|  |
| --- |
| Docx html Editor |
| **v0.0.1 Proof of Concept User Manual** |



|  |
| --- |
|  |

|  |
| --- |
|  |

Contents

[SOLUTION OVERVIEW 3](#_Toc371058483)

[CONTENTS 4](#_Toc371058484)

[INSTALLATION 4](#_Toc371058485)

[API 4](#_Toc371058486)

[INTEGRATION 5](#_Toc371058487)

[JAX-RS 5](#_Toc371058488)

[LOGGING 5](#_Toc371058489)

[DEPLOYMENT NOTES 6](#_Toc371058490)

[KNOWN ISSUES/LIMITATIONS 6](#_Toc371058491)

[ROADMAP 6](#_Toc371058492)

[SUPPORT 6](#_Toc371058493)

[Appendix 1 – LOAD REQUEST/RESPONSE 7](#_Toc371058494)

[Appendix 2 – SAVE REQUEST/RESPONSE 8](#_Toc371058495)

Docx Html Editor

This document explains how to use the docx Html Editor

docx html Editor is a ***proof of concept*** of a new approach to editing docx documents in a web browser.

The objectives are two fold:

* to be able to utilize/integrate with customer’s preferred HTML editor to provide the user interface
* after editing, to save a docx which preserves “long tail” content (eg charts, equations) which the editor was unable to manipulate

In this proof of concept, the editor used is CKEditor 3.6.6.1

# SOLUTION OVERVIEW

The docx file format contains a “long-tail” of features. The web editor is designed to enable in browser editing of:

* Paragraphs of text (including basic formatting such a bold, italics, font size)
* Images (insertion thereof)
* Simple tables

The solution handles most long tail features in one of two ways:

1. By showing it in the XHTML as a “non-editable content” icon, then reinstating wholly or substantially the original feature when the XHTML is converted back to docx format, or
2. stripping/filtering/resolving that feature/content during the conversion to XHTML (with the result that this content will be missing from the docx created as a result of the editing process).

By way of illustration, the following table sets out how various features are treated:

|  |  |  |
| --- | --- | --- |
| Feature | Stripped/ filtered | Reinstated |
| Shape |  | **✓** |
| Chart |  | **✓** |
| WordArt |  | **✓** |
| ActiveX Control: Option button |  | **✓** |
| Field | 🗶 |  |
| Signature line |  | **✓** |
| Content control | *some* | ***some*** |

Basic workflow:

1. Submit (POST) a docx to the server (specified URL)
2. Server-side, convert the docx file to (X)HTML
3. Stream the (X)HTML to the client device invoking web editor (ie CKEditor), with a WYSIWYG-like view of the (X)HTML, ready for editing
4. Once the user has made their edits, they can click a submit button, whereupon the (X)HTML document is posted to the server
5. The server converts the XHTML back to docx format
6. The server streams the docx back to client

# INSTALLATION

Install the WAR file in your application server following the procedure specified by its vendor.

After that, visit upload.html, for example:

<http://localhost:[port]/docx-html-editor-0.0.1/upload.html>

To test things are working properly, choose a docx file. You should see it load in CKEditor.

# API

This solution exposes an API with 2 basic functions:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ***Name*** | ***Input*** | | ***Output*** | ***Comments*** |
| ***Type*** | ***Endpoint*** |
| 1 | Load | POST docx using encoding type multipart/form-data, in form data param named "docxfile" | services/load | Web-editor ready XHTML | Creates session.  You can customise the XHTML which is served, by modifying docx2xhtml\_CKEditor2013.xslt |
| 2 | Save | POST XHTML using application/x-www-form-urlencoded in form param editorOutput | services/save | Docx stream | Be sure to include session info in your request |

Supporting the load operation are various additional endpoints:

|  |  |
| --- | --- |
| ***Endpoint*** | ***Description*** |
| services/css | Get the CSS which represents the styles in the docx |
| services/image | Get an image present in the docx; Upload a new image to the server |
| placeholders | Retrieve image representing “long tail” content |
| services/images | Browse images on the server |
| ckeditor | Get a CKEditor resource |

# INTEGRATION

To integrate in your application, your application can intermediate between the client and this web application to control:

* Which docx file is to be loaded/edited.  
  For example, your workflow determines this, or user browses your content management system
* What is done with the docx stream on services/save

## Session tracking

Session tracking is required in order for this web application to function correctly. Symptoms of missing session info include:

* Images missing
* No styles in editor
* Unable to save the editor HTML output as a docx

Our approach to session tracking is the widely used javax.servlet.http.HttpSession interface. This approach supports session tracking by either:

* Cookie (typically, ‘JSESSIONID’), or
* URL rewriting

In some integration scenarios, URL rewriting will be preferred. An application server may be able to be configured to force this. For example, with Tomcat, you can set attribute cookies="false" in conf/context.xml.

The appendices (1 and 2) show request/response pairs for the case where session tracking is via a cookie called JSESSIONID.

## Load Endpoint

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ***Name*** | ***Input*** | | ***Output*** | ***Comments*** |
| ***Type*** | ***Endpoint*** |
| 1 | Load | POST docx using encoding type multipart/form-data, in form data param named "docxfile" | services/load | Web-editor ready XHTML | Creates session.  You can customise the XHTML which is served, by modifying docx2xhtml\_CKEditor2013.xslt |

**adv=bare**: There is an optional parameter ‘adv’, which, if set to ‘bare’, will return a plain XHTML rendition of the docx (ie as opposed to an XHTML file which includes ckeditor.js and CKEDITOR.config). If you choose to use this, you will need to configure/invoke the web editor yourself. You will also need to configure it to return suitable well-formed XHTML to the save endpoint. See docx2xhtml\_CKEditor2013.xslt for an example of a working CKEditor configuration.

# JAX-RS

Docx4j Web Editor is a JAX RS 2.0 application.

It uses the Jersey implementation of JAX RS:

<dependency>

<groupId>org.glassfish.jersey.core</groupId>

<artifactId>jersey-server</artifactId>

<version>2.1</version>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.containers</groupId>

<artifactId>jersey-container-servlet-core</artifactId>

<version>2.1</version>

</dependency>

# LOGGING

Docx4j 3.0 logs using slf4j. Unfortunately, Jersey uses java.util.logging (JUL) for logging. You can vote to change this:

* <https://java.net/jira/browse/JERSEY-369>
* <https://java.net/jira/browse/JERSEY-1275>

For consistency with Jersey, docx4j Web Editor also uses JUL. For details on how to configure JUL, consult your application server/servlet container documentation.

To configure docx4j’s slf4j logging, you should add a binding jar (eg slf4j-log4j-1.7.5.jar) where your classloader can find it.

slf4j’s jul-to-slf4j module includes SLF4JBridgeHandler, which routes all incoming JUL records to the SLF4j API. However, this has a performance impact, so docx4j Web Editor doesn’t use it. For further information, please see:

* <http://www.slf4j.org/legacy.html#jul-to-slf4j>
* <http://blog.cn-consult.dk/2009/03/bridging-javautillogging-to-slf4j.html>

# DEPLOYMENT NOTES

[various app servers / servlet containers to be completed]

# KNOWN ISSUES/LIMITATIONS

[to be completed]

# Appendix 1 – LOAD REQUEST/RESPONSE example

## Request

**POST** /docx-html-editor-1.0.0-SNAPSHOT/**services/upload** HTTP/1.1

Host: localhost:8086

Connection: keep-alive

Content-Length: 13898

Cache-Control: no-cache

Pragma: no-cache

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

Origin: http://localhost:8086

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/30.0.1599.101 Safari/537.36

**Content-Type: multipart/form-data; boundary=----WebKitFormBoundarypkz1C4HemKRtrs7v**

Referer: http://localhost:8086/docx4j-web-editor-1.0.0-SNAPSHOT/upload.html

Accept-Encoding: gzip,deflate,sdch

Accept-Language: en-GB,en-US;q=0.8,en;q=0.6

------WebKitFormBoundarypkz1C4HemKRtrs7v

Content-Disposition: form-data; **name="docxfile";** filename="Hello world.docx"

Content-Type: application/vnd.openxmlformats-officedocument.wordprocessingml.document

## Response

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

**Set-Cookie: JSESSIONID=7CDDB5B0FD73F5BCC655FCD2AE949006**; Path=/docx4j-web-editor-1.0.0-SNAPSHOT/

Content-Type: text/html

Content-Length: 5378

Date: Thu, 31 Oct 2013 22:31:52 GMT

# Appendix 2 – SAVE REQUEST/RESPONSE example

## Request

**POST** /docx-html-editor-1.0.0-SNAPSHOT/**services/save** HTTP/1.1

Host: localhost:8086

Connection: keep-alive

Content-Length: 226

Cache-Control: no-cache

Pragma: no-cache

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

Origin: http://localhost:8086

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/30.0.1599.101 Safari/537.36

**Content-Type: application/x-www-form-urlencoded**

Referer: http://localhost:8086/docx4j-web-editor-1.0.0-SNAPSHOT/services/upload

Accept-Encoding: gzip,deflate,sdch

Accept-Language: en-GB,en-US;q=0.8,en;q=0.6

**Cookie: JSESSIONID=7CDDB5B0FD73F5BCC655FCD2AE949006**

**editorOutput**=%3Cp+class%3D%22Normal+DocDefaults+%22+id%3D%22DOCX4J\_EDITOR%3Ap148%22%3E%3Cspan+class%3D%22DefaultParagraphFont+%22+style%3D%22font-family%3A+Calibri%3B%22%3EHello%2C+world%21%3C%2Fspan%3E%3C%2Fp%3E&submit=Submit

## Response

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Disposition: attachment; filename=result.docx

Content-Type: application/vnd.openxmlformats-officedocument.wordprocessingml.document

Transfer-Encoding: chunked

Date: Thu, 31 Oct 2013 22:35:51 GMT