|  |
| --- |
|  |
|  |
| **DocXAPI – Java based MSWord templating Integration Guide** |
| **Version 1.0 24-March-2022** |
| Document ID: -ADDS-007 |

**REVISION HISTORY**

| Version | Author | Date | Summary of changes |
| --- | --- | --- | --- |
| 0.1 | Linus Neo | 24 March 2022 | Initial Release |

TABLE OF CONTENT

[1. Getting Started 1](#_Toc98959045)

[1.1. Purpose 1](#_Toc98959046)

[1.2. Java and Maven Configuration 1](#_Toc98959047)

[2. Application Programming Interfaces (API) 2](#_Toc98959048)

[2.1. Core MSWord templating engine (DocXAPI) 2](#_Toc98959049)

[2.2. Notice (Letter) Generating Microservice - NoticeAPI 9](#_Toc98959050)

[3. Use of Table in Document Template 13](#_Toc98959051)

[4. Profiling of API Response Time Performance 13](#_Toc98959052)

# Getting Started

## Purpose

This document serves as a developer guide on the use of the Docx4J library to develop a high-level API for the following use cases:

1. Creating simple MSWord document containing textual and image contents.
2. Performing mail merge based on field definition in a MSWord template file, including merging of headers and footers defined in separate word document

The API is implemented as 2 separate projects. The first project DocXAPI contains the lower level of abstraction and are not coupled with the any Microservice Framework. The second project NoticeAPI consist of integration with Spring Boot as well as involve database setup and relevant validation checks to ensure robustness.

It is not the intend of this guide described all the class methods as some of these are internal helpers or routine to modularize the API.

## Java and Maven Configuration

This API requires Java version 11 and above and of this writing on 1 Mar 2022, the docx4j library is at version 11.3.2. Refer to the Maven POM file for the list of dependencies.

# Application Programming Interfaces (API)

## Core MSWord templating engine (DocXAPI)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Sno | **API** | **Description** | |
| TextField | | Class which contains information pertaining to textual fields or content | |
| 1 | setText | Set the textual value | |
| 2 | getText | Get the textual value | |
| 3 | setFormat | Set the format fields with the supported attributes (eg bold, underline, italics and selected color) when the text is rendered onto the paragraph in the MSWord document. It is seldom invoked directly but set via the helper formatting APIs such as   * bold * italics * underline * textual color (red, blue) * highlight (yellow, green) * encrypt[[1]](#footnote-1) * is<attribute> to check if the above attributes had been set | |
| 4 | getFormal | Get the formatting attribute pertaining the textual content. | |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Sno | **API** | **Description** | |
| ImageField | | Class which contains information pertaining to image fields or content | |
| 1 | setImage | Set the byte array within the class via the polymorphic parameter:   * byte[] * base64 encoded image string | |
| 2 | getImage | Get the image and return as byte array (ie byte []) | |
| 3 | setSize | By default, all image will be rendered based on their original size.  If the size (in cm) has been set via this API, the image will be scaled by the size variable. | |
| 4 | getSize | Get the size to be rendered | |
| 5 | setProperties | For future enhancement, attributes related to the image is stored in a java hashmap. This allows for extension of other attributes based on key value pair by this API. | |
| 6 | getProperties | Get the image attribute based on the attribute name as the key to the hashmap. | |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Sno | **API** | **Description** | |
| QrCodeField | | Class which contains information pertaining to textual data that are to be used to create an image containing the QR code. | |
| 1 | QrCodeField | The sole constructor accepts the string valued data to generate the QR code, along with the size (in what unit ?) in width and height parameters. Both parameters are stored in a HashMap to allow for additional attributes in the future to be added easily. | |
| 2 | setQrCodeString | This API allows the string value to be overridden if already set in the constructor or when the parameterized constructor was not called during object instance creation. | |
|  | getQrCodeString | Get the string valued data which will be used to generate the QR code. | |
| 3 | setWidth | Set the Width attribute of the eventually generated QR code image. The default width is defined in the Constant class. | |
| 4 | getWidth | Get the intended width for QR code image | |
| 5 | setHeight | Set the Width attribute of the eventually generated QR code image. The default height is defined in the Constant class. | |
| 6 | getHeight | Get the intended height for the QR code | |
| 7 | setProperties | Similar to the imageField API in purpose | |
| 8 | getProperties | Similar to the imageField API in purpose | |
| 9 | generateQrCode | This is a wrapper API that calls the DocXAPI.createQrCodeImage API to generate the actual QR image. It accepts a MatrixToImageConfig that could alter the background and foreground color of the generated QR code image. | |
|  |  |  | |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Sno | **API** | **Description** | |
| RecipientAddress | | Class which recipient related information as well as provide linkage to notice reference and date. This class uses Lombok @Data annotation to generate the setter and getter method. It contains data field for   * Recipient Name * Transaction Id\Date * NCA address fields * Notice Number (where applicable) and Version | |
| 1 | format | Addresses are stored using the National Coded Address (NCA) format. This API serves to derive a 3 liners address that is consistent with the Singapore postal standard. | |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Sno | **API** | **Description** | |
| DocXAPI | | The heart and soul of the MSWord document generating API. It contains a few inner classes, namely   * HeaderFooter * NoticeSummary * Mergefields   The API in these inner class should not be called directly.  It also contains a special compile time variable to etch the binary version to the compilation date (retrieval via the apiVersionDate API) | |
| 1 | loadTemplate | This API will load a MSWord template. **It must be the first API to invoked** in order to initialize the rest of the structure required by the under docx4j library. In a nutshell, an empty word document must exist, even if there is no defined static contents to be generated. Underlying it set the default fonts to Ariel. | |
| 2 | loadImage | This API takes an image path and load the image into a byte array (ie byte[]) | |
| 3 | setHeaderFooter | This API creates the header or footer programmatically based on 3 parameters, namely type (“HEADER” or “FOOTER”, a HdrFtrRef and a list containing a list of object. The outer list is to segregate the list of objects and render them as paragraphs on separate lines within the header or footer. The inner list contains either byte[] for an ImageField instance to be created or String for an TextField instance to be created on the paragraph. The HdrFtrRef is a docx4j data type to denote how the header/footer is placed ie on the first page only, even pages only or default (all other pages).  Internally, the list of headers or footers are stored in a hashmap (named as headerFooter) where the key are denoted in the convention:  <type>-<hdrFtrRef>.  Internally, it does not render the defined headers and footers into the loaded template. The rendering is done on the loaded template when the mailMerge API is invoked. | |
|  |  |  | |
|  | clearHeaderFooter | Clears the header and footer in the loaded template | |
|  | createStdHeader | Accepts a list of string, TextField, byte [] or ImageField object each representing a line on the header. The second parameter is the type of header to create (i.e. only on first page, even page only or default) and the third justifies the paragraph. The API renders the paragraph onto the header of the loaded template. | |
|  | createStdFooter | Similar to createStdHeader, but for footer. It adds a right justified page number at the end of the footer. | |
|  | createHeaderObject | Called createJustifiedParagraph to create paragraphs containing left and right justified content for the header. The second parameter denotes the type of headers (First page only, even page only or default). | |
|  | createFooterObject | Similar to createHeaderObject but for Footer | |
| 4 | serializeHeaderFooter | Seriaize the headerFooter HashMap to Json | |
| 5 | deserializeHeaderFooter | Reconstruct the headerFoooter Hashmap by deserializing from the Json created by the serialiizeHeaderFooter API | |
| 6 | breakPage | Creates a page break in the MS word document | |
| 7 | setSingleLineSpacing | Creates a paragraph property object so that the line spacing is one line. | |
| 8 | justifyPargaph | Justify the paragraph by setting its property object to one of the JcEnumeration values, namely LEFT, RIGHT or CENTER | |
| 9 | createTextParagraph | Accepts either a string or a TextField object as the first parameter in order to return a paragraph with the text content justified based on the second parameter. API returns a paragraph object. It does not render the paragraph in the template. | |
|  | createDrawing | Low level API which convert a byte array containing an image into a “BinaryPart” that is then attached to the “Part” of the document that is passed in as the second parameter. The third parameter determine the size of the image in cm if it is not 0. 0 value means no scaling would be done. | |
|  | createImageParagraph | Creates a paragraph that contains a drawing that is created by the createDrawing API. This paragraph is added to the document part that is passed as the second parameter, the first parameter being the image byte array. The third is the image size and the fourth parameter specifies the justification. API returns a paragraph object. It does not render the paragraph in the template | |
|  | createSimpelParagraph | Accepts a list of objects that are either of type string, TextField, byte [] and ImageField as the first parameter, with the document part as the second parameter and the paragraph justification as the third parameter. This API uses createTextParagraph for string and TextField object and createImageParagraph for byte [] and ImageField. The resultant paragraph is rendered to the template part passed in as the second parameter. | |
|  | createJustifiedParagraph | Similar to createSimpleParagraph, except that the first parameter is a list of list of Object. The inner list is expected to have 2 elements, one representing the left justified content and the other representing the right justified content. The outer list denotes the separate line (aka paragraph) to be rendered in the template. Internally, due to limitation in the docx4j FOP, the justification is implemented based on tables instead of defining tabs stop. | |
|  | createSimpleSignOffParagraph | Accept the sign off protocol (ie Sincerely or Faithfully) as the first parameter, with the second parameter being an image byte array to denote the signature. The third parameter is an array of 3 elements to denote the 3 lines related to the signee’s designation. Internally, it is dependent on createSimpleParagraph to render the signatory at the bottom of the document. | |
|  | createSimpleField, createComplexField | Creates simple or complex fields (e.g. page numbering and number of pages etc.). | |
|  | createPageNumberParagraph | Calls createSimpleField to create a paragraph containing the current page number. Due to unknown bug, the ‘of {numpage}’ is currently not working. | |
|  | createTextField | Return a definition of textual based content of type TextField | |
|  | getTextFieldProperties | Get the paragraph properties pertaining to the TextField | |
|  | createImageField | Return a definition of the image based content of type ImageField | |
|  | createQrCodeField | Returns a definition of the eventual QR code of type QrCodeField. This API is not used directly within the DocXAPI library. It is a convenient ‘method’, specifically for the NoticeAPI microservice. | |
|  | createQrCodeImage | Returns a byte [] of the generated qr code image based on the string that is passed as the first parameter. The second and third parameter defines the image width and height. | |
|  | qrConfigureColor | Pass in the background and foreground color to setup the configuration to determine the color of the generated qrCode via API call to createQrCodeImge. | |
|  | setRecipient | Set the object of recipientAddress class. Internally, it calls the MergeField.setField API directly so that the following fields are captured as merge field to facilitate sending the information for outsource printing:   * recipeinname * transactionid * transactiondate * addressline1 * addressline2 * addressine3 | |
|  | setTextMergeFields | Set the textual merge fields passing in a Hashmap of TextField with the field name as the key, in preparation for mail merging. | |
|  | setImageMergeFields | Set image fields passing in a Hashmap of ImageField with the field name as the key, in preparation for mail merging. | |
|  | setTableMergeFields | Set the rows of data contained in a Hashmap where the value is a string and the key is the name of the column name, in preparation for mail merging. It accepts an arbitrary table name to group the rows so that the columnar content would be rendered on the same table. | |
|  | mailMerge | Mail merge the text, image and table merge into the loaded template. | |
|  | findAndReplace | Replace a target string in the loaded template with the replacement string. | |
|  | insertParagaph | Insert a new paragraph (second parameter) into the loaded template after the target string (first parameter) | |
|  | mergeHeaderFooter | Accepts a filename of a word document containing boilerplate of the header and footer to be rendered on the loaded template. | |
|  | saveDoc | Save the loaded template (after mail merging) to a new file with the filename passed in as the first parameters. | |
|  | saveAsPDF | Load an exiting MSWord document and convert into a PDF. The polymorphic version returns the PDF as ByteOutputArrayStream. | |
|  | getXML | Return a string representation (XML formatted) of the MSWord object. | |
|  | Other API(s) | defineTabStop, setMargins, setDocumentMargin, rTabTo, spaceTo  Provided for future considerations. | |

|  |  |  |  |
| --- | --- | --- | --- |
| Expected API invocation sequence | | |  |
| Sno | **API** | **Remarks** | |
| Use Case 1: Create simple paragraphs into a template | | | |
| 1 | loadTemplate | Mandatory to be the first API call, the template could be an empty Word document. | |
| 2 | Could be   * createSimpleParagraph * createJustifiedPargraph * createSimpleSignOffParagraph |  | |
| 3 | saveDoc |  | |
| 4 | saveAsPDF | If PDF version needs to be generated. | |
| Use Case 2 : Mail Merging | | | |
| 1 | Load Template | Mandatory to be the first API call | |
| 2 | mergeHeaderFooter | Headers and Footers are defined externally in another word document | |
|  | setRecipient |  | |
|  | Set merge fields via   * setTextMergeFields * setImageMergeFields * setTableMergeFields |  | |
|  | mailMerge |  | |
|  | saveDoc |  | |
|  | saveAsPDF | If PDF needs to be generated | |

## Notice (Letter) Generating Microservices - NoticeAPI

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Sno | **API** | **Description** | |
| NoticeAPI | | This class is the heart of the notice/letter generating framework. It wraps the DocXAPI and included routines to verify the validity of the mergefields which are defined and captured in the database table CM\_NoticeFields. This table is linked to the notice table CM\_Notices which contains other attribute such as pointing reference to the signatory type, and link to the signature record stored in CM\_Signatory via the notice\_id The image of the signature is stored as jpeg file in the file system and is named based on the signatory\_id field, prefixed by SI-  The filename of the word document template is prefixed by TP- and based the notice\_num and the notice\_id in the CM\_Notices.  The framework requires header/footer record to defined external as letter head. Letter head records are stored in CM\_LetterHead table and linked to CM\_Notices table via notice\_id, with the word template kept in the file named LH-<signatoryId>.docx  **The call sequence is as per the order specified in subsequent row in this table** | |
| 1 | createNotice | Pass in noticeNum, which must be defined in enum noticeNumEnum and the RecipientAddress, with all fields filled except the transaction id which is auto generated for the notice instance. | |
| 2 | populateMergeField | Accept a HashMap where the key is the merge field name defined in the CM\_NoticeFields table. The value could be a object of :   * DocXAPI.createTextField (via createTextField) * DocxAPI.createImageField (via createImageField) * DocxAPI.createQrCodeField (via createQrCodeField)   Image for QR code fields are internally generated and pass as imageField to DocXAPI.ImageField. Internally calls DocXAPI.setTextMergeFields and DocXAPI. setImageMergeFields. | |
| 3 | populateTableField | Accept a string denoting the name of the table grouping and a Hashmap where the keys and value are string value by definition. The API is called to pass in columnar content for each row in the table. It is important to note that the first column must be the merge field on the first column in the table within the word template. | |
| 4 | allFieldsReady | During call to populateMergeField and populateTableField, validation are done to ensure that the data type of the merge fields matches the definition found in the CM\_NoticeFields table. | |
| 5 | dbUploadFields | Insert the notice and merge fields information into the table CM\_NoticeInstances. | |
| 6 | generateNotice | Eventually calls the DocXAPI.mailMerge to perform the mail merging and return a byte output stream of the PDF. After this API returns the http header has to be set in order to return the PDF byte output stream to the web browser/client  HttpHeaders headers = new HttpHeaders ();  ContentDisposition contentDisposition =  ContentDisposition. Builder("inline"). filename ("GenerateUsingMocks" + "-" + recipientAddress.getTransactionId()  + ".pdf"). build ();  headers.setContentType(MediaType.parseMediaType("application/pdf"));  headers. setContentDisposition(contentDisposition);  headers. setCacheControl("must-revalidate, post-check=0, pre-check=0");  ResponseEntity<byte []> response =  new ResponseEntity<byte[]>(bos.toByteArray(), headers, HttpStatus.OK);  return response; | |

|  |  |  |  |
| --- | --- | --- | --- |
| Restcontroller - Microsevices | |  |  |
| Sno | **API** | **Description** | |
| Classes which contains information pertaining to textual merge fields for the generating RestController to handle end user for mail merging and downloading the pdf. | | | |
| 1 | GenerateNoticeAPI | localhost:8800/notice/generate/userjson/pdf  Generate pdf based on document contents provided as json in the request payload via the POST method:  noticeNum  recipientAddress {}  textFields {}  imageFileds {}  dataTables {  [ {} ]  }  Refer to generatePDF.http | |
| 2 | GenerateNoticeAPI | Endpoint is  localhost:8800/notice/regenerate/pdf?instanceId=<instanceId found in CM\_NoticeInstances)  Regenerate notice in pdf format based on notice instance record in the CM\_NoticeInstances table. Retrieve by instanceId. | |
| 3 | GenerateNoticeAPI | End point is localhost:8800/notice/getInstances?querytype=[*NAME,NOTICENUM,NOTICEID*]&value=[value]  Retrieve matching records from CM\_NoticeInstances. | |
| 4 | GenerateProgramCode | Endpoint is localhost:8800/code/generate/mergefields  Depending on the value for the property GengerateProgramCode.singleFile, in the application.properties file It navigates through the CM\_Notices table, picks up all the fields defined in CM\_NoticeFields for each record and generate mock data to generate the program codes to either   1. create the individual RestController class (with naming convention <noticeNum>.java) for each notice\_id with endpoint   or   1. create a singular RestController class (GenerateUsingMocks.java) in which each individual method is a designated endpoint which contains mock recipient and merge field data.   For both options, the generated endpoint is: localhost:8800/notice/generate/static/mockpdf/<noticeNum>.  The 2 options are mutually exclusive with the intend of providing a mechanism to test the templates. In particular, option (a) has the benefit that the logic can be reused in other API(s) that need to generate the specific notice.  These files generated under the controller subfolder “mockup” should be removed from the production system. | |
| 5 | GenerateProgramCode | The endpoint: localhost:8800/notice/generate/dynamic/mockpdf/<noticenum>  Instead of generating java classes using the microservice endpoint described in the preceding row (sno 4), this particular endpoint is the singular entry point which will dynamically generate the mock content based on the merge fields defined in the CM\_NoticeFields table. | |
| Name of Java classes that are dynamically generated when the endpoint localhost:8800/code/generate/mergefields is accessed. | | | |
|  | *GeneratedUsingMock* | Singularly generated classes described in sno 4 option (b) | |
|  | *GenerateTemplate<noticeNum>* | Individually generated classes described in sno 4 option (a) | |

# Use of Table in Document Template

The variable in a table rows needs to be defined as the 2nd row immediately after the header row (static text or can be simple merge field).

A document could have one or more tables in a single mail merge template. As MSWord does not have a concept of labelling or naming a table, the only way to identify a table is the first merge field on the 1st row after the header row. As such, the merge field on the first column MUST be unique in field name if there are more than 1 table. In addition, the 1st entry in the merge field Map object must also be the name of the 1st merge field defined in the table template. For example:

In the following hypothetical template, there are 2 tables. The merge field name for 1st column MUST be different. In this illustration, the merge field name for quantity is the same. Technically, the merge field name for the “Name” column can also be the same.

**Sales Item:**

|  |  |  |
| --- | --- | --- |
| No | Name | Quantity |
| **${no}** | ${sales-item} | ${quantity} |

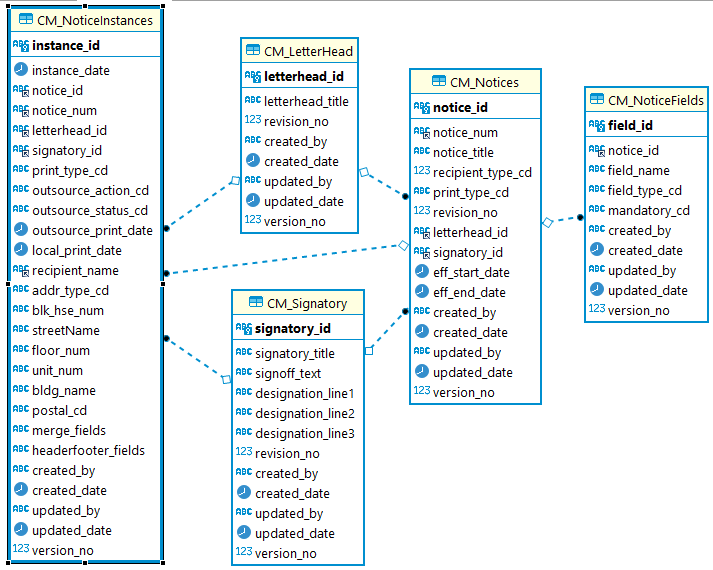
Total: ${total}

**Standard Item:**

|  |  |  |
| --- | --- | --- |
| No | Name | Quantity |
| **${sno}** | ${std-item} | ${quantity} |

Total: ${total}

# Database Structure to Support NoticeAPI



# Profiling of API Response Time Performance

The property “Profiler.enabled” in the application.properties determines if the Aspect found in the class Profiler.java would be turn-on to measure the turn-around execution time for each method that are found in the classes decorated by one of the annotations:-

1. org.springframework.stereotype.Repository
2. org.springframework.stereotype.Service
3. org.springframework.stereotype.Component
4. org.springframework.web.bind.annotation.RestController
5. org.eservice.notice.aop.ProfilerAnnotation

Annotations (a) to (d) are stereotype from the Spring Framework. For methods in any other classes not decorated by these, the profiling can be applied by annotating with the customized annotation @ProfilerAnnotation, which has no other effect except to mark the methods within the classes so that their turn-around execution time can be measured.

Beside setting the property to endure, there is a need to define the appropriate appender and logger as per below:

<?xml version="1.0" encoding="UTF-8"?>

<configuration>

    <property name="LOGPATH" value="d:/data/work/java/NoticesAPI/logs" />

    <appender name="PROFILE"

        class="ch.qos.logback.core.rolling.RollingFileAppender">

        <file>${LOGPATH}/profile.log</file>

        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">

            <Pattern>

                %d{yyyy-MM-dd HH:mm:ss}&#x9;%msg%n

            </Pattern>

        </encoder>

        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

            <!-- rollover daily -->

            <fileNamePattern>${LOGPATH}/archived/profile.%d{yyyy-MM-dd}.%i.log</fileNamePattern>

            <timeBasedFileNamingAndTriggeringPolicy

                class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">

                <maxFileSize>10MB</maxFileSize>

            </timeBasedFileNamingAndTriggeringPolicy>

        </rollingPolicy>

    </appender>

    <logger name="org.eservice.notice.aop.Profiler" level="DEBUG" additivity="false">

            <appender-ref ref="PROFILE"/>

    </logger>

    <root level="info">

        <appender-ref ref="STDOUT"/>

    </root>

</configuration>

Sample found in profile.log. The response times are measured in milliseconds.

2022-03-23 19:31:16 void org.eservice.notice.component.ApplicationStartupTask.run(String[]) 9

2022-03-23 19:32:33 String org.eservice.notice.component.NoticeAPI.getImagePath()   55

2022-03-23 19:32:33 ImageField org.eservice.notice.component.NoticeAPI.createImageField(String) 5

2022-03-23 19:32:33 List org.eservice.notice.repository.NoticeRepository.findByNoticeNum(String)    206

2022-03-23 19:32:43 void org.eservice.notice.component.NoticeAPI.createNotice(NoticeNumEnum,RecipientAddress)   9623

2022-03-23 19:32:43 List org.eservice.notice.repository.NoticeRepository.findByNoticeNum(String)    6

2022-03-23 19:32:43 List org.eservice.notice.repository.NoticeFieldsRepository.findMergeFieldsByNoticeId(String)    23

2022-03-23 19:32:43 TextField org.eservice.notice.component.NoticeAPI.createTextField(String)   0

2022-03-23 19:32:43 TextField org.eservice.notice.component.NoticeAPI.createTextField(String)   0

# Logging of Hibernate Generated SQL statements

There are 2 ways to do so. One is to define the following settings in the application.properties to turn-on hibernate logging.

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

The alternative and preferred way is to leverage on the p6spy library. This requires the database settings in application.properties

database.driver=com.mysql.cj.jdbc.Driver

database.url=jdbc:mysql://localhost:3306/appstore?createDatabaseIfNotExist=true&useSSL=false&allowPublicKeyRetrieval=true

to be replaced by

database.driver=com.p6spy.engine.spy.P6SpyDriver

database.url=jdbc:p6spy:mysql://localhost:3306/appstore

In addition, there is a need to have the spy.properties file in the classpath with the following settings:

stacktrace=false

appender=com.p6spy.engine.spy.appender.FileLogger

append=true

logfile=/data/work/java/noticesAPI/logs/p6spy.log

customLogMessageFormat=%(currentTime)|%(executionTime)|%(category)|%(sqlSingleLine)

# Known Bugs Pending Resolution

* The instance\_date in CM\_NoticeInstances is always set to 1 day before current date and the time is consistently 16:00:00. So far, unable to locate the root cause.

1. The resulting encrypted text would contain the necessary token to decrypt the text into it original value based on algorithm that is the source code for the Crypto class which is deliberately not documented here due to security considerations. [↑](#footnote-ref-1)