTransparentBackground value="true" />
04     <UseBackgroundColor>
05         <Color alpha="0" blue="255" green="255" red="190" />
06     </UseBackgroundColor>
07     <PageSize height="500" width="500" />
08     <Margins top="20" bottom="190" left="26" right="29"/>
09     <Assembly>
10         <InputFileName value="box.3ds"/>
11     </Assembly>
12     <Watermark text="PDF3D-SDK" />
13     <DrawText bottom="100" left="40" value="PDF3D DrawText">
14         <Font family="Times New Roman" file="" bold="true"
                 italic="false" size="54.0" underline="false" />
15         <Color alpha="250" blue="0" green="0" red="0" />
16     </DrawText>
17     <DrawHyperLink bottom="20" left="100" text="PDF3D Link"
                                          url="www.pdf3d.com" />
18 </pdf3d:InputParameters>
```

Lines 3-8: Set up transparent background and viewport dimensions as explained in section 9.

Line 12: Set watermark on top of the 3D scene. Desired watermark text is given by the “text” field.

Line 13: `<DrawText>` renders specified text at position given by “bottom” and “left”. You can specify the font and color by enclosing `<Font>` (line 14) and `<Color>` (line 15) tags respectively within `<DrawText>`.

Line 17: Renders hyperlink on the specified position. It will use the most recent settings used by `<DrawText>` tag by default.

Following image shows final output:

b. Lines and Curves

Following code shows how to draw basic lines, paths, shapes and curves:

```
01 <?xml version="1.0" encoding="iso-8859-1"?>
02 <pdf3d:InputParameters ... >
03     <SetTransparentBackground value="true" />
04     <UseBackgroundColor>
05         <Color alpha="0" blue="255" green="255" red="190" />
06     </UseBackgroundColor>
07     <PageSize height="500" width="500" />
08     <Margins top="20" bottom="190" left="26" right="29"/>
09     <Assembly>
10         <InputFileName value="box.3ds"/>
11     </Assembly>
12     <DrawLine x1="10" y1="10" x2="300" y2="300" />
13     <DrawCircle fill="false" bottom="100.0" left="250.0" width="150.0" height="150.0" />
14     <DrawCircle fill="false" bottom="100.0" left="150.0" width="250.0" height="70.0" />
15     <DrawRect fill="false" bottom="50.0" left="100.0" width="150" height="80" />
16     <DrawArc fill="false" angle1="40.0" angle2="80.0" radius="170.0" x="50.0" y="50.0" />
17 </pdf3d:InputParameters>
```

Lines 3-8: Set up transparent background and viewport dimensions as explained in section 9.

Line 12: Draws a line from start point `(x1,y1)` to ending point `(x2,y2)`.

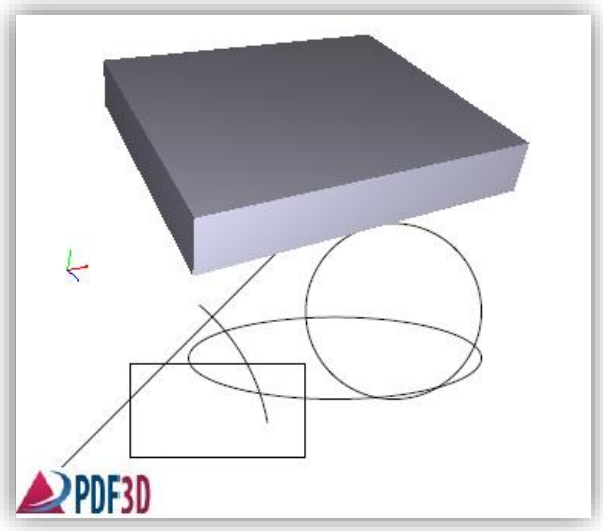
Line 13: `<DrawCircle>` tag draws a circle at location specified by bottom and left parameters. Exactly equal width and height yield a perfect circle.

Line 14: You can easily vary the width and height to make an ellipse.

Line 15: <DrawRect> draws a rectangle from the lower left point (bottom, left) having given "width" and "height".

Line 16: This tag draws an arc (portion of a circle) of given radius from center point (x, y).

For more drawing commands please see the input specification XML tag documentation. Following figure shows the output of above code:



c. Stroke and Fill

You can specify stroke and fill color and set stroke weight for all of the above-mentioned drawing commands:

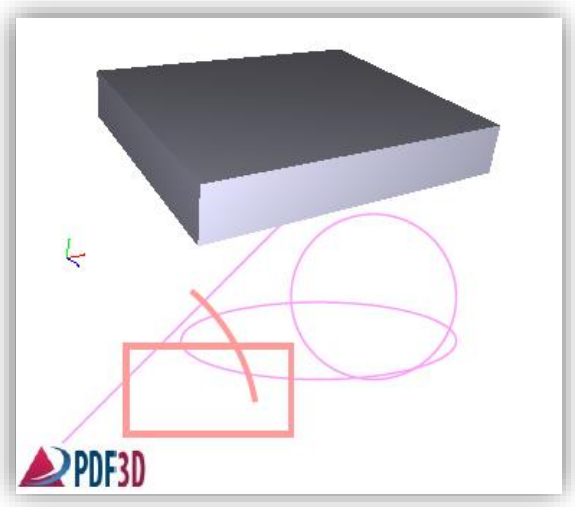
```
01 <?xml version="1.0" encoding="iso-8859-1"?>
02 <pdf3d:InputParameters ... >
03     <SetTransparentBackground value="true" />
04     <UseBackgroundColor>
05         <Color alpha="0" blue="255" green="255" red="190" />
06     </UseBackgroundColor>
07     <PageSize height="500" width="500" />
08     <Margins top="20" bottom="190" left="26" right="29"/>
09     <Assembly>
10         <InputFileName value="box.3ds"/>
11     </Assembly>
12     <SetCurrentColor>
13         <Color alpha="255" blue="255" green="155" red="255" />
14     </SetCurrentColor>
15     <SetCurrentLineWidth width="2.0" />
16     <DrawLine x1="10" y1="10" x2="300" y2="300" />
17     <DrawCircle fill="false" bottom="100.0" left="250.0" width="150.0" height="150.0" />
18     <DrawCircle fill="false" bottom="100.0" left="150.0" width="250.0" height="70.0" />
19     <SetCurrentColor>
20         <Color alpha="255" blue="155" green="155" red="255" />
21     </SetCurrentColor>
22     <SetCurrentLineWidth width="5.0" />
23     <DrawRect fill="false" bottom="50.0" left="100.0" width="150" height="80" />
24     <DrawArc fill="false" angle1="40.0" angle2="80.0" radius="170.0" x="50.0" y="50.0" />
25 </pdf3d:InputParameters>
```

Line 12: This tag sets the current color to the one specified by the <Color> tag (line 13).

Line 15: Sets pen stroke width to the one specified in the `<SetCurrentLineWidth>` tag.

Lines 19-22: Setting stroke color and weight to different values.

Output pdf:



d. 3D Annotations and PMI

Pdf3d.io allows you to add 3D annotations to your scene. For example, if you want to show the dimensions of the walls in a house model or lengths of window borders, you can easily do that with 3D annotations. Take a look at the code below:

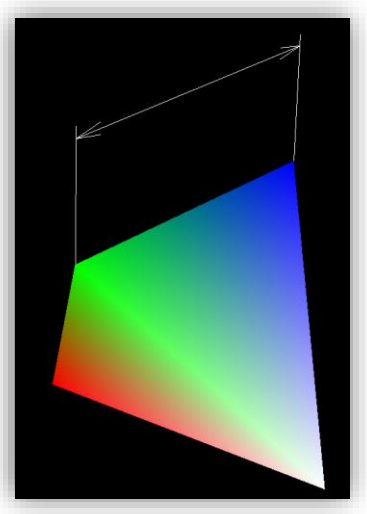
```
01 <?xml version="1.0" encoding="utf-8"?>
02 <InputParameters>
03     <CompressionFormat value="PRC"/>
04     <ExplodedView value="false"/>
05     <Assembly>
06         <InputFileName value="tetrahedron.wrl"/>
07     </Assembly>
08     <PMIMarkup name="linearSize">
09         <PMILinearSize>
10             <Anchor x="2.5" y="-0.5" z="1"/>
11             <Opposite x="2.5" y="0.5" z="1"/>
12             <Anchor2 x="0.5" y="-0.5" z="-1"/>
13         </PMILinearSize>
14     </PMIMarkup>
15 </InputParameters>
```

Line 8: `<PMIMarkup>` is the header tag for specifying annotations and their settings.

Line 9: `<PMILinearSize>` specifies the simple PMI label with anchor. It takes a few parameters which are optional. Please see input specification documentation for full details.

Line 10-12: Specifies the starting and ending points of 3D label.

Above code results in the following output:

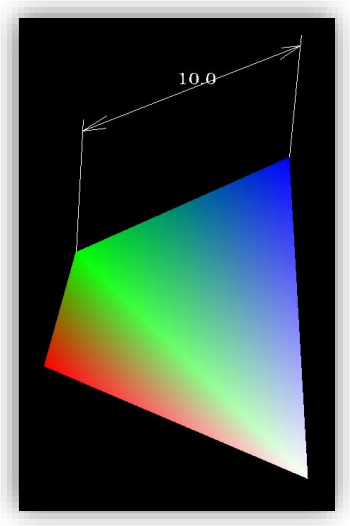


You can also specify text labels to the annotations as shown below:

```
01 <?xml version="1.0" encoding="iso-8859-1"?>
02 <InputParameters>
03     <CompressionFormat value="PRC"/>
04     <ExplodedView value="false"/>
05     <Assembly>
06         <InputFileName value="tetrahedron.wrl"/>
07     </Assembly>
08     <PMIMarkup name="linearSize">
09         <PMILinearSize>
10             <Anchor x="2.5" y="-0.5" z="1"/>
11             <Opposite x="2.5" y="0.5" z="1"/>
12             <Anchor2 x="0.5" y="-0.5" z="-1"/>
13             <Text sizeType="markupFixedSize" value="10.0"/>
14         </PMILinearSize>
15     </PMIMarkup>
16 </InputParameters>
```

Line 13: Specifies the text label to be displayed onto the annotation. `sizeType` parameter tells whether it is of fixed size or is it zoomable. `value` parameter specifies the actual text.

Yielding following output:



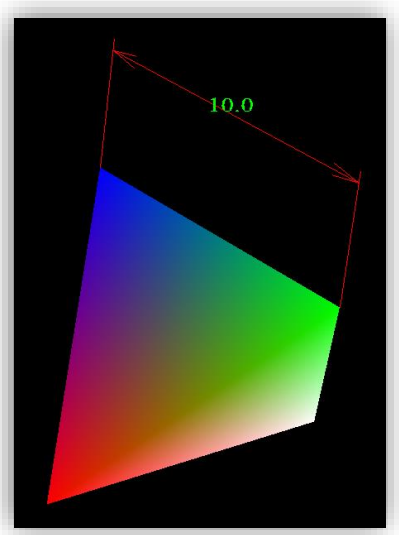
You can also change the color of the line and text:

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <InputParameters>
03     <CompressionFormat value="PRC"/>
04     <ExplodedView value="false"/>
05     <Assembly>
06         <InputFileName value="tetrahedron.wrl"/>
07     </Assembly>
08     <PMIMarkup name="linearSize">
09         <PMILinearSize>
10             <Anchor x="2.5" y="-0.5" z="1"/>
11             <Opposite x="2.5" y="0.5" z="1"/>
12             <Anchor2 x="0.5" y="-0.5" z="-1"/>
13             <Text sizeType="markupFixedSize" value="10.0"/>
14             <LineColor red="255" green="0" blue="0" />
15             <TextColor red="0" green="255" blue="0"/>
16         </PMILinearSize>
17     </PMIMarkup>
18 </InputParameters>
```

Line 14: Specifies the line color.

Line 15: Specifies text color.

Following output shows the result of above code:



17. Generating Tables

Tables are an integral part of any report generation system. Pdf3d.io provides intuitive mechanisms to create complex tables with little effort. Take a look at the following input.xml source code:

```
01 <?xml version="1.0" encoding="iso-8859-1"?>
02 <pdf3d:InputParameters ... >
03     <PageSize width="842" height="595"/>
04     <Margins top="100" bottom="50" left="26" right="172"/>
05     <DataTable>
06         <ColWidths>
07             <ColWidth width="54"/>
08             <ColWidth width="90"/>
09         </ColWidths>
10         <Row></Row>
11         <Row>
12             <Cell>Name</Cell>
13             <Cell>PDF3D SDK</Cell>
14         </Row>
15         <Row>
16             <Cell>Version</Cell>
17             <Cell>2.4.0</Cell>
18         </Row>
19     </DataTable>
20     <Assembly>
21         <InputFileName value="box.3ds"/>
22     </Assembly>
23 </pdf3d:InputParameters>
```

Line 5: All of the table generation tags and commands are enclosed within the <DataTable> tag.

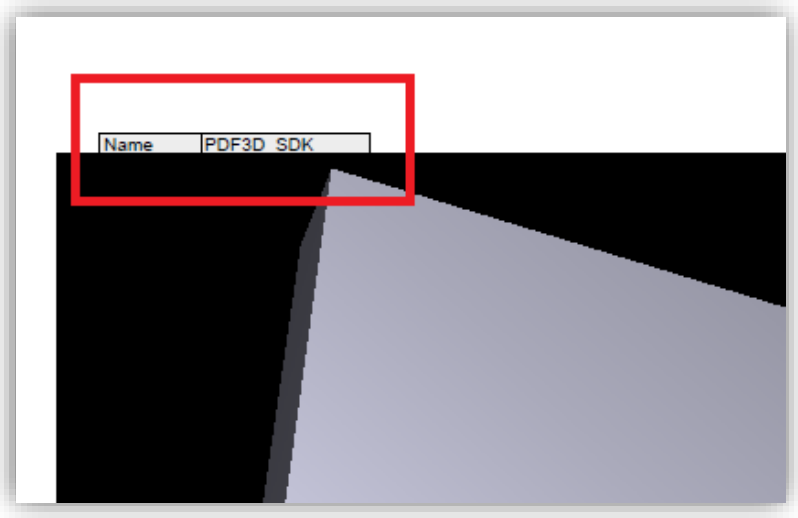
Line 6-9: <ColWidths> tag encloses a number of columns and their respective widths. The number of columns in a table depends on the number of <ColWidth> tags given inside <ColWidths>. In this case the table has two columns as only two <ColWidth> tags are given inside of <ColWidths>.

Line 11-14: <Row> adds a row to the table. This tag contains a number of <Cell> tags. <Cell> tags define the value of the current cell. You can add as many cells in a row as there are columns in the table.

Line 15-18: Another row added to the table.

Line 19: End of current table.

Above code yields following output:



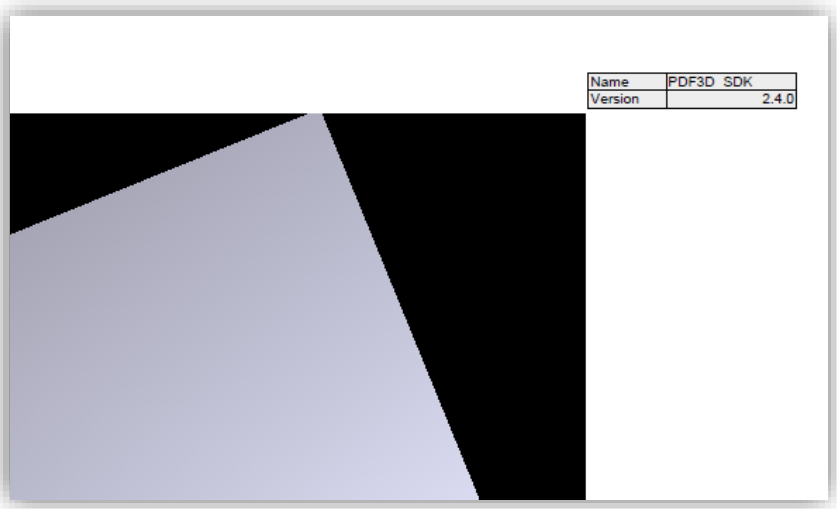
As you can see by default the Pdf3d.io places the table behind the 3D viewport. We have to adjust its position in the above mentioned code as follows:

```
01 <?xml version="1.0" encoding="iso-8859-1"?>
02 <pdf3d:InputParameters ... >
..     ....
05     <DataTable>
06         <Settings>
07             <TopMargin value= "52"/>
08             <LeftMargin value= "672"/>
09             <RightMargin value= "0"/>
10             <BottomMargin value= "80"/>
11         </Settings>
..         ....
25     </DataTable>
26     <Assembly>
27         <InputFileName value="box.3ds"/>
28     </Assembly>
29 </pdf3d:InputParameters>
```

Line 6: `<Settings>` tag is responsible for overriding table size, color, fonts and a lot of other customizable parameters. For a full reference please see the **pdf3d.io** input xml specification documentation.

Line 7-10: Specify top, left, right and bottom of the table relative to the current page.

Please note that if the current page is not large enough then the **pdf3d.io** will publish the table onto a new page of right dimensions. Following figure shows adjusted table:



18. Report Generation

Report elements such as 3D views, customized text, drawing elements, embedding multimedia have been discussed briefly in this document. These simple concepts can be extended easily to create extremely complicated and polished reports. Please see the `sample-report*.pdf3dsettings` for further details.

19. Embedded X3D Model Content within XML

The X3D interface is normally used to load and convert 3D model data in an external .x3d file. However, in the PDF3D.IO implementation, it is possible to put inline X3D inside the pdf3dsettings XML avoiding any external file. The key settings to enable this mode are illustrated by a red box:

```
<Assembly>
  <InputFileName value="Embedded X3D Content"/>
  <X3D>
    <Scene>
      <Transform DEF='BOX' translation='2.0 3.0 4.0'>
        <Shape>
          <Appearance>
            <Material diffuseColor="1.0 0 0"/>
          </Appearance>
          <Box size="2 2 2"/>
        </Shape>
      </Transform>
    </Scene>
  </X3D>
</Assembly>
```

The InputFile should be set to the special keyword value "*Embedded X3D Content*", then followed an X3D tag inside of the assembly, rather than specifying any external file.

20. Adding JavaScript for Dynamic Documents

JavaScript can be embedded into the PDF document, and executed when the document is opened or the user makes some type of interaction on the page. JavaScript can be added in three places:

a) As event actions triggered by buttons, check boxes, forms
b) As 3D events by mouse or keyboard interaction with the 3D view port
c) As general page and document level javascript to setup modes or cross-couple 3D with 2D.

For a button triggering a JavaScript Action, add a "*script*" tag:

```
<WidgetPushButton name="MyButton" caption="My Button">
        <Rectangle>
            <Position bottom="220" left="830"/>
            <Size width="100" height="30"/>
        </Rectangle>
        <Script>
            console.println("hello world ");
        </Script>
</WidgetPushButton>
```

For 3D view interaction, add 3D JavaScript by:

```
<JavaScript>
       c3d = getAnnots3D(0)[0].context3D;
       …
</JavaScript>
```

For 2D page and document level JavaScript, add:

```
<DocumentLevelJavaScript name="myDocJS">
    <Body>
        app.runtimeHighlight=false;
    </Body>
</DocumentLevelJavaScript>
```

21. Dynamic Library (DLL) In-Memory XML Conversion

In many of the supported scripting and programming languages the conversion DLL can be loaded and called where the parameters are dynamically created by in-memory XML strings. A quick example of how a typical C# .NET program might load and call the DLL is illustrated by this code snippet:

Load:

```
[DllImport("pdf3d.io.core.dll", CallingConvention = CallingConvention.StdCall)]
```

Use with:

```
string xmlParamString = "<?xml version=\"1.0\" encoding=\"iso-8859-1\"?>";
xmlParamString += "<InputParameters>";
```

```
xmlParamString += "<OutputFileName value=\"";
xmlParamString += this.outputFilename.Text;
xmlParamString += "\"/>";
xmlParamString += "<Assembly>";
xmlParamString += "<InputFileName value=\"";
xmlParamString += this.inputFilename.Text;
xmlParamString += "\"/>";
xmlParamString += "</Assembly>";
xmlParamString += "</InputParameters>";

result = PDF3DIOWrapper.PDF3D_IO_ConvertDotNet(xmlParamString,…);
```

22. Package Integration

The flexible architecture allows Pdf3d.io to operate on a production computer. Depending on the operating system, a subset of supplied files may be copied to a production computer, the converter processing program or dynamic library itself, plus one or more plugins, and supporting libraries for each plugin.

Each plug-in has its own specific dependencies. For example:

PDF3DCoinPlugin.dll has dependency from coin3d.dll, simage1.dll, libsndfile.dll files.

PDF3DGrdPlugin.dll has dependencies from VTK library binaries (it is dependent from VTK library version). Typically it uses vtkCommon.dll, vtkGraphics.dll, vtkHybrid.dll, vtkImaging.dll, vtkFiltering.dll, vtkIo.dll. It is however advised to ship all VTK binaries, as some part of VTK library are dependent from another parts of VTK library.

PDF3DStlPlugin.dll has no dependencies.

PDF3DVtkPlugin.dll has dependencies from VTK library binaries, similar to PDF3DGrdPlugin.dll.

PDF3DOsgPlugin.dll has dependencies from OSG (OpenSceneGraph) library. It is better to ship all of the OSG's binaries. Also OSG library is dependent from OpenThreads library. You may need to ship OpenThreads as well library if you plan to re-distribute PDF3DOsgPlugin.

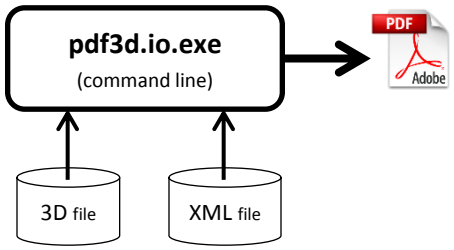
In addition to the above library files, appropriate runtime will be required. Also, 3rd party libraries can require additional binaries (for example, OSG library can be built by MSVC2005 and require MSVC2005 runtime C library; or be built by MSVC2010 and require MSVC2010 runtime C library).

After the related files have been copied onto the new system, you must then set the "PDF3D_LICENSE_PATH" environment variable to the location where license is residing, or use the PDF3D License utility to configure operations. For further information please contact PDF3D support.

23. Various Integration Alternatives

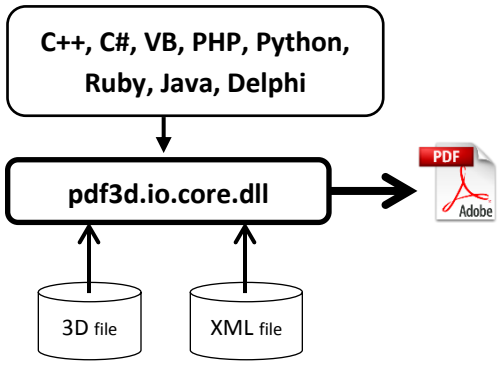
pdf3d.io is flexible enough to be integrated with a wide range of systems and configurations. Following scenarios are presented as guideline and are not at all exhaustive:

a.



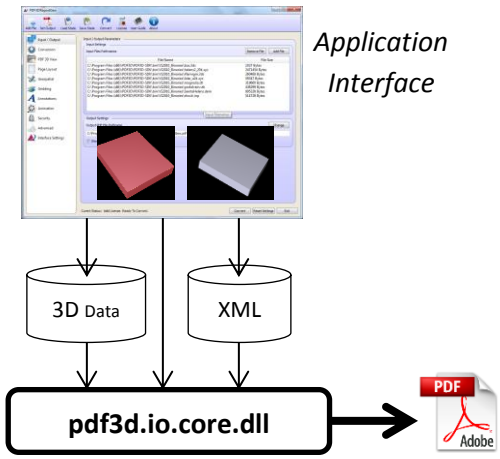
3D models and input XML configuration files can directly be fed into the **pdf3d.io.exe** console based application. It then produces output PDF and saves the file onto the hard drive.

b.



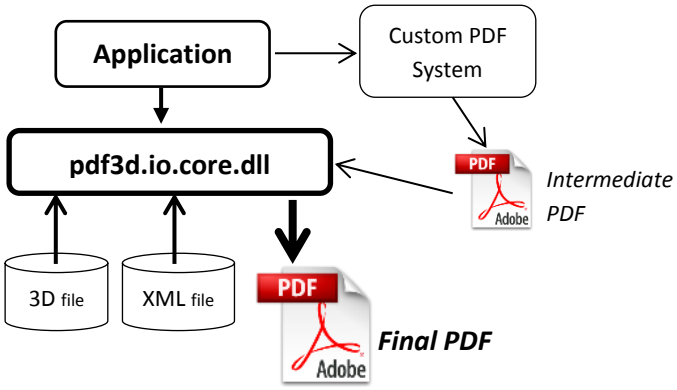
Languages like C++, C#, VB, PHP, Python, Ruby, Java and Delphi can directly interface the Pdf3d.io dynamic library (name may vary depending upon the operating system under consideration. For this document please assume the windows operating system). Languages may call API functions accessible through **pdf3d.io.core.dll** to load both 3D data and corresponding input xml. You can however bypass the standard Pdf3d.io convenience functions and replace them with your own, for example you may want to write your own model loader or custom xml parser for parsing and extracting data from input.xml. The loaded data can easily be passed to the **pdf3d.io** dynamic library to generate highly polished PDF documents. You also have the option to generate your own XML code dynamically and pass that to the **pdf3d.io** as string argument.

c.



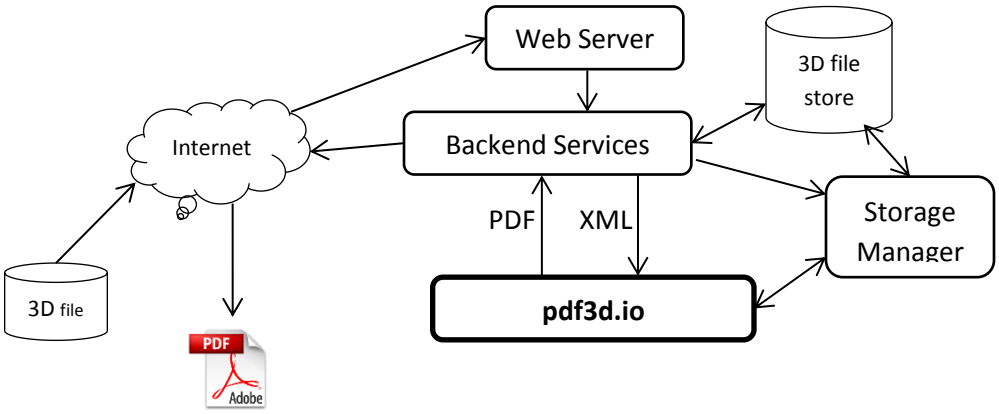
You can use customized WYSIWYG application to generate 3D geometry data and pass it directly into the xml library. This can be via memory or via external storage etc. **pdf3d.io** will take care of the conversion process and output pdf file depending on the given configurations.

d.



Application can have its own intermediate PDF generation system. Afterwards this PDF is fed into the **pdf3d.io**. **pdf3d.io** then inserts any 3D data into the intermediate PDF and finally outputs a final PDF file containing user configured PDF options as well as 3D data.

e.



pdf3d.io can be installed on an online webserver. The user can then upload the 3D model and configure conversion parameters via dynamic webpage interface. The PHP services send the 3D model and related parameters to **pdf3d.io** and send back the output PDF file for the user to download. This could be configured by using the command-line or libraries described above, or by directly using the RESTful API which provides the http web services layer directly.

24. Migration from PDF3D XML Server

The **pdf3d.io** system replaces and is a major new upgrade to the PDF3D XML Server product. For those applications wishing to migrate to the new pdf3d.io, the following short guide is provided for key changes. Most of the XML tags for conversion are forward compatible, and most of command-line programs take the same arguments, and the DLL exposes a very similar API. The most obvious change is new filenames, new installer package and the new REST API. The PHP web server example is now removed.

a. Filename Changes:

PDF3D XML Server	pdf3d.io
PDF3DXmlServer-Win.xml	pdf3d.io-Win.xml
PDF3DXmlServer-Linux.xml	pdf3d.io-Linux.xml
PDF3DXmlServer-Apple.xml	pdf3d.io-Apple.xml
PDF3DXmlServer.exe	pdf3d.io.exe
PDF3DXmlServer.xsd	pdf3d.io.xsd
pyPDF3DXmlServer.py	pyPDF3DIO.py
PDF3DXmlSrv.dll	pdf3d.io.core.dll

b. XML Schema Changes:

The state files, *.pdf3dsettings files XML header is changed to include the new XSD. ReportGen is able to load both new and old styles, however only saves the new format going forward.

```
<pdf3d:InputParameters xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.pdf3d.com/ns/2012/03/pdf3d.io pdf3d.io.xsd"
   xmlns:pdf3d="http://www.pdf3d.com/ns/2012/03/pdf3d.io"
   xsi:noNamespaceSchemaLocation="pdf3d.io.xsd">
```

c. License Feature Key Changes:

Previous versions of the PDF3D License Utility included “PDF3DXmlServer” on the available feature selector. This is now removed, replaced with “pdf3d.io”. Previous licenses will continue to work using the new pdf3d.io feature key.