Cloud Web Services Guide

Version DWS2

Create Documents and Reports Fast from Templates

Cloud Web Services Guide

Copyrights

Trademarks

Microsoft Word and MS Windows are registered trademarks of the Microsoft Corporation.

http://office.microsoft.com/en-us/default.aspx

http://www.microsoft.com/windows/

Adobe® PDF is a trademark of the Adobe Corporation.

http://www.adobe.com/products/acrobat/adobepdf.html

LibreOffice is a trademark of LibreOffice contributors and/or their affiliates

http://www.libreoffice.org

Contents

Preface

Welcome to the *Cloud Web Services Guide*. This manual is intended for document application developers and integrators who need to produce richly formatted document and reports from applications.

The *Cloud Web Services Guide* provides information for making the most of Docmosis Cloud services.

Related Reading

Please refer to the *Docmosis Cloud Template Guide* for information about how to create and maintain templates.

1 Introduction

Docmosis Cloud Services provide an easy way to generate sophisticated and dynamic documents from virtually any application. The combination of web services and the Docmosis engine provides a great capability that can be integrated surprisingly fast.

Whether you are developing a large enterprise application or a trend setting mobile application, Docmosis Cloud services allows you to produce great documents based on merging your templates and data.

1.1 Feature Summary

Docmosis cloud services are:

1. *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.
2. *Accessible* - as long as you have internet connectivity you can render your documents using just about any development environment and delivered to multiple destinations
3. *Secure* - all communications between Docmosis and your application are SSL encrypted and Docmosis doesn't hold your data or documents after processing.
4. *Reliable* - built on the Amazon Web Services platform providing security and reliability.
5. *Powerful* - the Docmosis engine provides amazing template abilities and output formats
6. *Simple API* - calls to the service are made using HTTPS/SSL form posting. The *render* service is the only service that need be called.

1.2 Quick Overview

Using the cloud services is easy:

1. Sign up to get an account
2. Upload / modify your templates, or start working with some of the samples
3. Use the example code to make calls to the `render` service to produce your documents

When you sign up, you are provided with a unique token (*"access key"*) which you use to access the services. Your access key is a private identifier and should be treated with care like a password.

Templates can be uploaded and downloaded using either calls to the Docmosis services, or using the Docmosis web site.

Now that you have a taste for what is involved, go ahead and give it a go. The remainder of this document detail the developer API.

Note

There is an extended developer API available providing product-level Docmosis extensions. This allows your application to sign up users automatically, provide default and shared templates, send emails from "you" and much more. Please contact Docmosis Support to get started.

1.3 Character Encoding

All data passed to Docmosis Services 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.

1.4 Fonts in your templates

Your templates will need to use standard/common fonts. If you use fonts which the Docmosis Service does not have, then you may see unexpected font substitutions in your PDF documents or inaccurate page references when using indexes or tables of content.

1.5 Dynamic and Stock Images

Docmosis Cloud Services allow you to stream image data with which your templates can be populated. This is done by base64 encoding your image data and putting it in your data like any other textual information.

See section 2.4.5 Including Base-64 Dynamic Image Data for more information.

The services also provide the concept of "stock" images which can be uploaded to the site and dynamically inserted during document creation without the need to send the image data every time you make the request. This is ideal for logos and signatures which change only occasionally or there is a set to select from.

See section 2.4.7 Including "Stock" Image Data for more information.

Image data can also be dynamically sourced from URL references in your data. This feature requires white-listing of the URLs for your account for Docmosis support staff:

See section 2.4.6 Including Dynamic Image Data from URLs for more information.

1.6 Document Storage

Some Docmosis accounts are provisioned with the ability to store Documents within the cloud. Storage may be within Docmosis' own storage areas, or to remote Amazon S3 locations owned and managed by your own Amazon accounts.

Documents can be rendered directly into these storage locations using the render service. There is a set of service end-points allowing you to upload, list, download and store these files too. The specifics of the API for these methods are detailed in the remainder of this document.

1.7 Template Merging

The render process is powerful enough to merge multiple templates into a single set of documents. Templates may reference other templates dynamically (via data) or statically (in the template itself). This provides an ideal mechanism for inserting common content across multiple templates.

See section 2.5 The Upload Template Service for more information.

1.8 Important Reading

The Docmosis Template Guide is essential reading to making the most of the services. It provides fundamental details about how to create templates. Please note that the web services have a new feature allowing field mark-up to be done using PLAIN TEXT instead of merge fields.

1.9 Production vs Development Mode

Some services provide 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.

2 The Developer API

2.1 Fundamentals

The Docmosis cloud services is a REST-based API. You can find more information about REST here WikiPedia REST. All calls to Docmosis are made using HTTPS POST requests. You can write code to call the API directly 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.

Alternatively you may use a toolset that can read a Web Application Description Language (WADL) and generate the code to access the service automatically.

The Docmosis WADL is available here:

```
https://dws2.docmosis.com/services/rs/application.wadl
```

You can examine the WADL by pointing a browser to it to see the definition of the available services.

2.2 Response Codes and Messages

For every call you make to Docmosis services, 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 then choose whether or not to check for further information in the response body.

The Docmosis service 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.

Docmosis services also return 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.

Field	Definition
`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 service may also return additional information in the response information as indicated in the sections to follow.

In the case of some errors, the response body will NOT include the data structure containing "succeeded", "shortMsg" etc. This is because some errors indicate the call failed to even reach the Docmosis cloud. For example, a status code 404 indicates a bad URL in which case the status code alone is enough to indicate what the problem is.

2.3 A Quick Tour of the API

The API is divided into 4 areas:

1. Rendering - the most important service
2. Template Management - uploading, downloading and more
3. Stock Image Management - much the same as template management
4. Document/File Storage Management - uploading, downloading and more

Each of these areas is detailed in the sections to follow.

2.4 The Render Service

The `render` service is the document-production work-horse, and it is the only service you need to invoke from your application since template operations may also be carried out via the Docmosis web site. You invoke the render service with data and instructions indicating which template to use, what formats you would like, 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. You may override this with the `devMode` flag.

2.4.1 Service URL

`/render`

`/renderForm` (use this when data items are posted as key/value pairs in the api call, rather than in a "data" element).

2.4.2 Content-Type

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

Content Type	Description
`Multipart/form-data`	Parameters are passed as separate form parameters with multipart boundaries. The `data` parameter may be either XML or JSON. The "data" parameter may be omitted if using the renderForm end point.
`application/x-www-form-urlencoded`	Parameters are passed as separate form parameters url-encoded. The `data` parameter may be either XML or JSON. The "data" parameter may be omitted if using the renderForm end point.
`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 one that makes it easiest for you to work with.

2.4.3 Request 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",
 "accessKey":"xxx-my-access-key",
 "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"
 accessKey="xxx-my-access-key">
  <data>
   <report title="Company Profile Report" scope="Initial Scoping Re-
port"/>
  </data>
</render>
```

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

Parameter (bold=mandatory)	Description	Default
`accessKey`	Your unique access key you were given when you created your account.	
`templateName`	The name of the template to use. Template must have been uploaded previously with the template upload request.	
`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.	
`outputFormat`	The format(s) of the rendered document. ; delimited. Multiple formats imply a zip file and `ouputName` will have .zip appended as required. Files inside zip will be named using `outputName` and will have the	

Parameter (bold=mandatory)	Description	Default
	format-specific extension appended as required. Valid options are pdf, doc, odt, rtf, html, txt.	
`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 ; delimited list of destinations will receive the result. Valid options are `stream, mailto, storage, s3.` See section 2.4.4 StoreTo Options for more details	`stream`
`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`
`devMode`	Document production can run in development and 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 Docmosis will do it's best to produce a document. Such a document may contain errors such as missing images and data, and wherever possible, Docmosis 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, dev mode is off).	`false`
`data`	The data to populate the document with. This may be either XML or JSON format. The type of data given determines the format of the response. If you call the "`/renderForm`" variant of the render end-point (instead of "`/render`") the data field is not required. Instead data will be extracted from the submitted form parameters using the parameter name as the key.	
`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.	
`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`
`isSystemTemplate`	If set to "true", `templateName` refers to a System template, as opposed to your own template. System templates are managed by administrators.	`false`
`requestId`	Any string you would like to use to identify this job. This string will be returned in responses and so can be useful for matching responses when multiple requests are running in parallel.	
`sourceId`	Any ID you would like to associate with this render. This could be a device id, a project code or any meaningful piece of data you associate with this render. This value can be reported later with associated monthly counts. Limited to 150 characters.	
`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 eg "`this is <b>bold</b>`" - Italics eg "`this is <i>italics</i>`" - Underline eg "`this is <i>underline</i>`"	`false`

Parameter (bold=mandatory)	Description	Default
	- Cell Colouring eg "<bgcolor="#ff0000"/>This cell is now red. 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.	
passwordProtect	If specified, this parameter will set the password for PDF and DOC 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.	
pdfArchiveMode	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
pdfWatermark	If specified, PDF documents will have the specified text added as a watermark across the document.	
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
ignoreUnknownParams	If true, unknown parameters in the request are allowed and ignored. By default the render service will return an error if a parameter is specified that is not expected (defined by this table).	false
tags	A semi-colon delimited list of tags to record against this render. The tags can be later queried (using the getRenderTags end point) to retrieve stats such as page-counts and document-counts related to the tags.	
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

2.4.4 StoreTo Options

Docmosis can render to several destinations at once, and optionally send different formats for delivery to each destination. As a simple example:

```
stream:pdf;mailto:me@gogo.com:doc
```

which indiates a PDF document should be streamed back to the caller, and a DOC document should be emailed.

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>" eg "stream:pdf". If you wish to specify multiple email addresses, use multiple mailto: directives. Note that email behaviour is also determined by other parameters in the render call such as subject and body message.

The following table describes the available storage options.

Destination	Examples
stream	Stream the document back to the caller. By default this will be a binary stream direct response. If "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".

Destination	Examples
mailto	eMail the document to the specified address
storage	Store the document in the Docmosis cloud storage for your account
s3	Store the document in an external Amazon S3 location

The following table provides some examples

Destination	Examples

stream	`stream`
	`stream:pdf`
	`stream:pdf,doc`

mailto	`mailto:support@docmosis.com` `mailto:support@docmosis.com:pdf`
storage	`storage:mydocument1.pdf` `storage:documents/doc1.zip:pdf,odt`
s3	`s3:my.amazon.bucket,mydocument1.pdf` `s3:my.amazon.bucket,mydocument1.zip:pdf,odt`

The storage destinations may be repeated as required. For example multiple emails can be sent by specifying mailto:address1@my.com;mailto:address2@my.com.

2.4.5 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 key matches the marker in your template and the image data (ie its value) must be prefixed by "`image:base64:`" so that Docmosis 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.....", ...
```

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

Note

Image data is typically large compared with textual information. You 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.

2.4.6 Including Dynamic Image Data from URLs

Image data can also be dynamically sourced from URL references in your data. As normal, your template would have marked up the image with a name that ties to your data, for example "pic1".

The URL in the data should be prefixed by "`imageUrl:`" and be enclosed in square brackets (`[…]`).

To dynamically replace the image "pic1" with an image from a URL, the data would look something like:

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

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

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

and put it into the document dynamically.

Note

Note the use of URLs requires white-listing of the URLs on your account by the Docmosis support staff due to the potential impact on the cloud platform.

2.4.7 Including "Stock" Image Data

Where the same image is repeated in document production, such as logos or signatures you have options about how to obtain the image:

1. stream the image every time you render - this is wasteful of bandwidth and processing if the image is repeatedly the same.
2. put all the options for the image into the template then have Docmosis dynamically strip out the undesired image during the document render. This can be done using conditional sections (See the Docmosis Template Guide for more information).
3. upload the images in advance to your Docmosis account - these are called "stock" images. You can reference your uploaded images in your data providing an efficient way to get images into documents.

To use a stock image, you will need to upload it first. This can be done by logging into the Accounts module of the Docmosis web site and switching to the templates tab. Down the bottom you can upload images as well as templates. You can also use the API to upload images programmatically - see section 2.11 for more information.

Once your stock image has been uploaded to the cloud, you can reference it in your data using a key that matches your template image, and a specially formatted value. For example, if your template has the image named `img_pic1` and you've uploaded `face1.jpg`, your key is `pic1` and your value is "`[userImage:face1.jpg]`". In JSON format, your data would look something like this:

```
...
"data":{"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:

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

As a final note about stock images, product vendors may create images for you to utilise. These are known as "System Images" and they are referenced using "`systemImage`" instead of "`userImage`". At this time there are no stock images for standard Docmosis users.

2.4.8 Response Messages

The response from the render method varies depending on:

1. whether it succeeds or fails
2. whether your destinations include streaming back in your request

Remember, you should always check the status code first 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 you should perform to extract the response information:

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).

2.4.9 Response Header

For the render service, if you supply a `requestId` in the request this will always be returned in the *header* of the response in addition to the response message. This means whether the render succeeds or fails, streams back or not, you will always be able to use the header to determine the related request. This is particularly handy in scenarios where the request is run asynchronously by your code.

2.5 The Upload Template Service

This service allows you to upload templates to be rendered into documents. When you upload a template, Docmosis analyses it and will report any errors in the template at this time.

There are two categories of templates:

- User Templates - templates owned and managed by you
- System Templates - system managed templates that you may use, but only administrators can modify.

By default all template operations work with your templates, but you may set the `isSystemTemplate` flag to override the default behaviour.

Docmosis has a special capability to report errors in the template within rendered documents. This means that when you render a document you can see the error AND its location in the template. By default this capability is enabled (`devMode=true`) to assist with development. Any template uploaded in development mode with errors will not render unless the render also uses development mode .

Note

Uploading of templates can be done via the Docmosis web when you log into your cloud services account. This makes direct use of this service optional.

Note
Templates are stored on servers, encrypted at rest in the USA.

2.5.1 Service URL

`/uploadTemplate`

2.5.2 Content-Type

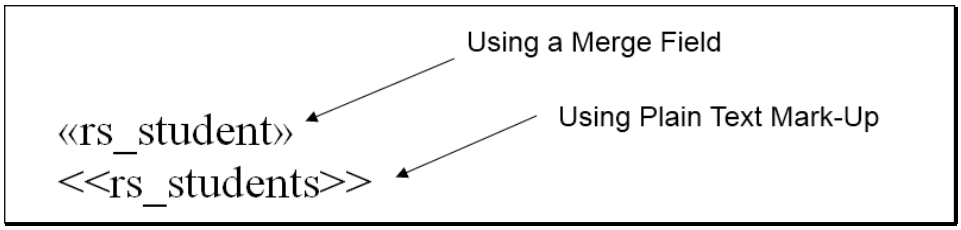
The content-type for the upload is "multipart/form-data".

2.5.3 Plain Text Markup

Docmosis normally uses merge fields for identifying fields in a template (details of how to build templates can be found in the Docmosis Template Guide). The Docmosis Cloud Services have a new capability allowing fields to be created using plain text mark-up, making template maintenance much simpler.

Merge fields can complicate templates because sometimes what you see in the document is not what the merge field is using, and sometimes the fields can be hidden entirely. Sometimes even inserting the first field is difficult with newer versions of MS Word.

The difference in appearance (using MS Word) between merge fields and plain text fields can be seen here:



The delimiters are set to << and >> by default, and they will be used when uploading templates. If your templates use << or >> for other purposes, then you will need to override the delimiters by setting the corresponding parameters when you upload.

2.5.4 Request Parameters

Field	Definition
`accessKey`	Your access key.
`templateFile`	The template file you are uploading.
`templateName`	An optional overriding name which may include a path.
`templateDescription`	A description for the template.
`isSystemTemplate`	If set to "y", "yes" or "true" this template will be uploaded as a system template. Only authorised users may modify system templates. Defaults to "false".
`devMode`	If set to "y", "yes" or "true" the upload is run in developer mode - meaning that Docmosis will do it's best to handle errors and report them within a rendered document to ease development. Defaults to "true".
`keepPrevOnFail`	If set to "y", "yes" or "true" the previous template of the same name will be left in place if the uploaded template has errors. If not specified (or "n", "no" or "false") the original template is always removed, even if this uploaded template has errors. This only has effect when devMode is disabled (since devMode is intended to allow templates with errors to be displayed by Docmosis). This parameter means that in production mode (non-developer mode) template uploads will not replace a working template with a bad template. Defaults to "false".
`fieldDelimPrefix`	If using plain text mark-up in your templates this specifies the

Field	Definition
	prefix delimiter identifying a field. The default is <<
`fieldDelimSuffix`	This is the closing delimiter for plain text fields. The default is >>
`normalizeTemplateName`	If set to "y", "yes" or "true" the template name given with be NFC normalized (Unicode NFC normalization). The default is false.

2.5.5 Response Messages

The response is an indicator of success or failure plus any further helpful information. When the upload is successful, details are returned under the `templateDetails` key.

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.
templateDetails (when succeeded = true)	The details of the template in JSON: `name` - the template file name `lastModifiedMillisSinceEpoch` - last modified in milliseconds `lastModifiedISO8601` - last modified yyyy-MM-dd'T'HH:mm:ssZ `sizeBytes` - the size in bytes `isSystemTemplate` - whether a system template ("true" or "false") `templatePlainTextFieldPrefix` - the prefix used when it was uploaded `templatePlainTextFieldSuffix` - the suffix used when it was uploaded `templateDevMode` - the dev mode setting used when it was uploaded `templateHasErrors` - true if the uploaded template has errors `templateDescription`- the description uploaded with the template if any `md5` – the md5 hash code for the template

2.6 The Get Template Service

Get template retrieves the template that was originally uploaded. Multiple templateName parameters can be specified to download multiple templates in one zip response (up to 100 in one request).

2.6.1 Service URL

`/getTemplate`

2.6.2 Content-Type

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

2.6.3 Request Parameters

To list templates, you need only supply your access key.

Field	Definition
`accessKey`	Your access key.
`templateName`	The name of the template. This parameter can be specified multiple times to download multiple templates (up to 100) in a zip file.
`isSystemTemplate`	Indicator as to whether the template is a system template or not (optional) - defaults to false.

2.6.4 Response Messages

On success (status=200), the body of the response will contain the binary stream for the template.

On failure, the response provides the following information:

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.

2.7 The Get Template Details Service

Get Template Details retrieves the details of a template that was originally uploaded, but not the template itself.

2.7.1 Service URL

`/getTemplateDetails`

2.7.2 Content-Type

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

2.7.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`templateName`	The name of the template.
`isSystemTemplate`	Indicator as to whether the template is a system template or not (optional) - defaults to false.

2.7.4 Response Messages

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.
templateDetails	The details of the template in JSON: `name` - the template file name `lastModifiedMillisSinceEpoch` - last modified in milliseconds `lastModifiedISO8601` - last modified yyyy-MM-dd'T'HH:mm:ssZ `sizeBytes` - the size in bytes `isSystemTemplate` - whether a system template ("true" or "false") `templatePlainTextFieldPrefix` - the prefix used when it was uploaded `templatePlainTextFieldSuffix` - the suffix used when it was uploaded `templateDevMode` - the dev mode setting used when it was uploaded `templateHasErrors` - true if the uploaded template has errors `templateDescription`- the description uploaded with the template if any `md5 - the md5 hash code for the template`

2.8 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).

2.8.1 Service URL

`/getTemplateStructure`

2.8.2 Content-Type

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

2.8.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`templateName`	The name of the template.

2.8.4 Response Messages

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. The simplest case will be a list of fields, for example:
	`"templateStructure":{`
	`"field.0":"firstName",`
	`"field.1":"lastName",`
	`}`
	
	The JSON keys use the terms "field", "repeat", "condition" and "image" to instruct what type of item it is, and then an index starting at 0.
	
	Importantly, items are in and order and nested structure matching the template. This means that fields within repeating sections will be depicted within a matching structure. For example:
	
	`"templateStructure":{`
	`"field.0":"firstName",`
	`"field.1":"lastName",`
	`"repeat.0.addresses":{`
	`"field.3":"addressLine1",`
	`"field.4":"addressLine2",`
	`}`
	`}`

Field	Definition

2.9 The List Templates Service

List templates lists the templates available to you, including system templates which are managed by vendors.

2.9.1 Service URL

`/listTemplates`

2.9.2 Content-Type

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

2.9.3 Request Parameters

To list templates, you need only supply your access key.

Field	Definition
`accessKey`	Your access key.
`includeDetail`	Include extra detail about parameters. Default=true.
`paging`	Whether or not to return results in pages. Default=false. If true, pages of 1000 records are returned.
`pageToken`	When paging is true, this token identifies the next page to retrieve. The page token is null for the first page. When the first page response returns, it contains the token required to request the next page.

2.9.4 Response Messages

The response includes the normal success indicator and messages as well as a JSON object containing the list of templates.

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.
templateListStale	"true" or "false". If Docmosis detects changes to the template list are in progress (such as updates or deletions) this flag will be set to "true" to indicate the list is not necessarily up to date. This is only

Field	Definition
	ever expected to be "true" for a short period after deletes or updates.
templateList	The list of templates in JSON format having attributes for each template: `name` - the template file name `lastModifiedMillisSinceEpoch` - last modified in milliseconds `lastModifiedISO8601` - last modified yyyy-MM-dd'T'HH:mm:ssZ `sizeBytes` - the size in bytes `isSystemTemplate` - whether a system template ("true" or "false") `*templatePlainTextFieldPrefix` - the prefix used when it was uploaded `*templatePlainTextFieldSuffix` - the suffix used when it was uploaded `*templateDevMode` - the dev mode setting used when it was uploaded `*templateHasErrors` - true if the uploaded template has errors `*templateDescription`- the description uploaded with the template `*md5` - the md5 hash code for the template `*These items are only returned when the includeDetail parameter was specified as true.`
`nextPageToken`	If returned, this specifies the token to use in the next call to listTemplates (parameter pageToken) to get the next page of results. This element is not present if the request was not-paged or there are no further pages.

The `templateList` is an array of objects giving details for each template in the list.

2.10 The Delete Template Service

The delete template service deletes the specified template. Multiple templateName parameters can be specified to delete multiple templates.

2.10.1 Service URL

`/deleteTemplate`

2.10.2 Content-Type

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

2.10.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`templateName`	The name of the template. This parameter can be specified multiple times to delete multiple files.
`isSystemTemplate`	Indicator as to whether the template is a system template or not (optional) - defaults to false.

2.10.4 Response Messages

The delete template 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.

2.11 The Image Upload Service

The upload image service works the same as the template upload service, but is significantly simpler since there are few options to set. Image uploading is used to create "stock" images that your data can reference when rendering and Docmosis will take care of placing them into your document. This saves bandwidth and time when rendering the same images frequently. An option to uploading "stock" images is to place the images into your template and selectively filter them out using conditional sections.

As for templates, there are two categories of images:

- User Images - images owned and managed by you
- System Images - system managed images that you may use, but only administrators can modify.

By default all image operations work with your own images, but you may set the `isSystemImage` flag to override the default behaviour if there are system-provided images available to you. At this time, Docmosis does not provide any system-level stock images.

2.11.1 Service URL

`/uploadImage`

2.11.2 Content-Type

The content-type for the call may be "multipart/form-data".

2.11.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`imageFile`	The file stream of the image.
`imageName`	An overriding name for the image which may include a path (eg `project1/stock/logo1.jpg`).
`imageDescription`	A short description for the image.
`isSystemImage`	Indicator as to whether the image is a system image or not (optional) - defaults to false.
`normalizeImageName`	If set to "y", "yes" or "true" the image name given with be NFC normalized (Unicode NFC normalization). The default is false.

2.11.4 Response Messages

The delete template service responds with a simple indication of success or failure as well as details about the uploaded image:

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.
imageDetails (when succeeded = true)	The details of the image in JSON: `name` - the image file name `lastModifiedMillisSinceEpoch` - last modified in milliseconds `lastModifiedISO8601` - last modified yyyy-MM-dd'T'HH:mm:ssZ `sizeBytes` - the size in bytes `isSystemTemplate` - whether a system template ("true" or "false") `md5` - the md5 hash code for the image

2.12 The List Images Service

List images lists the images available to you, including system images which are managed by vendors.

2.12.1 Service URL

`/listImages`

2.12.2 Content-Type

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

2.12.3 Request Parameters

To list images, you need only supply your access key.

Field	Definition
`accessKey`	Your access key.

2.12.4 Response Messages

The response includes the normal success indicator and messages as well as a JSON object containing the list of images.

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.
imageListStale	"true" or "false". If Docmosis detects changes to the image list are in progress (such as updates or deletions) this flag will be set to "true" to indicate the list is not necessarily up to date. This is only ever expected to be "true" for a short period after deletes or updates.
imageList	The list of images in JSON format having attributes for each image: `name` - the image file name `lastModifiedMillisSinceEpoch` - last modified in milliseconds `lastModifiedISO8601` - last modified yyyy-MM-dd'T'HH:mm:ssZ `sizeBytes` - the size in bytes `isSystemImage` - whether a system image ("true" or "false") `md5` - the md5 hash code for the image

The `imageList` is an array of objects giving details for each template in the list.

2.13 The Delete Image Service

The delete image service deletes the specified image. Multiple imageName parameters can be specified to delete multiple images.

2.13.1 Service URL

`/deleteImage`

2.13.2 Content-Type

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

2.13.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`imageName`	The name of the image. This parameter can be specified multiple times to delete multiple files.
`isSystemImage`	Indicator as to whether the image is a system image or not (optional) - defaults to false.

2.13.4 Response Messages

The delete template 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.

2.14 The Get Image Service

Get image retrieves the image that was originally uploaded. Multiple imageName parameters can be specified to download in a zip file (up to 100).

2.14.1 Service URL

`/getImage`

2.14.2 Content-Type

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

2.14.3 Request Parameters

To list templates, you need only supply your access key.

Field	Definition
accessKey	Your access key.
imageName	The name of the image. This parameter can be specified multiple times to download multiple images (up to 100) in a zip file.
isSystemImage	Indicator as to whether the image is a system image or not (optional) - defaults to false.

2.14.4 Response Messages

On success (status=200), the body of the response will contain the binary stream for the image.

On failure, the response provides the following information:

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.

2.15 The Get File Service

This service retrieves a file from the storage area associated with your account. The files that can be found in storage are those rendered directly to storage using the render service, or those uploaded using the Put File Service. The List Files Service can list the files available for download.

Note: the file storage feature of Docmosis is not available for all account types.

2.15.1 Service URL

`/getFile`

2.15.2 Content-Type

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

2.15.3 Request Parameters

To retrieve the file you must specify your access key and the name of the file you require.

Field	Definition
accessKey	Your access key.
fileName	The name of the file, optionally including its path.

2.15.4 Response Messages

On success (status=200), the body of the response will contain the binary stream for the file.

On failure, the response provides the following information:

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.

2.16 The Put File Service

This service stores a file in the storage area associated with your account. This allows you to store files of any type in your account.

Note: the file storage feature of Docmosis is not available for all account types.

2.16.1 Service URL

`/putFile`

2.16.2 Content-Type

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

2.16.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`file`	The file stream to upload.
`fileName`	An optional overriding file name which may also include a path.
`contentType`	An optional setting for the content-type of this file. Docmosis will attempt to work out the content type if not specified.
`metaData`	An option string of information to store with this file that can be retrieved with the file later.

2.16.4 Response Messages

The body of the response contains a success indicator and error details if the request failed:

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.

2.17 The List Files Service

This service list the files in the storage area associated with your account.

Note: the file storage feature of Docmosis is not available for all account types.

2.17.1 Service URL

`/listFiles`

2.17.2 Content-Type

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

2.17.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`folder`	An option starting folder (path). If not specified all files in your storage will be listed.
`includeSubFolders`	An optional specification as to whether you would like the list to include items within sub-folders. Defaults to false. "y", "yes" and "true" are all positive indicators.
`includeMetaData`	If "y", "yes" or "true" meta data for each file will be included in the results. Defaults to "false".

2.17.4 Response Messages

The response includes the normal success indicator and messages as well as a JSON object containing the list of files.

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.

Field	Definition
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.
storedFileListStale	"true" or "false". If Docmosis detects changes to the stored file list are in progress (such as updates or deletions) this flag will be set to "true" to indicate the list is not necessarily up to date. This is only ever expected to be "true" for a short period after deletes or updates.
storedFileList	The list of images in JSON format having attributes for each file: `name` - the file name `lastModifiedMillisSinceEpoch` - last modified in milliseconds `lastModifiedISO8601` - last modified yyyy-MM-dd'T'HH:mm:ssZ `sizeBytes` - the size in bytes `metaData` - the metadata stored with the file (if requested)

2.18 The Delete Files Service

The Delete Files service does the obvious - delete the specified stored files.

Note: the file storage feature of Docmosis is not available for all account types.

2.18.1 Service URL

`/deleteFiles`

2.18.2 Content-Type

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

2.18.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`path`	The name of the file or folder.
`includeSubFolders`	If "y", "yes" or "true" all files within the given path are deleted also.

2.18.4 Response Messages

The delete files 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.

2.19 The Rename Files Service

The Rename Files service allows files and folders to be renamed.

Note: the file storage feature of Docmosis is not available for all account types.

2.19.1 Service URL

`/renameFiles`

2.19.2 Content-Type

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

2.19.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`fromPath`	The original name of the file or folder.
`toPath`	The new name for the file or folder.

2.19.4 Response Messages

The rename files 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.

2.20 The Convert Document Service

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

2.20.1 Service URL

`/convert`

2.20.2 Content-Type

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

2.20.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`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.

2.20.4 Response Messages

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.

2.21 The Get Render Tags Service

The get render tags service allows statistics to be retrieved on renders that were tagged with user-defined phrases ("tags"). The statistics include page counts and document counts that have been collected against the tags, aggregated monthly. This may be useful for reporting the level of activity of a group of users, or a feature in your application.

2.21.1 Service URL

`/getRenderTags`

2.21.2 Content-Type

The content-type for the is "multipart/form-data" or "application/x-www-form-urlencoded".

2.21.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`tags`	The tags to query. This can be a single tag or a list of tags separated by the ; character. This parameter is mandatory.
`year`	The year on which to report statistics. Defaults to the current year.
`month`	The month on which to report statistics (1=Jan). Defaults to the current month.
`nMonths`	The number of months on which to report statistics. Defaults to 1. If more than one month is being reported, the months prior to the given year and month are included. This means that this call always reports up to the specified month.
`padBlanks`	If true (or 'y') zero values will be included where no data exists. This may make parsing the returned result easier since it will always contain values for the tags requested over the given time period by padding the data with zero-values as required. Defaults to false.

2.21.4 Response Messages

The get render tags 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. In the case of an error this will be a long error message. It may be blank.
renderTags	An array of objects as with values as follows: `year` – the year `month` – the month number `tags` – an array of objects as follows: `name` – the tag `countPages` – the number of pages rendered for this tag in the year and month `coundDocuments` – the number of documents rendered against this tag in the year and month

Field	Definition
	For each tag queried, there will be an entry for that tag in the year and month. There will also be a "combined" result showing the statistics for the combined set of tags queried. This combined tag will report the stats that match all of the tags specified in the call to the getRenderTags request (that is, it will the stats only for renders that included all the tags).

For example, if a call is made with the following parameters:

```
tags:  abc;def
padBlanks: true
year: 2017
month: 9
nMonths: 2
```

Then the following result shows and example JSON response:

```
{
  "succeeded": true,
  "renderTags": [
    {
      "year": "2017",
      "month": "8",
      "tags": [
        {
          "name": "abc",
          "countPages": "0",
          "countDocuments": "0"
        },
        {
          "name": "def",
          "countPages": "0",
          "countDocuments": "0"
        },
        {
          "name": "abc;def",
          "countPages": "0",
          "countDocuments": "0"
        }
      ]
    },
    {
      "year": "2017",
      "month": "9",
      "tags": [
        {
          "name": "abc",
          "countPages": "6",
          "countDocuments": "4"
        },
        {
          "name": "def",
          "countPages": "4",
          "countDocuments": "2"
        },
        {
          "name": "abc;def",
          "countPages": "2",
          "countDocuments": "1"
        }
      ]
    }
  ]
}
```

In the above output, we can see that we have two objects in the array of "`renderTags`", one for month 8 (Aug) and one for month 9 (Sep). Each object in the array contains the stats for each requested tag ("`abc`" and "`def`") and the combined tag ("`abc;def`"). There is no data for August but because `padBlanks` is `true`, the data is filled out with zeros.

In September, we can see that 6 pages were generated by renders with tag "`abc`", 4 pages with tag "`def`" and 2 pages with both "`abc`" and "`def`" tags.

2.22 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 created in JSON or XML format, and can then be fed back to the render service to generate populated documents.

The service is currently quite basic, creating data values like "value1", "value2" and it doesn't create sample data for fields in expressions or functions.

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

2.22.1 Service URL

`/getSampleData`

2.22.2 Content-Type

The content-type for the is "multipart/form-data" or "application/x-www-form-urlencoded".

2.22.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`templateName`	The name of the template for which to create sample data
`format`	If blank or "json", JSON format data will be returned. Otherwise XML format data will be returned

2.22.4 Response Messages

The get render tags 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.

Field	Definition
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.
templateSampleData	JSON or XML formatted sample data that can be used to populate the template.

2.23 The Ping Service

The ping service provides a direct check that the Docmosis REST web services are online and there is at least one Docmosis server listening. This is useful for diagnostics and for monitoring purposes. The response is empty, and the http response code 200 is the success indicator.

2.23.1 Service URL

`/ping`

2.23.2 Content-Type

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

2.23.3 Request Parameters

None.

2.23.4 Response Messages

There is no response body returned.