Cloud (DWS4)
Web Services Guide

Version DWS4

April 2024

docmosis

Trademarks
Docmosis is a registered trademark of Docmosis Pty Ltd.
https://www.docmosis.com

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

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Overview

The Cloud (DWS4) Web Services Guide is intended for software application developers and integrators who need to produce formatted documents and reports from applications.

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

The Cloud Services are:

- *Template Driven* - Docmosis templates can be changed at any time with a word processor and uploaded for immediate effect - wherever the calling application is running.
- *Accessible* – applications with internet connectivity can render documents and delivered to multiple destinations. Processing can be focussed within geographical region to meet privacy and data-sovereignty obligations.
- *Secure* – all communications between Docmosis and calling applications are SSL encrypted and Docmosis doesn't hold data or documents after processing.
- *Reliable* – multiple levels of redundancy have been built upon on the Amazon Web Services (AWS) platform, providing security and reliability. 24x7 internal and external monitoring integrated with Pingdom and Status.io allows customer visibility to Docmosis Cloud performance.
- *Flexible* - the Docmosis engine provides rich template capabilities and output formats.
- *Simple API* - calls to the service are made using HTTPS/SSL form posting. There is a single service endpoint (Render) that applications use to create documents. Deeper application integration can optionally make use the API to manage templates and images.

Templates can be uploaded and downloaded using either calls to the Cloud Service API or using the Cloud Console.

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 the insertion and removal of data and content.
Render	The process of merging data with a template to generate a document.	**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. Related Reading

The *Cloud Template Guide* provides fundamental details on the creation of templates. Refer to this document to ensure the data sent to Docmosis matches the data required by the template.

1.2. Key Concepts

1.2.1. Processing Locations

There are multiple locations where the Docmosis Cloud services operate:

- United States
- Europe
- Australia

The services running at each of these locations are equivalent except that each location has it's own templates, images, storage and processing. Docmosis environments have access to one or more of these locations, allowing:

1. Optimizing performance by choosing a location which is closest to calling applications. This will reduce the latency of communications.
2. Geo-bounding of processing to a location as required.
3. Partitioning an application to provide different documents in different locations.

1.2.2. Environments

Docmosis Cloud offers the ability to create multiple Environments within an account. Each Environment will be available within each processing location, and have its own unique templates, images and access keys.

For example, an application may have the following 4 environments defined in the Docmosis Cloud account:

- Development Environment
- Testing Environment
- Staging Environment
- Production Environment

When calling the services from application code, or using the Docmosis Cloud Console, communication is always with a specific Docmosis location **AND** a specific environment. When calling the API, the API key used (accessKey) determines the environment.

1.2.3. Character Encoding

All data passed to the Cloud Services should be `UTF-8` encoded. This provides a great balance between flexibility and compatibility. If data is passed containing special characters, then is should use `UTF-8` encoding it, otherwise unexpected characters may appear in generated documents.

1.2.4. Fonts in Templates

The Cloud Service has most of the common and popular fonts.

Templates should only use fonts that are available on the Cloud Service. If fonts are used which are not installed, font substitution will occur and layout may be affected.

See our resources website (https://resources.docmosis.com) for a list of available fonts.

1.2.5. Dynamic and Stock Images

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

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

See *Image Data*

Sending Base 64 Dynamic Image Data on page *27* for more information.

Sourced from files uploaded to the Cloud Service: "Stock" images are images which are uploaded to the Cloud Service 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.

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

See *"Stock" Image Data* on page *28* for more information.

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

See Image Data from URLs on page *28* for more information.

1.2.6. 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 *Cloud Template Guide* for information about how to reference one template from another.

1.2.7. 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 documents can be produced that contain errors, helping users to locate errors 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 users can be assured that documents will never be delivered that have fundamental errors in processing.

In Docmosis, this is controlled via the `devMode` parameter of the `render` service. By default, production mode is enabled.

1.3. Troubleshooting

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

2. THE DEVELOPER API

2.1. Fundamentals

Docmosis Cloud Services is a REST-based API. More information about REST is available at: WikiPedia REST.

All calls to Docmosis are made using HTTPS POST requests. Code can call the API directly, use the Docmosis Cloud OpenAPI specification or use a third-party toolset to create requests. There is example code in various languages available on the Docmosis web site and in the OpenAPI specification.

Access and identification are provided by using an API Key "`accessKey`". Cloud Environments may have multiple access keys that can be rotated and disabled as required. An `accessKey` is required with each API call and can be a request parameter, or provided in the http header.

All API calls will use a location-based url. The available locations are:

Location	URL base
United States	https://us1.dws4.docmosis.com/api/
Europe	https://eu1.dws4.docmosis.com/api/
Australia	https://au1.dws4.docmosis.com/api/

Each location is independent of the others and Environments will have access to one or more locations. Each location and environment combination has its own templates/images and all processing is confined to that location to support data-sovereignty and privacy requirements as well as optimizing performance. Each environment has its own unique API keys, and these apply in all regions.

As an example, to upload a template to Docmosis in the US and then render it, use these two API urls:

https://us1.dws4.docmosis.com/api/**uploadTemplate**

https://us1.dws4.docmosis.com/api/**render**

Since each Docmosis location is independent of the others, the templates must be in the same location as being used to render documents.

2.2. Response Codes and Messages

For every call made to the Cloud Service, first check the response code to determine whether the call succeeded or failed. Once it is known whether the call succeeded or not, the response body can be processed for further information.

The service returns status codes as follows:

Status Code	Definition
200	Successful operation
400	The request is not valid or indicates a not-ready status.
500	A server error has occurred
404	Invalid URL (not found)

Other 4** and 5** response codes may also occur. Always confirm that a 200 status was received before continuing processing.

The Cloud Service 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 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 Cloud Service. For example, a status code 404 indicates a bad URL in which case the status code alone is enough to indicate the nature of the problem.

2.3. A Quick Tour of the API

The API has multiple areas:

1. Rendering – the most important service.
2. Template Management – uploading, downloading, listing templates.
3. Image Management – uploading, downloading, listing images.
4. Conversion service – pure document format conversion
5. Health/Monitoring

The following sections detail the endpoints and the related parameters.

2.4. The Render Service

The `render` service is the document production 'work-horse', and it is typically the main service called from applications, since template operations may also be carried out via the Cloud Console. Applications can call the Render service with data and instructions indicating: which template to use, the formats required, 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).

2.4.1. Service URL

```
/render
```

2.4.2. Request Headers

2.4.2.1. 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 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 to work with.

2.4.2.2. Access Key

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

2.4.3. 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:

```
{"accessKey":"XXXXX",
 "templateName":"template1.docx",
 "outputName":"result.pdf",
 "data":{"title":"Company Profile",
         "scope":"Initial Report"}}
```

The above shows 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 accessKey="XXXXX" templateName="template1.docx"
outputName="result.pdf">
  <data>
    <report title="Company Profile"
            scope="Initial Report"/>
  </data>
</render>
```

2.4.3.1. Mandatory Parameters

Parameter	Description
`accessKey`	An API key identifying the Environment . Can be found in the API section of the Cloud Console. Note the accessKey can be alternatively provided using a http header.
`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.
`templateName`	The name of the template to use. The template must have been uploaded previously with the template upload request or via the Cloud Console. 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.

2.4.3.2. Data Parameters

Parameter	Description	Default
`data`	The Data that will populate the document. This may be either 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", 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` mark-up must be at the beginning of the data value 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`

2.4.3.3. 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.	

Parameter	Description	Default
`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`
`mailGatewayName`	The custom mail gateway name to use. If using a "mailTo" `storeTo` directive then emails will be sent via this gateway. Note that the gateway must be preconfigured in the related cloud Account for the chosen Environment.	

2.4.3.4. Pdf Specific Parameters

Parameter	Description	Default
`pdfArchiveMode`	*Note: deprecated, 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 (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).	`PDF1.6`

Parameter	Description	Default
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 skips, for example, it is not valid to have Heading 1, Heading 3, and no Heading 2. • Text does not convey additional meaning with (direct) formatting.	false
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 doesn't exist on the server then the default watermark font will be used.	
pdfWatermarkRotation	If a watermark is specified, this rotation angle will be used. This must be a value between 0 and 360.	

Parameter	Description	Default
`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 = above 1-3 enabled, but page extraction disabled Requires that `pdfRestrictPassword` has been set.	`4`
`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.	

2.4.3.5. 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`
`devMode`	The render service can be used in development or production modes 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 its 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, `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, docx, 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.	

Parameter	Description	Default
`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`	A string to use to identify this job. This string will be returned in responses.	
`sourceId`	An ID to associate with this render. This could be a device id, a project code or any meaningful piece of data to associate with this render. This value can be reported later with associated monthly counts. Limited to 150 characters. The value is also returned in the response headers/body allowing the response to be linked to the original request.	
`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. Valid options are `stream, mailto, s3`. See section *2.4.4 Delivery Options* StoreTo Options for more details.	`stream`
`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`

Parameter	Description	Default
`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`
`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.	

2.4.4. Delivery Options

2.4.4.1. 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:bob@example.com,docx
```

This indicates a PDF document should be streamed back to the caller, and a DOCX document should be emailed to bob.

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>"` e.g. `"stream:pdf"`. 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`".
`mailto*`	Email* the document to the specified address. If "`mailGatewayName`" is specified, the preconfigured custom mail gateway will be used to send the email. Please contact our support team to enable and configure this option.
`S3`	Store the document in an external Amazon Web Services S3 location. This option is disabled by default. Please contact our support team to enable and configure this option.

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`
`S3`	`s3:my.amazon.bucket,mydocument1.pdf` `s3:my.amazon.bucket,mydocument1.zip:pdf,odt`

* 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 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. Response

2.4.5.1. 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).
`queue`	A JSON structure providing details of the render queue (can be used to help control application scaling): • `rejected` – Set to true if queue rejection has occurred (because queue is at capacity) • `availablePct` – Estimated % of queue available. • `delaySeconds` – A suggested back off time in seconds. This information is also provided via the headers described below.

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).
`queue`	As described above.

2.4.5.2. Response Headers

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

Header	Notes
`X-Docmosis-Server`	An identifier for this service “dws4”.
`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 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.
`X-Docmosis-Queue-Rejected`	Set to true if queue rejection has occurred (because queue is at capacity)
`X-Docmosis-Queue-Available-Pct`	Estimated % of queue available.
`X-Docmosis-Queue-Delay-Seconds`	A suggested back off time in seconds.

2.4.6. Image Data

2.4.6.1. Sending 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 a template image is using. The image data (i.e. 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....."
```

or (in RFC2397 format):

```
"data":{"pic1":"....."
```

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

Image data is typically large compared with textual information. Keep in mind the impact on network 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.

2.4.6.2. Image Data from URLs

Image data can also be dynamically sourced from URL references in the data.

As normal, a template would have marked up the image with a name that ties to the 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/image/Image103.png]"
```

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

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

and put it into the document dynamically.

Due to the potential impact on the Cloud platform, the use of URLs requires Docmosis support staff to white-list the URLs on an environment.

2.4.6.3. "Stock" Image Data

Where the same image is repeated in document production, such as logos or signatures, there are 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 Docmosis 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. upload the images in advance to the Docmosis environment - these are called "stock" images. Data can then reference these images, providing an efficient way to insert images into documents.

To use a stock image, it must be uploaded first into the related Environment. This can be done by logging into the Cloud Console and selecting Upload Image on the Images page. The

API allows applications to upload images programmatically – see *The Image Upload Service* in section *2.14* on page *47*.

Once an image has been uploaded to the cloud, it can be referenced it in data using a key that matches the template image, and a specially formatted value. For example, if the template has an image named `img_pic1` and `face1.jpg` has been uploaded into the Environment, the key is `pic1` and the value is `"[userImage:face1.jpg]"`. In JSON format, the data would look like this:

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

When an image is uploaded, a path-like structure may be used for organizing images. For example, and image uploaded with the name:

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

would be referenced like this:

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

2.4.7. Request Queueing

The render service automatically queues requests to facilitate spikes in traffic for a Docmosis Environment. If the queue is overrun for an Environment, the render will be rejected and the calling service must re-try. By monitoring the render queue, calling applications can:

- Avoid rejections
- Increase the scale document production over time

There are two ways to monitor the render queue:

1. Extract the information in the response to calls to the render service. The information includes the queue capacity information and the suggested back off time, as noted in the response body and response headers sections above.
2. Call the Get Render Queue service to monitor the queue information.

Calling applications can scale up document production over time by observing the queue information and increasing or reducing the amount of rendering being done. The Docmosis Cloud will scale in response to load but it takes some minutes for the capacity changes to take effect.

2.5. The Upload Template Service

This service uploads a template to be (later) rendered into documents. During upload, Docmosis analyses the template and will detect errors at this time.

Docmosis has a special capability to report errors in the template within rendered documents. This means that when a document is rendered the error AND its location in the template are visible. 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. This means that devMode can typically be ignored for a template, and used only at render-time.

Uploading of templates can also be done via the Cloud Console.

Templates can also be uploaded in bulk using *The Upload Template Batch Service* detailed in section *2.11*.

2.5.1. Service URL

```
/uploadTemplate
```

2.5.2. Request Headers

2.5.2.1. Content-Type

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

2.5.2.2. Access Key

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

2.5.3. Request Body Parameters

Field	Definition
`accessKey`	The Environment API key.
`templateFile`	The template file to upload.

Field	Definition
`templateName`	An optional, overriding name, which may include a path.
`templateDescription`	A description for the template.
`devMode`	If set to "y", "yes" or "true" the upload is run in developer mode, meaning that if errors are detected in the template the upload still succeeds. This allows later render requests to optionally display the errors in the template in a rendered document (using the render service's own devMode flag). In a production setting, a devMode of true for upload is perfectly valid since the devMode of the render service has the final say as to whether a document is produced. 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 this specifies the 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 will be NFC normalized (Unicode NFC normalization). The default is false.

2.5.4. Response Body

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`	The details of the template in JSON: (when succeeded = true) • `name` - the template file name • `lastModifiedMillisSinceEpoch` - last modified in milliseconds • `lastModifiedISO8601` - last modified yyyy-MMdd'T'HH:mm:ssZ • `sizeBytes` - the size in bytes • `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. Request Headers

2.6.2.1. Content-Type

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

2.6.2.2. Access Key

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

2.6.3. Request Body Parameters

Field	Definition
`accessKey`	The Environment API 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.

2.6.4. Response Body

On success (status=200), the body of the response will contain the binary stream for the template, or a zip file if multiple templates are specified.

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. Request Headers

2.7.2.1. Content-Type

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

2.7.2.2. Access Key

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

2.7.3. Request Body Parameters

Field	Definition
`accessKey`	The Environment API key.
`templateName`	The name of the template.
`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.

2.7.4. 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.

Field	Definition
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 • 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 currently in a template (such as creating dynamic data forms etc).

2.8.1. Service URL

```
/getTemplateStructure
```

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) identifies the Environment and can be specified as a request header or a body parameter (see below).

2.8.3. Request Body Parameters

Field	Definition
accessKey	The Environment API key.
templateName	The name of the template.
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

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

Field	Definition

	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_>>
	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"]
	}
	]
	}
	]
	}
	]

2.9. The List Templates Service

List templates lists the templates available in the Environment.

2.9.1. Service URL

```
/listTemplates
```

2.9.2. Request Headers

2.9.2.1. Content-Type

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

2.9.2.2. Access Key

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

2.9.3. Request Body Parameters

If only the API key is specified, all templates are listed.

Field	Definition
`accessKey`	The Environment API key.
`includeDetail`	Include extra detail about parameters. Default=true.
`folder`	Limit processing to the given folder. Optional. The returned names are always relative to the folder.
`includeSubFolders`	Include the contents of sub folders (ie work recursively). (folder names are always included at the current level being listed regardless of this setting). Default=true
`paging`	Whether or not to return results in pages. Default=false. If true, pages of 1000 records are returned.

Field	Definition
`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.
`pageSize`	Specify the size of pages. Defaults to 1000 but can be reduced.

2.9.4. Response Body

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 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 • `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.`

Field	Definition
`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.
`pageSize`	The size of pages when paging is active. Default is 1000 (which is also the maximum supported). Since the list returns folder names as items, the folders themselves count in the page size.

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. Request Headers

2.10.2.1. Content-Type

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

2.10.2.2. Access Key

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

2.10.3. Request Body Parameters

Field	Definition
`accessKey`	The Environment API key.
`templateName`	The name of the template. This parameter can be specified multiple times to delete multiple files.

2.10.4. Response Body

The 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 Upload Template Batch Service

This service allows templates to be uploaded in a Zip file.

The zip file may be a simple list of templates or may have templates organized into folders. The folder structure of the zip file will be replicated in the cloud Environment. The upload is additive, meaning the templates will be added to those already in the Environment and will overwrite any existing template in the same location with the same name.

This service, unlike most other services, runs asynchronously. A “job” is launched to perform the processing which could potentially take significant time. The caller can then check on the status of the job or cancel it using the appropriate end points described below.

Jobs are assigned a `userJobId` either by the caller or automatically. This id can be used to query the status of the job or to cancel a running job.

Uploading of templates can also be done via the Cloud Console.

In terms of concurrent batch uploads, the following rules apply to each environment:

- Uploads to different Docmosis Regions may occur concurrently (since the templates are independent) as long as there is not another job running with the same userJobId (if specified) for the Environment.
- Uploads to the same region may occur concurrently provided:

- The userJobId (if provided) is unique against any other currently running job (that is running in any region)
- There is no overlap to where the templates are being stored (ie the explicit or implied intoFolder).

Limitations to the size of the upload and the number of templates apply. Contact Docmosis support to upload zip files larger that 50Mb or with more than 1000 templates.

The zip is assessed in step one and the process will be terminated before any templates are affected if a problem with the zip is detected. The following tests are included: zip too large or with too many entries, any entries with invalid paths (references to drives or parent folders), any entries that look like scripts, executables, archives, etc.

2.11.1. Service URL

```
/uploadTemplateBatch
```

2.11.2. Request Headers

2.11.2.1. Content-Type

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

2.11.2.2. Access Key

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

2.11.3. Request Body Parameters

Field	Definition
accessKey	The Environment API key.
templateZip	The template files to upload. This can include folders which will be created in the cloud Environment. The zip is discarded at the end of the process, only the uploaded templates persist.

Field	Definition
`userJobId`	An identifier for the batch upload job which can be used to check the status of the job or to cancel the job. If no id is provided a unique one will be generated and returned in the response message. Maximum of 40 characters. The jobId can be reused. When checking on the status of a job, only the latest version of the job with the id is considered.
`intoFolder`	The path to the folder to upload the templates to. If no path is provided the templates will be uploaded to the 'root' directory of the cloud Environment.
`devMode`	If set to "y", "yes" or "true" the upload is run in developer mode, meaning each template will be uploaded in Developer mode (errors allowed) and the job will complete if possible. 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. In a batch of templates, this setting will only affect the last template processed because a failure processing the template will abort the batch job. Defaults to "false".
`fieldDelimPrefix`	If using plain text mark-up in the templates this specifies the 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 names will be NFC normalized (Unicode NFC normalization). The default is false.

2.11.4. Response Body

The response is an indicator of success or failure plus any further helpful information. When the upload is successful, job details are returned under the `jobStatus` 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.
`jobStatus`	The status of the job in JSON: (when succeeded = true) • `userJobId` – An identifier for the batch upload job which can be used to check the status of the job or to cancel the job. If no id was provided in the request a generated one is returned. • `isEnded` – "true" or "false". • `status` – The current status of the job. • `type` – The type of job being run. • `processingMsg` – A message about the template processing. • `startedTime` – Started time in milliseconds (Epoch time). • `finishedTime` – Finished time in milliseconds (Epoch time). 0 if job is ongoing. • `duration` – Duration of job in milliseconds • `pctComplete` – Percentage complete, 0 – 100.

2.12. The Upload Template Batch Status Service

This service reports on a currently or previously running template batch upload job. In the response, the “jobStatus” payload contains multiple items to establish the status of the job. Perhaps the most important data items for managing the job are “isEnded” and the “errorsDetected” field of the “jobResult” field.

When the job is completed, successfully or not, the jobStatus will contain the jobResult which has the details of the level of success of the entire batch.

2.12.1. Service URLs

```
/uploadTemplateBatchStatus
```

2.12.2. Request Headers

2.12.2.1. Content-Type

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

2.12.2.2. Access Key

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

2.12.3. Request Body Parameters

Field	Definition
`accessKey`	The Environment API key.
`userJobId`	The identifier for the batch upload job.

2.12.4. Response Body

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.

Field	Definition
jobStatus	The status of the job in JSON: (when succeeded = true) • userJobId – An identifier for the batch upload job which can be used to check the status of the job or to cancel the job. If no id was provided in the request a generated one is returned. • isEnded – "true" or "false". • status – The current status of the job. • type – The type of job being run. • processingMsg – A message about the template processing. • startedTime – Started time in milliseconds (Epoch time). • finishedTime – Finished time in milliseconds (Epoch time). 0 if job is ongoing. • duration – Duration of job in milliseconds (Epoch time). • pctComplete – Percentage complete, 0 – 100. • jobResult – The result of the job in JSON (when isEnded = true) • errorsDetected – true if any of the uploaded templates have errors • uploadedIntoFolder – the provided folder path to upload the templates to. • devMode – the dev mode setting used when it was uploaded. • uploadedTotalCount – total number of templates uploaded. • processedWithErrorsCount – number of uploaded templates processed with errors. • processedWithoutErrorsCount – number of uploaded templates processed without errors.

2.13. The Upload Template Batch Cancel Service

This service allows a currently running template upload batch job to be cancelled. This service will return failure only if there was an error cancelling the job. If the job is already completed (so the cancel has no effect), this will return success and the message will indicate that it was already finished.

2.13.1. Service URLs

```
/uploadTemplateBatchCancel
```

2.13.2. Request Headers

2.13.2.1. Content-Type

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

2.13.2.2. Access Key

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

2.13.3. Request Body Parameters

Field	Definition
`accessKey`	The Environment API key.
`userJobId`	The identifier for the batch upload job.

2.13.4. Response Body

Field	Definition
`succeeded`	"true" or "false" If the job has already completed, “true” will be returned. “false” is only returned if there was an error attempting to cancel a running job or the userJobId is invalid.
`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 Image Upload Service

The Image Upload service works the same as the template upload service, but is significantly simpler since there are fewer options to set.

Image uploading is used to create "stock" images that data can reference when generating documents and Docmosis will take care of placing them into a document. This saves

bandwidth and time when rendering the same images frequently. An alternative to uploading "stock" images is to place the images into the document and selectively filter them out using conditional sections.

2.14.1. Service URL

```
/uploadImage
```

2.14.2. Request Headers

2.14.2.1. Content-Type

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

2.14.2.2. Access Key

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

2.14.3. Request Body Parameters

Field	Definition
`accessKey`	The Environment API key.
`imageFile`	The file stream of the image.
`imageName`	An overriding name for the image which may include a path (e.g. `project1/stock/logo1.jpg`).
`imageDescription`	A short description for the image.
`normalizeImageName`	If set to "y", "yes" or "true" the image name given with be NFC normalized (Unicode NFC normalization). The default is false.

2.14.4. Response Body

The response is an indicator of success or failure plus any further helpful information. When the upload is successful, details are returned under the `imageDetails` 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.
`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 • `md5` - the md5 hash code for the image

2.15. The List Images Service

The List Images service lists the images available in the Environment.

2.15.1. Service URL

```
/listImages
```

2.15.2. Request Headers

2.15.2.1. Content-Type

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

2.15.2.2. Access Key

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

2.15.3. Request Body Parameters

If only the API key is specified, all images are listed.

Field	Definition
accessKey	The Environment API key.
folder	Limit processing to the given folder. Optional. The returned names are always relative to the folder.
includeSubFolders	Include the contents of sub folders (ie work recursively). (folder names are always included at the current level being listed regardless of this setting). Default=true

2.15.4. Response Body

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 • md5 - the md5 hash code for the image

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

2.16. The Delete Image Service

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

2.16.1. Service URL

```
/deleteImage
```

2.16.2. Request Headers

2.16.2.1. Content-Type

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

2.16.2.2. Access Key

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

2.16.3. Request Body Parameters

Field	Definition
accessKey	The Environment API key.
imageName	The name of the image. This parameter can be specified multiple times to delete multiple files.

2.16.4. Response Body

The 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.

2.17. 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).

2.17.1. Service URL

```
/getImage
```

2.17.2. Request Headers

2.17.2.1. Content-Type

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

2.17.2.2. Access Key

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

2.17.3. Request Body Parameters

Field	Definition
`accessKey`	The Environment API key.

Field	Definition
imageName	The name of the image. This parameter can be specified multiple times to download multiple images (up to 100) in a zip file.

2.17.4. Response Body

On success (status=200), the body of the response will contain the binary stream for the image, or a zip file if multiple images are specified..

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.

2.18. 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.

2.18.1. Service URL

```
/convert
```

2.18.2. Request Headers

2.18.2.1. Content-Type

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

2.18.2.2. Access Key

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

2.18.3. Request Body Parameters

Field	Definition
accessKey	The Environment API 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.

2.18.4. Response Body

The 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 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 an application.

2.19.1. Service URL

```
/getRenderTags
```

2.19.2. Request Headers

2.19.2.1. Content-Type

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

2.19.2.2. Access Key

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

2.19.3. Request Body Parameters

Field	Definition
`accessKey`	The Environment API key.
`Tags`	The tags to query. This can be a single tag or a list of tags separated by the ; (semicolon) 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 specified year and month are included. In other words, 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.

2.19.4. Response Body

The service responds with a JSON structure as follows:

Field	Definition
`succeeded`	"true" or "false".

Field	Definition
`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
	  ▪ `countDocuments` – the number of documents rendered against this tag in the year and month
	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 return 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.

2.20. 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 which 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.

2.20.1. Service URL

```
/getSampleData
```

2.20.2. Request Headers

2.20.2.1. Content-Type

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

2.20.2.2. Access Key

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

2.20.3. Request Body Parameters

Field	Definition
`accessKey`	The Environment API 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.
`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.

2.20.4. 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.

2.21. The Get Render Queue Service

The get render queue service returns info about the current status of the render queue for the specified Environment. The response information can be used to manage scaling of the application calling the Docmosis render service.

See section 2.4.7 Request Queueing for information about how to use the information provided by this service.

2.21.1. Service URL

```
/getRenderQueue
```

2.21.2. Request Headers

2.21.2.1. Content-Type

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

2.21.2.2. Access Key

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

2.21.3. Request Body Parameters

Field	Definition
`accessKey`	The Environment API key.

2.21.4. 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.
`queue`	A JSON structure providing details of the render queue: • `availablePct` – Indicates % of queue available for current server. • `delaySeconds` – A suggested back off time in seconds.

An example response is:

```
{
  "succeeded": true,
```

```
  "queue": {
    "availablePct": 100,
    "delaySeconds": 0
  }
}
```

2.21.5. Response Headers

The following headers are also returned to align with the render endpoint.

Header	Notes
X-Docmosis-Queue-Available-Pct	Indicates % of queue available for current server.
X-Docmosis-Queue-Delay-Seconds	A suggested back off time in seconds.

2.22. The Ping Service

The ping service provides a direct check that the Docmosis Cloud services are online and there is at least one Docmosis server listening. This is useful for diagnosing reachability issues. For automated monitoring purposes see the Environment Ready service.

2.22.1. Service URL

```
/ping
```

2.22.2. Request Headers

2.22.2.1. Content-Type

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

2.22.3. Request Body Parameters

None.

2.22.4. Response Codes

200 – successfully pinged

2.22.5. Response Body

There is no response body returned.

2.23. The Environment Ready Service

The environment ready service provides the ability to check whether an environment is ready to service document requests. This service is suitable for linking to automated monitoring systems for important Docmosis Environments.

This service confirms whether the Environment is active and within quota limits. Note: Docmosis Environments may be over quota and still operational (ie hard limit not applied), in which case this service will indicate the environment is ready, even though over quota.

Consider creating an API Key in the Environment specifically for use with this service.

2.23.1. Service URL

```
/environment/ready
```

2.23.2. Request Headers

2.23.2.1. Content-Type

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

2.23.2.2. Access Key

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

2.23.3. Request Body Parameters

Field	Definition
`accessKey`	The Environment API key.

2.23.4. Response Codes

This service is intended for monitoring / confirmation of ready state. The return codes are:

200 – environment is ready

400 – environment is not ready

2.23.5. 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 where the environment is not ready, this message will indicate why.

2.24. The Environment Summary Service

The environment summary service returns details about an environment including plan information, status and quotas. It is for information purposes and not intended for monitoring.

Note: this service returns a 200 response even if the Environment is not ready which is different from the "ready" service endpoint.

2.24.1. Service URL

```
/environment/summary
```

2.24.2. Request Headers

2.24.2.1. Content-Type

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

2.24.2.2. Access Key

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

2.24.3. Request Body Parameters

Field	Definition
accessKey	The Environment API key.

2.24.4. Response Body

The service responds with a JSON structure as follows:

Field	Definition
succeeded	"true" or "false".
shortMsg	A short message about the ready-status of the environment
accountEnvironmentSummary	A JSON structure providing details of the environment • ready – whether the environment is ready to create documents • accountEnvDetails: ▪ isActivated is the environment activated ▪ isDisabled has the environment been disabled ▪ isDeleted has the environment been deleted ▪ name the environment name ▪ description the environment description • pageQuota: ▪ used – pages rendered this month ▪ quota – monthly page quota ▪ pctUsed – quota used as a percentage ▪ pctUsedStr – quota used as a string ▪ isHardLimited – whether the environment is blocked because of over-quota • plan: ▪ name – the name of the plan.

An example response is:

```
"accountEnvironmentSummary":{
  "ready":"true",
  "accountEnvDetails":{
    "isActivated":"true",
    "isDisabled":"false"
```

```
      "isDeleted":"false"
      "name":"Development"
   },
   "pageQuota":{
      "used":"467",
      "quota":"230",
      "pctUsed":"203",
      "pctUsedStr":"203%",
      "isHardLimited":"false"
   },
   "plan":{
      "name":"Free Trial"
   }
}
```

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