Tornado
Web Services Guide

Version 2.11

March 2026

docmosis

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Overview

The *Tornado Web Services Guide* is intended for software application developers and integrators who need to produce formatted documents and reports from applications.

Docmosis Tornado provides an easy way to generate sophisticated and dynamic documents from virtually any application. The combination of web services and the Docmosis engine provides capability that can be integrated rapidly.

Tornado services are:

- ***Template Driven -*** you can change your templates any time with a word processor; upload and they will take effect immediately - wherever your application is running.
- ***Accessible -*** as long as you have network connectivity, you can render your documents using just about any development environment and deliver to multiple destinations.
- ***Secure –*** Tornado server runs where you determine it should run. It can be exposed only within your network as you determine. Tornado also provides access control to the configuration console and to the web service end-points.
- ***Flexible -*** the Docmosis engine provides rich template capabilities and output formats.
- ***Simple API -*** calls to the service are made using HTTP/HTTPS form posting. The *Render* service may be the only service that needs to be called. However, it is supported by further services for extra control.

1.1.1. Terminology Used in this Document

Term	Definition	Term	Definition
Template	A normal Microsoft Word or LibreOffice document containing special Docmosis fields.	**Fields/ Placeholders**	Docmosis specific mark-up within the template that controls where data should be inserted.
Render	The process of merging data with a template to generate a document.	**Converter**	A single computer process that performs one render request at a time.
Access Key	A unique string of characters sent by the application calling the API to verify it has permission to use the service(s).		

1.1.2. Getting Started

To use the Tornado Web Service you will need to:

1. Download and install Tornado.
2. Open the Tornado console using a web browser.
3. Enter a license key and specify folders for a working area and your templates.
4. Place your templates in the folder you specified.
5. Connect your application to the Web Service provided by Tornado using the code samples we provide.

The remainder of this document discusses using the Tornado Web Service via the API.

1.1.3. Related Reading

The *Tornado Installation & Configuration Guide* provides everything you need to know about getting started with Tornado, from downloading and installing the software to configuring your settings and generating a document using test templates and data.

The *Tornado Template Guide* provides fundamental details on the creation of templates. Refer to this document to ensure the data you send to Tornado matches the data required by the template.

1.2. Key Concepts

1.2.1. Character Encoding

All data passed to Tornado should be UTF-8 encoded. This provides a great balance between flexibility and compatibility. If you pass data containing special characters, then you will need to ensure you are UTF-8 encoding it, otherwise you'll get strange characters in your resulting documents.

1.2.2. Fonts in your Templates

Your templates should only use fonts that are available on the server where Tornado runs. If you use fonts which are not installed on the server, then you may see unexpected font substitutions in your PDF documents or inaccurate page references when using indexes or tables of content.

1.2.3. Dynamic and Stock Images

When Tornado generates a document containing images, the images can be sourced in three different ways:

Sent with your data: To send images with your data, they should be Base64 encoded and included in your data like any other textual information.

See Including Base 64 Dynamic Image Data on page 21 for more information.

Sourced from the templates location: “Stock” images are images which are placed in the Template area and dynamically sourced and inserted during document generation. This is ideal for logos and signatures which change only occasionally or there is a set to select from.

Tornado will retrieve the images when needed, so they don’t need to be streamed each time.

See section 2.3.6.3 "Stock" Image Data on page 22 for more information.

Sourced from a URL: Image data can also be dynamically sourced from URL references in your data.   This means your data has a URL reference to an image and Tornado fetches and inserts the image during document generation.

See section 2.3.6.2 Image Data from URLs on page 21 for more information.

1.2.4. Document Storage

Tornado provides the ability to send files to:

1. The local file system
2. The calling application
3. Email destinations

Documents can be rendered directly into these storage locations using the `render` service.

1.2.5. Template Merging

The render process is powerful enough to merge multiple templates into a single document. Templates may reference other templates dynamically (via data) or statically (in the template itself). This provides a mechanism for inserting common content across multiple templates. See the *Tornado Template Guide* for information about how to reference one template from another.

1.2.6. Production vs Development Mode

Tornado provides the option to operate in a forgiving manner (development mode) or in a very strict manner (production mode). The intention is that in development mode you are allowed to produce documents that contain errors, helping you to locate the error and make the necessary adjustments.

In production mode, no document with detected errors will be produced. Instead, the operation will fail with diagnostic information so you can be assured that documents will never be delivered that have fundamental errors in processing.

The mode is chosen on a render-by-render basis by passing the devMode parameter to the render call.

1.3. Troubleshooting

The FAQ section of the Docmosis Resources website (https://resources.docmosis.com) may help with troubleshooting problems when using the Tornado Web Service API.

2. THE DEVELOPER API

2.1. Fundamentals

Tornado offers a REST-based API. You can find more information about REST at: WikiPedia REST.

All calls to Tornado are made using HTTP or HTTPS POST requests. You can write code to call the API directly, use the Docmosis Tornado OpenAPI specification (downloadable from the Tornado console) or use a third-party toolset like the Java Jersey Client (http://jersey.java.net) creating your own requests. There is example code in various languages available on the Docmosis web site.

Your application will use URLs to POST requests to the Tornado host running in your environment. For example, the URL to render might look like:

```
https://localTornado1:8080/api/render
```

and the POST would contain the instructions and data for the render.

2.2. Response Codes and Messages

For every call you make to Tornado, you should first check the response code to determine whether the call succeeded or failed. Once you know whether the call succeeded or not, you can choose whether or not to check for further information in the response body.

Tornado returns status codes as follows:

Status Code	Definition
200	Successful operation
400	Your Docmosis request is not valid
500	A server error has occurred
404	Invalid URL (not found)

Other 4** and 5** response codes may also occur. You should always confirm that you received a 200 response before assuming success.

Tornado also returns information about the result in JSON or XML format as follows:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

Each call may also return additional information in the response information, as indicated in the sections to follow.

2.3. The Render Service

The `render` service is the document production 'work-horse', and it is typically the main service you need to call from your application. You can call the Render service with data and instructions indicating which template to use, the formats you require, where to send the result, and more.

Render works in *production* mode by default, meaning that any errors in the template or data supply are considered fatal and the render call will fail with an error. The default behaviour can be changed using the `devMode` flag (recommended: `devMode=true` for dev/test environments).

2.3.1. Service URL

```
/render
```

2.3.2. Request Headers

2.3.2.1. Content-Type

There are different ways to call the render service based on `content-type`. Set the `content-type` in request as follows:

Content Type	Description
`Multipart/form-data`	Parameters are passed as separate parameters. The `data` parameter may be omitted if using the flag `strictParams=false`. In this case, data items can

Content Type	Description
	be mixed with known request parameters. See `strictParams` for more information.
`application/x-www-form-urlencoded`	Parameters are passed as separate parameters url-encoded. The `data` parameter may be omitted if using the flag `strictParams=false`. In this case, data items can be mixed with known request parameters. See `strictParams` for more information.
`application/xml`	A single XML document string provides instructions and data (see examples below).
`application/json`	A single JSON document string provides instructions and data (see examples below).

Choose the `Content-type` that is easiest for you to work with.

2.3.2.2. Access Key

The `accessKey` (API Key) can be specified as a request header or a body parameter (see below).

2.3.3. Request Body Parameters

There are many parameters to control the render method, but most are optional. Please see the details in the table below for each parameter.

As an example, using the `application/json` content type a simple JSON format request could look like this:

```
{"templateName":"template1.doc",
 "outputName":"result.pdf",
 "data":{"title":"Company Profile Report",
         "scope":"Initial Scoping Report"}}
```

You can see the data and instructions are combined into a single JSON structure.

The same request in XML format would look like:

```
<?xml version="1.0" encoding="utf-8"?>
<render templateName="template1.doc" outputName="result.pdf">
  <data>
```

```
      <report title="Company Profile Report"
              scope="Initial Scoping Report"/>
    </data>
  </render>
```

The table below details the settings and options for the render request.

2.3.3.1. Mandatory Parameters

Parameter	Description	Default
`templateName`	The name of the template to use. Multiple templates can be specified using the ; character to separate the templates. With multiple template names, a single `outputName` for PDF output results in a single combined PDF. In all other cases, a ZIP file will be returned containing each rendered document.	
`outputName`	The name to give the rendered document. If no format is specified (see `outputFormat`), the format of the resulting document is derived from the extension of this name. For example, "resume1.pdf" implies a PDF format document. The name may be supplied without an extension (eg "resume1") and the `outputFormat` parameter will specify the format(s) to return. If multiple template names are specified, multiple (matching) output names can be specified to create a zip of named outputs.	

2.3.3.2. AccessKey Parameter

Parameter	Description	Default
`accessKey`	Your access key if specified in the Tornado configuration. **Mandatory**, if specified. Note the accessKey can be alternatively provided using a http header.	

2.3.3.3. Data Parameters

Parameter	Description	Default
`data`	The Data that will populate the document. This may be etiher XML or JSON format. The type of data given determines the format of the response. Also see `strictParams` about interaction between data and parameters.	
`stylesInText`	If set to "y","yes" or "true", your data will be parsed looking for html-like mark-up. The following mark-up is supported: • Bold e.g. "`this is <b>bold</b>`" • Italics e.g. "`this is <i>italics</i>`" • Underline e.g. "`this is <u>underline</u>`" • Cell Colouring e.g. "`<bgcolor="#ff0000"/>`" The bgcolor tag must be at the beginning of your field data and the template field must be inside a table cell to take effect.	`false`
`strictParams`	Controls whether extra parameters in the request should be treated as data or as an error. If true, the render service will return an error if a parameter is specified that is not listed in this table. The error message lists the acceptable parameters. If false, parameters not listed in this table are allowed and are added to the data used to populate the template. In this case, data values included as parameters will take precedence over any matching value passed via the data parameter.	`true`

2.3.3.4. Email Delivery Parameters

Parameter	Description	Default
`mailSubject`	If sending email, this will be used as the subject line of the email.	
`mailBodyHtml`	If sending email, this will be used as the body of the email and will be sent as html format.	
`mailBodyText`	If sending email, this will be used as the body of the email and will be sent as text.	

Parameter	Description	Default
`mailNoZipAttachments`	If this is set to true, any email attachments will be attached as individual files rather than as a single zip (when multiple formats are being used).	`false`

2.3.3.5. Pdf Specific Parameters

Parameter	Description	Default
`pdfArchiveMode`	*Note: depreciated, please see* `pdfVersion` Create pdf documents in PDF-A mode for long term storage. Note this setting disables certain PDF features such as password protection and external hyperlinks.	`false`
`pdfVersion`	Set the PDF version to use. By default, the engine will use the highest normal PDF version available (eg PDF 1.6), but this can be changed to use long term archive specifications as required (eg PDF/A-1b, PDF/A-2b, PDF/A-3b).	
`pdfUniversalAccessibility`	Create pdf documents with universal accessibility settings. Default = false. Note that when set to true, a document will be generated even if there are accessibility problems within the template. Items to check include: • The document title is set. • The document language is set, or all styles in use have the language property set. • All images, graphics, OLE objects have an alternate (alt) text or a title. • Tables do not contain split or merged cells. • Only integrated numbering is used, no manual numbering. For example, do not type "1.", "2.", "3." at the beginning of paragraphs. • Hyperlink texts without the underlying hyperlinks. • The contrast between text and the background meets the WCAG specification. • No blinking text. • No footnotes or endnotes. • Headings must increase sequentially with no	`false`

Parameter	Description	Default
	skips, for example, you cannot have Heading 1, Heading 3, and no Heading 2. • Text does not convey additional meaning with (direct) formatting.	
`pdfWatermark`	If specified, PDF documents will have the specified text added as a watermark across the document.	
`pdfWatermarkColor`	If a watermark is specified, this font color will be used. Must be a valid hexadecimal color, eg "#FF0000" for red.	
`pdfWatermarkFontSize`	If a watermark is specified, this font size will be used.	
`pdfWatermarkFontName`	If a watermark is specified, this font will be used. If the font is not installed on the server, then a substitute will be used.	
`pdfWatermarkRotation`	If a watermark is specified, this rotation angle will be used. This must be a value between 0 and 360.	
`pdfTagged`	If specified, the PDF documents will have extra information inserted to assist with low-vision readability tools. The alt-text for images in particular becomes "readable"	`false`
`pdfSkipEmptyPages`	If false, blank pages due to odd/even breaks will be included in the PDF.	`false`
`pdfRestrictPassword`	Set the password for further security restrictions. Setting this also enables the further restrictions.	
`pdfRestrictPrinting`	Restrict printing of the created PDF: 0 = cannot be printed 1 = print only low resolution 2 = print in full quality Requires that `pdfRestrictPassword` has been set.	`2`
`pdfRestrictEditing`	Restrict editing of the created PDF: 0 = no edits allowed 1 = pages can be inserted and rotated 2 = form fields can be filled in 3 = form fields can be filled and comments can be added	`4`

Parameter	Description	Default
	4 = above 1-3 enabled, but page extraction disabled Requires that `pdfRestrictPassword` has been set.	
`pdfRestrictCopy`	Set whether copy of PDF content is disabled. “true” or “y” disables the copy. Requires that `pdfRestrictPassword` has been set.	
`pdfRestrictAllowAccessibilty`	Set whether content can be extracted by accessibility applications. “true” or “y” disables the accessibility access. Requires that `pdfRestrictPassword` has been set and also `pdfRestrictCopy` must be set.	
`pdfEInvoiceSpec`	The eInvoice Specification to use if creating an eInvoice. For example: `ZUGFeRDv2` (Uses EN 16931 profile) `ZUGFeRDv1` (Uses COMFORT profile) `Order-Xv1` (Uses COMFORT profile) `Factur-X-v1-EN16931` The `pdfEInvoiceAttachment` must also be provided and `pdfVersion` must be set to `PDF/A-3B`.	
`pdfEInvoiceAttachment`	The xml eInvoice file in base64 format. The provided xml is expected to be valid and conforming to the specification provided in pdfEInvoiceSpec.	

2.3.3.6. Other Parameters

Parameter	Description	Default
`compressSingleFormat`	Optionally choose to zip the result when a single output document is produced. The zip archive will contain a document in the specified format with a name based on `outputName + outputFormat`. The resulting zip file name will be the `outputName` with the .zip extension appended as required. This option is ignored if more than one `outputFormat` is specified. Positive values are "y", "yes" and "true" (case-insensitive).	`false`

Parameter	Description	Default
`devMode`	Document production can run in development or production respectively. If set to "y", "yes" or "true" this operation will work in "dev" mode, meaning that if something is incorrect in the template, data or instructions Tornado will do it's best to produce a document. Such a document may contain errors such as missing images and data, and wherever possible, Tornado will highlight problems to indicate the failure. In production mode, errors in document rendering will result in a failure result only, and no document will be produced. The production mode is to ensure that a bad document is never produced/delivered to a recipient. The default mode is production (that is, `devMode` is off).	`false`
`outputFormat`	The format(s) of the rendered document. This can be a single format, or a ';' semi-colon delimited list. Multiple formats imply a zip file and `outputName` will have .zip appended as required. Files inside zip will be named using `outputName` and will have the format-specific extension appended as required. Valid options are pdf, doc, odt, rtf, html, txt. Note that the header `X-Docmosis-Zip-Created` (*described below*) will indicate if a zip file has been created and returned.	
`passwordProtect`	If specified, this parameter will set the password for PDF and DOCX files created by the render. The password is used when opening the document. Use with care as setting the password means the recipient must know the password to read the document. Note: `pdfArchiveMode` will disable any password setting for PDF documents.	
`requestId`	Any string you would like to use to identify this job. This string will be returned in responses.	
`storeTo`	Specify where to send the resulting document. If no specification is given, "stream" is assumed and the result will be streamed back to the requester, otherwise the ';' semi-colon delimited list of destinations will receive the result.	`stream`

Parameter	Description	Default
	Valid options are `stream, mailto, file` See section *2.3.4 Delivery Options* StoreTo Options for more details	
`streamResultInResponse`	If set to "y", "yes" or "true", the streamed result will be base64 encoded and included in the JSON or XML response under the key "`resultFile`". Note this only applies if the request includes (or implies) a "stream" result (see the `storeTo` parameter above).	`false`

2.3.4. Delivery Options

2.3.4.1. StoreTo Options

Tornado can render to a variety of destinations using the `storeTo` parameter. The table below summarizes the options.

Destination	Examples
`stream (default)`	Stream the document back to the caller. By default, this will be a binary stream direct response. If the parameter `streamResultInResponse` is specified in the request, the document will be base64 encoded and included in the JSON or XML response instead under the key "`resultFile`".
`mailto`	Email* the document to the specified address. Please note the other email related parameters (described in section 2.3.3.4 Email Delivery Parameters) to set the subject and other features of the email.
`file`	Store the document in the specified file (which is local to the Tornado server).

* Note: email delivery is not guaranteed. It is generally less secure, less reliable and slower than other delivery mechanisms since much of the delivery process is outside of the control of Docmosis.

The following table provides simple examples of using the storeTo options.

Example	Examples
`stream (or not specified)`	Document is returned to the caller
`mailto:bob@example.com`	Email the document to bob@example.com

Example	Examples
`file:/documents/doc1.pdf`	Create a pdf file at the location /documents/doc1.pdf

2.3.4.2. Delivering to Multiple Destinations

Multiple destinations can also be specified at once, using a semi-colon to separate the destinations. For example:

```
stream;mailto:bob@example.com;mailto:anne@example.com
```

will deliver the document back to the calling application as well as emailing it to two recipients.

2.3.4.3. Overriding the Output Format

By default, all destinations will receive all formats specified by `outputFormat` (or implied by the `outputName` if `outputFormat` not specified). Each destination may override the defaults settings and specify what to receive using this style `"stream:<format[,format]>"` e.g. `"stream:pdf"`.

The following table shows some examples.

Destination	Examples
`stream:pdf`	Return a pdf to the caller
`stream:pdf,docx`	Return a zip to the caller containing a pdf and a docx
`mailto:bob@example.com:docx,pdf`	Send an email with docx and pdf attachments
`file:c\:documents\doc1.zip:pdf,odt`	Create a zip file in the specified location containing pdf and odt documents

2.3.5. Response

2.3.5.1. Response Body

The response from the render method varies depending on:

1. whether it succeeds or fails

2. whether the request specifies *stream* as a destination (implied or explicit)

Calling applications must check the status code of the response to determine what to do next. Any status other than **200** means the render failed, and error information will be available in the response body.

The following cases show the types of check extract the response information as follows:

1. **On Success (status code = 200) and `storeTo` includes "stream":**

the body of the response is the binary document stream.

2. **On Success (status code = 200) and `storeTo` excludes "stream":**

the body of the response is a JSON object containing:

Field	Definition
`succeeded`	"true".
`requestId`	The `requestId` given in the render request (if any).

3. **On failure (status code <> 200):**

the body of the response is a JSON object containing:

Field	Definition
`succeeded`	"false"
`shortMsg`	A short message about the cause of the failure.
`longMsg`	A more descriptive message about the failure if applicable. It may be blank.
`requestId`	The `requestId` given in the render request (if any).

2.3.5.2. Response Headers

For the render service, the following headers may be returned to assist response processing.

Header	Notes
`X-Docmosis-Server`	An identifier for this service “tornado”.
`X-Docmosis-RequestId`	Returned if the original request provided a `requestId`.
`X-Docmosis-PagesRendered`	The number of pages rendered
`X-Docmosis-Document-Errors-Detected`	Whether or not errors were detected in the document rendering. This is useful in “dev mode” to

Header	Notes
	be able to quickly determine whether the document created is known to have errors.
`X-Docmosis-Zip-Created`	Set to true if the result being returned is a zip file.

2.3.6. Image Data

2.3.6.1. Including Base 64 Dynamic Image Data

Image data can be included in the data stream. This is achieved by Base64 encoding the image data and assigning the value to the key which your template image is using. The image data (i.e. its value) must be prefixed by "`image:base64:`" so that Tornado can identify and decode it as required.

As an example, an image in a template marked with "img_pic1" expects to find an image called pic1 specified in the data. In JSON format it might look like:

```
"data":{"pic1":"image:base64:mawv0dga423g0345....."
```

or (in RFC2397 format):

```
"data":{"pic1":"data:image/jpeg;base64,mawv0dga423g0345....."
```

Base64 encoding is outside the scope of this guide, but it is easy to find libraries and reference material to help you create it.

Image data is typically large compared with textual information. Keep in mind the impact on your bandwidth and document size when using image data. If there are only a few options for an image, consider using different templates, sub-templates or separately uploading "stock" images.

2.3.6.2. Image Data from URLs

Image data can also be dynamically sourced from URL references in your data. This is disabled by default and so needs to be enabled by adding a setting in the Tornado Configuration Custom Settings. The setting is a single line, semicolon delimited whitelist of allowed URL patterns:

```
docmosis.external.resources.whitelist=http://eg.com/images;
```

The whitelist is used to match the start of the URL, so note that you can narrow down the allowed URLs to specific paths under the domain. If you were to use both http and https you would need to add both.

As normal, your template would have marked up the image with a name that ties to your data, for example “pic1”. To dynamically replace the image “pic1” with an image from a URL, the data would look something like

```
"pic1":"[imageUrl:http://image.site/Image103.png]"
```

The above data would cause Docmosis to fetch the image from:

```
http://image.site/Image103.png
```

and put it into the document dynamically.

2.3.6.3. "Stock" Image Data

Where an image is used repeatedly in document generation, such as logos or signatures, you have options about how to obtain the image:

1. stream the image with every render – this is wasteful of processing and bandwidth if the image is repeated.
2. put all the options for the image into the template then have Tornado dynamically strip out the undesired image(s) during the document render. This can be done using conditional sections (See the *Cloud Template Guide* for more information).
3. place the images in the same folder as the Templates - these are called "stock" images. You can reference these images in your data providing an efficient way to get images into documents.

To use a stock image, you firstly need to place it in the template folder. This is done the same way as you would place templates into the configured template location.

With your stock image in place, you can reference it in your data using a key that matches your template image, and a specially formatted data value. The data value should be formatted so the file name and location of the image is prefixed with “userImage:” and the entire data value surrounded by square brackets.

For example, if your template has an image named img_pic1 and you've uploaded an image called face1.jpg, your data key is pic1 and your data value is "[userImage:face1.jpg]". In JSON format, your data would look something like this:

```
"pic1":"[userImage:face1.jpg]"
```

When you upload an image, you may also use a path-like structure for organising your images. For example, you may have uploaded the image with the name:

```
projectA/first/face1.jpg
```

in which case, the request above would look like this:

```
"pic1":"[userImage:projectA/first/face1.jpg]"
```

2.4. The Get Template Structure Service

Get Template Structure retrieves the structure of a template that has been uploaded. The structure returned describes fields, repeating and conditional sections etc. The primary purpose of this method is to allow automated processing based on what is actually in a template (such as creating dynamic data forms etc).

2.4.1. Service URL

```
/getTemplateStructure
```

2.4.2. Request Headers

2.4.2.1. Content-Type

The content-type for the call may be "`application/json`", "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.4.2.2. Access Key

The `accessKey` (API Key) can be specified as a request header or a body parameter (see below).

2.4.3. Request Body Parameters

Field	Definition
`templateName`	The name of the template.
`accessKey`	Your access key if specified in the Tornado configuration.
`stringify`	If set to "y", "yes" or "true" then the json result will be stringified, otherwise the json response object will be sent in full. Defaults to false.

2.4.4. Response Body

On success (status=200), the body of the response will contain the data structure below.

On failure, the response will contain at least the succeeded and shortMsg fields.

Field	Definition
`succeeded`	"true" or "false"
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.
`templateStructure`	A JSON format description of the template structure. Template elements are returned in an array, for example: “templateStructure”:[{"type":"field", "fieldIdx":0, "text":"name", "dataRefs":["name"]}, {"type":"field", "fieldIdx":1, "text":"address", "dataRefs":["address"]}] The above example show two fields were found and each correlates with a lookup on a single data item (“dataRefs”). A template element such as an expression-field may correlate with multiple data references. For example, the template expression: <<{firstName + ‘ ‘ + lastName}>> is reported as a field with two dataRefs: {"type":"field", "fieldIdx":2, "text":"{firstName + ' ' + lastName}", "dataRefs":["firstName","lastName"]} The types reported are: “field”, “repeat”, “condition”, “image”, “templateRef” corresponding to matching structures in the template. Each type has it’s own index which is global to the document. “text” shows the string as presented in the template and dataRefs reports the identified data elements correlated to the template element. The result also indicates the nesting of elements. The template content: <<cs_hasPeople>> <<rs_people>> <<name>> <<es_>> <<es_>>

Field	Definition

	Would be expressed as:
	[
	  {
	   "type": "condition", "conditionIdx": 0, "text": "cs_hasPeople",
	   "dataRefs": ["hasPeople"],
	   "contains": [
	    {
	     "type": "repeat", "repeatIdx": 0, "text": "rs_people",
	     "dataRefs": ["people"],
	     "contains": [
	      {
	       "type": "field", "fieldIdx": 0, "text": "name",
	       "dataRefs": ["name"]
	      }
	     ]
	    }
	   ]
	  }
	]

2.5. The Convert Service

The convert service allows files to be converted between formats. The process is a simple conversion with no concept of templates and data. It applies to spreadsheet, presentation and drawing types of documents.

2.5.1. Service URL

```
/convert
```

2.5.2. Request Headers

2.5.2.1. Content-Type

The content-type for the call is "`multipart/form-data`".

2.5.2.2. Access Key

The `accessKey` (API Key) can be specified as a request header or a body parameter (see below).

2.5.3. Request Body Parameters

Field	Definition
`file`	The file to convert
`outputName`	The name of the new file to create. The extension is used to determine the type of file to create. For example, "result.pdf" causes a PDF document to be created.
`accessKey`	Your access key if specified in the Tornado configuration.

2.5.4. Response Body

The converter service responds with a simple indication of success or failure using the standard structure:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

2.6. The Ping Service

The Ping service allows monitoring systems to detect whether the Tornado server has been started.

The "status" service is more useful in general. The ping service may be helpful with certain types of diagnostics.

2.6.1. Service URL

```
/ping
```

2.6.2. Request Headers

2.6.2.1. Content-Type

The content-type is not specified, since the call takes no parameters.

2.6.3. Request Body Parameters

None.

2.6.4. Response Codes

200 – successfully pinged

2.6.5. Response Body

There is no response body returned.

2.7. The Status Service

The Status service allows monitoring systems to determine the operational status of Tornado, including whether it is ready to render documents.

2.7.1. Service URL

```
/status
```

2.7.2. Request Headers

2.7.2.1. Content-Type

The content-type is not relevant.

2.7.3. Request Body Parameters

None

2.7.4. Response Codes

200 – the server is ready to render documents. Additional information is also returned, see table below.

2.7.5. Response Body

Field	Definition
`ready`	"true" or "false".
`message`	A short message indicating details of the current status

`detail`	A JSON structure as follows:
	• converterCountInUse – then number of converters currently in use
	• converterCountOnline – the number of converters currently online
	• converterCountOffline – the number of converters configured but not currently online
	• conveterCountTotal – the total number of configured converters
	• uptimeSeconds – how long Tornado has been running
	
	An example :
	`{`
	` "detail": {`
	`   "converterCountInUse":"0",`
	`   "converterCountOffline":"0",`
	`   "converterCountOnline":"4",`
	`   "converterCountTotal":"4",`
	`   "uptimeSeconds":"20"`
	` },`
	` "message":"ready",`
	` "ready":"true"`
	`}`

2.8. The Get Sample Data Service

The Get Sample Data service allows sample data to be generated for a template based on the current structures in the template. The sample data can be created in JSON or XML format and can then be fed back to the render service to generate populated documents.

The service creates values like “value1”, “value2” for each field element.

If the template has an error in it, Docmosis will generate a blank data set.

2.8.1. Service URL

```
/getSampleData
```

2.8.2. Request Headers

2.8.2.1. Content-Type

The content-type for the call may be "`application/json`", "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.8.2.2. Access Key

The `accessKey` (API Key) can be specified as a request header or a body parameter (see below).

2.8.3. Request Body Parameters

Field	Definition
`templateName`	The name of the template for which to create sample data
`accessKey`	Your access key if specified in the Tornado configuration.
`format`	If blank or “json”, JSON format data will be returned. Otherwise XML format data will be returned
`stringify`	If set to "y", "yes" or "true" then the json result will be stringified, otherwise the json response object will be sent in full. Defaults to false.

2.8.4. Response Body

The service responds with a JSON structure as follows:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. It may be blank, or, In the case of an error, a long error message.
`templateSampleData`	JSON or XML formatted sample data that can be used to populate the template.
`templateDetails`	A JSON structure providing details of errors in the template: • `templateHasErrors` – true or false • `templateFirstError` – a string of the first error message in the template if errors are present .

Docmosis Pty Ltd

Address
Suite 8 / 5 Hasler Road,
Osborne Park,
WA 6017 Australia

Website
https://www.docmosis.com

Resources
https://resources.docmosis.com

docmosis