

XML Processing of SLA Files

Thomas Zastrow

Table of Content

Motivation.....	3
Basics.....	3
Examples.....	4
Search and Replace.....	4
Execute.....	4
The Source.....	5
The FormLetter Class.....	6
Execute.....	6
The Source.....	7
Conclusion.....	8

License

Copyright (c) 2009 Thomas Zastrow

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license can be found on-line:
<http://www.fsf.org/licensing/licenses/fdl.txt>

Motivation

In general, file formats in Desktop Publishing are very complex. Scribus' SLA format is no exception. But the facts, that SLA is a) Opensource and b) pure XML, makes it possible to manipulate the files with common XML techniques from outside Scribus.

Basics

Manipulating XML files is possible with a bunch of common XML techniques, including:

- XSL-T
- Xquery
- DOM or S(T)AX parsing

In this document, only one of these approaches is shown: manipulating SLA files with the built-in DOM parser of Java.

If you have Java 6 installed on your machine, download the file *jScribus.tar.gz* and unzip it to your harddrive. The resulting directory *jScribus* contains a Netbeans project which manages the source. For your convenience, the subfolder *dist* contains a compiled version of the project (file *jScribus.jar*). With the help of this jar file, you can execute the examples in this documentation.

If you take a look at the source code, you will see that it makes use of some external classes:

- FileFunctions encapsulates Javas possibilities to save and read files
- DOMHandler encapsulates the DOM functionality of Java

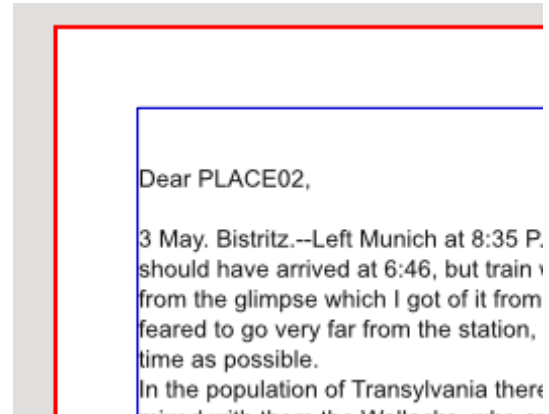
You don't need a look in greater depth at these classes – they are just helper classes and not relevant for understanding XML processing of SLA files. You can find these classes in the library *darkzone.jar* in the subfolder *lib* of the *dist* directory. So, if you move the file *jScribus.jar* to another location than the *dist* directory, don't forget to add *darkzone.jar* to your classpath!

Examples

Search and Replace

Execute

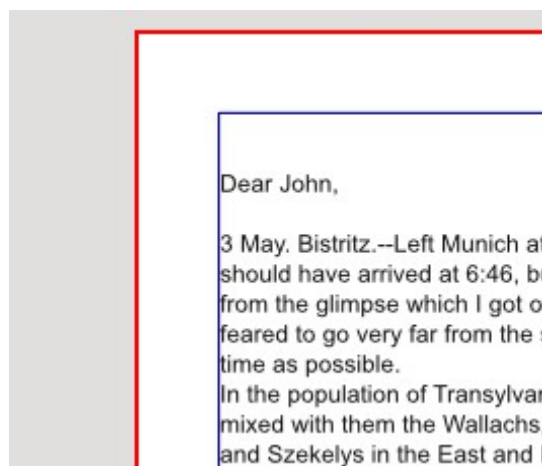
The program searches all textboxes in a Scribus file for a given string and replaces it by another string. To use this program, create a Scribus file and place in one or more textboxes the string which you want to be replaced. Here, it is PLACE02.



Now, call the program with the following command and don't forget to change the parameters so that they fit:

```
java -cp jScribus.jar jscribus.SearchAndReplace dok1.sla PLACE02 John
```

Now, open the file again in Scribus und you will see the difference:



The Source

Please note: the following shows just some of the most important lines of code, not the complete class!

After creating the DOM object and initialize it with a SLA file, the following Xpath statement extracts all ITEXT elements which are containing the search string:

```
NodeList nl = dh.evaluateXPath("//ITEXT[contains(@CH, \"" + search + "\"]");
```

If the list contains some elements, we can loop over them and replace the search string with the replace string:

```
for (i = 0; i < nl.getLength(); i++) {  
    Element aNode = (Element) nl.item(i);  
    String aDate = aNode.getAttribute("CH");  
    aDate = aDate.replace(search, replace);  
    aNode.setAttribute("CH", aDate);  
}
```

Just save back the DOM tree to the SLA file and you are done.

The FormLetter Class

This class extends the functionality of the SearchAndReplace class a little bit. Instead of a document wide replacement of strings, it

- opens additional a datafile and reads the data to search & replace line by line
- for every line in the datafile, it clones the first page of a SLA document and replaces the strings on this page with the strings from the datafile

The datafile should have the following format:

```
PLACE01=4/1/09;PLACE02=John;PLACE03=Peter Pattern
```

```
PLACE01=4/2/09;PLACE02=Peter;PLACE03=Pitt Patt
```

Every line consists of key-value pairs of strings to search and replace, separated by a semicolon. In the example above, the string „PLACE01“ on the first page will be replaced by „ 4/1/09“, on the second page „PLACE01“ will be replaced by „ 4/2/09“ and so on: „PLACE03“ will change to „Peter Pattern“ on the first and „Pitt Patt“ in the second page etc. The resulting Scribus file contains as many pages as lines in the datafile plus one (the original one), where on every page the search strings are replaced by the replace strings.

Execute

So, you need a Scribus file which contains exactly one page. On that page, there should be placeholders for the strings which should be replaced.

Second, you need a datafile in the format as described above. Now, you can call the programm:

```
java -cp jscribus.jar jscribus.FormLetter dokument.sla data.dat
```

where dokument.sla is the SLA file and data.dat the data file. When the program finished successfully, the original SLA file will contain as more pages as lines are in the datafile.

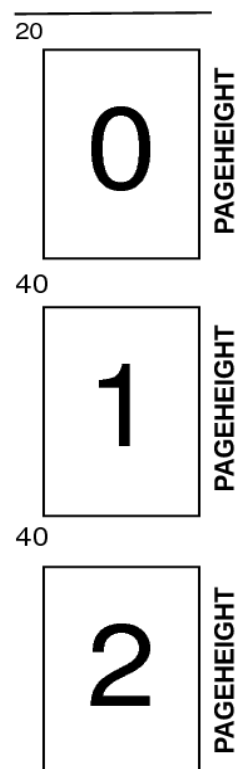
The Source

For every line in datafile, we clone the first page (method clonePage()). Therefore, we need some Xpath statements to clone the page itself and all the objects on that page:

count(//PAGE)	Gives the total number of pages in the document
//PAGE[@NUM="3"]	Returns page number three
//PAGEOBJECT[@OwnPage="3"]	All objects on page number 3

Some words on the position of the PAGEOBJECTs. Every object has an attribute YPOS which defines its vertical position in the SLA file. This is a continuous value which is independent of the OwnPage attribute. With standard preferences, you can calculate it as follows:

- 20 are above the first page
- 40 are separating two pages
- the height of a page can be found in the attribute PAGEHEIGHT of a page object



So, if you clone a page with all its objects, don't forget to clone also all the objects on that page and

recalculate the new position of the cloned objects!

After that is done, we just need to execute some Search-and-Replace operations on the cloned page respective its objects.

Conclusion

SLA files are wellformed XML. That opens up the world of XML processing to Desktop Publishing. Of course, creating and editing a layout is still better done manually in Scribus. But once a layout is created, XML processing allows to conflate layout with data in an automatic way. This can be done with XSL-T, Xquery (from within databases, but putting SLA files into a database is another story), or programming languages with a mixture of Xpath and parsing techniques.

One thing, which would complete the workflow, is still missing. This is the possibility to render SLA files into PostScript- or PDF files on the commandline. Until now, every – automatically created – SLA file needs to be opened in Scribus and then exported to PDF or whatever *manually*. Adding the possibility to render SLA files on the commandline would be the last step in an automated workflow.

The top dogs in Desktop Publishing – Quarks Xpress and Adobes InDesign – have seen the signs of time and have added XML functionality to their DTP flagships. Up to now, these solutions are islands in the ocean of Desktop Publishing. XML processing could build up bridges between the islands and SLA as an open format could be the pacemaker. Another player in this field is XSL-FO, XSL Formating Objects. With this open standard, it is also possible to automatically create layouts and – in contrast to Scribus and its SLA files – render them directly on the commandline. But XS-FOs approach is a little bit other than the fram oriented one of Scribus and other classic DTP applications.

To be continued.