aplonREPORTER

aplonREPORTER User Manual

SWIFT MT MESSAGE REPORTER

** aplonREPORTER 16.0**

Paymentcomponents Ltd

288 Bishopsgate,

EC2M 4QP

London, UK

---

www.paymentcomponents.com

info@paymentcomponents.com

Overview

Use aplonREPORTER to automatically generate a PDF report directly from an existing SWIFT MT message. With support for all SWIFT MT Categories from 1 to 9 it offers the ideal way for a bank to provide its customers with an attractive, easily readable, version of any SWIFT MT message.

aplonREPORTER has been developed in Java to provide the best cross-platform flexibility within the banks environment.

In addition to the stand-alone functionality, aplonREPORTER can also be integrated with existing bank applications to generate PDF reports on-the-fly.

- Cross platform
- It can be used as a stand-alone application or deployed on a server depending upon the level of license purchased.
- Available for all SWIFT MT message categories from 1 to 9

aplonREPORTER parses the information from a message and outputs the following user-friendly report:

aplonREPORTER contains mapping definitions for each tag and message type to enable a user-friendly description of a field to be displayed on the report.

aplonREPORTER (DEMO)

The aplonREPORTER DEMO provides an easy way to input the text of an existing SWIFT MT message and generate a PDF report.

In order to run GUI, first set the properties in **gui.properties** file.

In aplonReporter-x.x folder, run the command:

```
java -cp lib/\*:aplonReporter-x.x.jar gr.datamation.swift.SwiftMessageParser
```

The SWIFT message text window can be populated either by pasting the message text or loading from a file.

The PDF can then be generated by the click of a button. The Output window displays a log of activity.

aplonREPORTER (API)

The aplonREPORTER command line provides an easy way to pass the text of existing SWIFT MT messages and generate PDF reports.

Calling parameters

The API can be called directly using the following combination of parameters.

Single message – report generation

Input and Output filename

In aplonReporter-x.x folder, run the command:

```
java -cp lib/*:aplonReporter-x.x.jar gr.datamation.swift.CommandLineMessageParser c:\aplonREPORTER\msgs\103.txt c:
\aplonREPORTER\pdfs\103_report.pdf
```

Multi message (RJE) – report generation

Input filename of RJE file true and target directory for output files

The `true` parameter indicates to aplonREPORTER that the input file is an RJE format file containing multiple messages.

The SWIFT message reference is used as the file name for each of the reports generated.

In aplonReporter-x.x folder, run the command:

```bash
java -cp lib/*:aplonReporter-2.0.jar gr.datamation.swift.CommandLineMessageParser c:\aplonREPORTER\msgs\msgs.rje t
rue c:\aplonREPORTER\pdfs\
```

aplonREPORTER (SDK)

When added to a Java project, the aplonREPORTER developer library provides an easy way to add the generation of PDF SWIFT MT message reports to existing projects.

The following examples demonstrate how easily you can access the power of aplonREPORTER from within your own projects.

The 'templates' folder should be placed in the same directory with the 'test' package.

Create PDF directly to the file system

PDF Creation in the file system

```
package test;

import gr.datamation.swift.report.SwiftMessageReport;

public class Test {
        public void createReport(String msgText) {
                SwiftMessageReport smr = new SwiftMessageReport("/path/to/SWIFT_REPORT"); //the parameter is
the name of desired .jasper file
                try {
                        smr.createReportFile(msgText, "/path/to/output.pdf");
                        //argument 1 is the swift message plain text, argument 2 is the output file name
                } catch (Exception e) {
                        System.err.println("Failed to parse the message and create a report due to exception: "
+ e.getMessage());
                }
        }

        public static void main(String[] args) {
                new Test().createReport("{1:F21SIIBUS30CXXX0038089609}{4:{177:0906100929}{451:0}}{1:
F01SIIBUS30CXXX0038089609}{2:O1020731090610LPLPXXXX4C0900002194660906100331N}{3:{108:MT102 002 OF 026}{119:STP}}
{4:\n" +
                        ":20:00049\n" +
                        ":23:SPAY\n" +
                        ":50A:/9631357\n" +
                        "YMBKJPJT\n" +
                        ":71A:SHA\n" +
                        ":21:02AA14\n" +
                        ":32B:USD55443322,44\n" +
                        ":59A:/732-896-C\n" +
                        "BMSXMXMM\n" +
                        ":33B:USD55443322,44\n" +
                        ":32A:011231USD55443322,44\n" +
                        ":19:55443322,44\n" +
                        "-}{5:{CHK:AF7608BD6DED}{TNG:}}{S:{SPD:}{SAC:}{COP:P}}\n");
        }
}
```

Create PDF as a Byte Stream

The file path, for example

`C:\\\\aplonREPORTER\\\\msgs\\\\103.txt`

should be added in the program arguments.

PDF Creation as Byte Stream

```
package gr.datamation.swift;

import gr.datamation.swift.report.SwiftMessageReport;

import java.io.*;

public class TestStream {
        /**
         * @param args
         */

        public static void main(String[] args) {
                SwiftMessageReport smr = new SwiftMessageReport("templates/SWIFT_REPORT"); //the parameter is
the name of desired .jasper file
                String messageText = "";
                BufferedReader buf;
                try {
                        buf = new BufferedReader(new FileReader(args[0]));
                        String line = buf.readLine();
                        while (line != null) {
                                messageText += line + "\r\n";
                                line = buf.readLine();
                        }
                } catch (FileNotFoundException e1) {
                        e1.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                try {
                        if (args.length == 1) {
                                byte[] byteArray = smr.generateReportByteArray(messageText, "pdf");
                                InputStream is = new ByteArrayInputStream(byteArray);
                                OutputStream out = new FileOutputStream(new File("/path/to/output.pdf"));
                                int read = 0;
                                byte[] bytes = new byte[1024];

                                while ((read = is.read(bytes)) != -1) {
                                        out.write(bytes, 0, read);
                                }

                                is.close();
                                out.flush();
                                out.close();
                        }
                } catch (Exception e) {
                        System.err.println("Failed to parse the message and create a report due to exception: "
+ e.getMessage());
                }
        }
}
```

Customizing the PDF

Customize Message Title

If you want to have another description, than the default for a message title, then you have to update the **titles.propeties** file int the **templates** folder.

custom_messages.properties

```
103=Custom desc for MT103 message 103_13C=Custom desc for Sender's reference
```

Add sender and receiver descriptions

If you want to add descriptions for the sender or the receiver, then you have to add a new property file in the **templates** folder. Name it **bic_directory.properties** **custom_messages.properties**

custom_messages.properties
`BIGABRRJSPO=Sender's custom description CSFBGB2LXXX=Receiver's custom description`

Add custom fields.

If you want to send some custom fields to the PDF, then you have to add them to the end of the MT message. You can add up to 99 custom parameters, as follows

MT103 with custom parameters

MT103 with custom parameters

```
public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put("CUSTOM_PROPERTY_1", "This is my custom property");
        SwiftMessageReport smr = new SwiftMessageReport("templates/CUSTOM_SWIFT_REPORT", properties);//the
parameter is the name of desired .jasper file

        String messageText = "{1:F01AAAAGRA0AXXX0057000289}{2:O1030919010321BBBBGRA0AXXX0057000171010321 0920N}
{4:\n" +
                ":20:5387354\n" +
                ":23B:CRED\n" +
                ":23E:PHOB/20.527.19.60\n" +
                ":32A:000526USD1101,50\n" +
                ":33B:USD1121,50\n" +
                ":50K:FRANZ HOLZAPFEL GMBH\n" +
                "VIENNA\n" +
                ":52A:BKAUATWW\n" +
                ":59:723491524\n" +
                "C. KLEIN\n" +
                "BLOEMENGRACHT 15\n" +
                "AMSTERDAM\n" +
                ":71A:SHA\n" +
                ":71F:USD10,\n" +
                ":71F:USD10,\n" +
                ":72:/INS/CHASUS33\n" +
                "-}{5:{MAC:75D138E4}{CHK:DE1B0D71FA96}}";

        try {
                if (args.length == 1) {
                        byte[] byteArray = smr.generateReportByteArray(messageText, "pdf");
                        InputStream is = new ByteArrayInputStream(byteArray);
                        OutputStream out = new FileOutputStream(new File("/path/to/output.pdf"));
                        int read = 0;
                        byte[] bytes = new byte[1024];

                        while ((read = is.read(bytes)) != -1) {
                                out.write(bytes, 0, read);
                        }

                        is.close();
                        out.flush();
                        out.close();
                }
        } catch (Exception e) {
                System.err.println("Failed to parse the message and create a report due to exception: " + e.
getMessage());
        }
}
```

and add the custom parameters to the jrxml file.

Preview

Installation Instructions

Project embedding

The SDK is packaged within a JAR file, that can be added as a dependency to any JVM project within the IDE. During the deployment, make sure that the JAR file is included in the runtime classpath. The JASPER file should be also placed within the scope of the classpath.

Requires at least JDK 1.6.x

External dependencies:

```
commons-beanutils-1.8.0
commons-collections-3.1
commons-digester-2.1
commons-logging-1.1.1
groovy-1.8.1
hamcrest-core-1.3
iText-2.1.7
jasperreports-4.7.1
jdom2-2.0.6
jdom-1.1.3
smv-16.0.0
junit-4.11
```

Modifying the report layout

The templates folder contains a sample bank logo image file that should be replaced with the required image. The SWIFT_REPORT.jasper file can be copied and customized using the Jasper iReport tool. iReport can be downloaded from the following link.

To use the customized report with aplonREPORTER just edit the **gui.properties** file to reflect the new report name.

Contact and Support

The support team is available to answer questions on using the component, code assistance, error determination, ideas, examples etc.

Forward your questions to: info@paymentcomponents.com

Visit www.paymentcomponents.com for other components and services by PaymentComponents.