Retrieving the incoming payments from the XML statements of postfinance bank account

Emin Gabrielyan
Switzernet
Updated on 2008-03-20
Created on 2008-03-07

This document describes a method for retrieving the incoming payments of customers arrived to the postfinance account of the company. We retrieve the data from daily XML files, downloadable from the postfinance web interface. We show how to create a DOC file containing full information on incoming payments. The red payment slips and the electronic payments are sorted in a single flow according to the order of their arrival.

You will find scripts for downloading, which help to convert any set of consecutive XML files, downloaded from postfinance interface, into an output DOC file. Our script can be extended for inserting the records directly into a database. The advantage of the new method is that the electronic payments and scanned images can be retrieved, stored, and processed together.

1. Introduction

Sections 2 and 3 describe the payment retrieval and submission procedure to be carried out recurrently by the company employees. You will find screenshots of the postfinance interface for downloading the XML files. Note that the XML file generation must be explicitly requested to the bank account manager, since they are not generated by default. Section 3.3, is required only once for the transition from the old style submissions to the new style (not interesting for an outside reader).

The sections following section 3 are not required for recurrent retrievals and submissions. These sections describe the scripts and the details of the XML format used by postfinance. These sections are needed for understanding the script and for any modifications in scripts (e.g. due to possible format changes of XML files provided by postfinance).

Section 4 introduces the format of XML files used by postfinance. Section 5 describes our Perl script. We rely on the powerful mechanism of Perl's regular expressions for processing the XML entries. Section 6 describes the main bash script which processes blocks of input XML files downloaded from postfinance interface.

2. The daily and weekly XML statement files

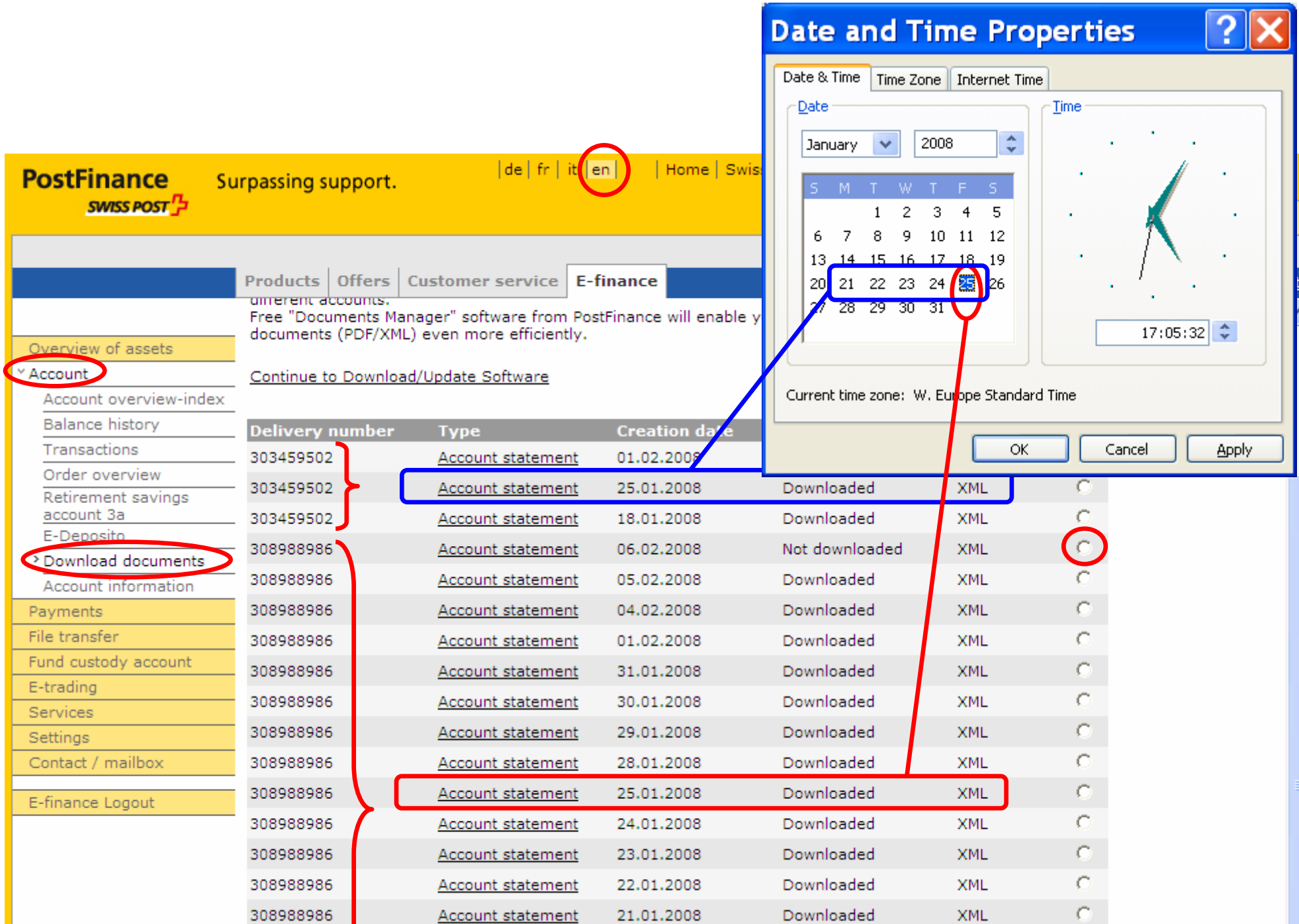
The postfinance web account interface provides an access to the XML files containing the daily and weekly financial statements of the postfinance account. The XML files are generated automatically by postfinance every open day.

2.1. Location of XML files

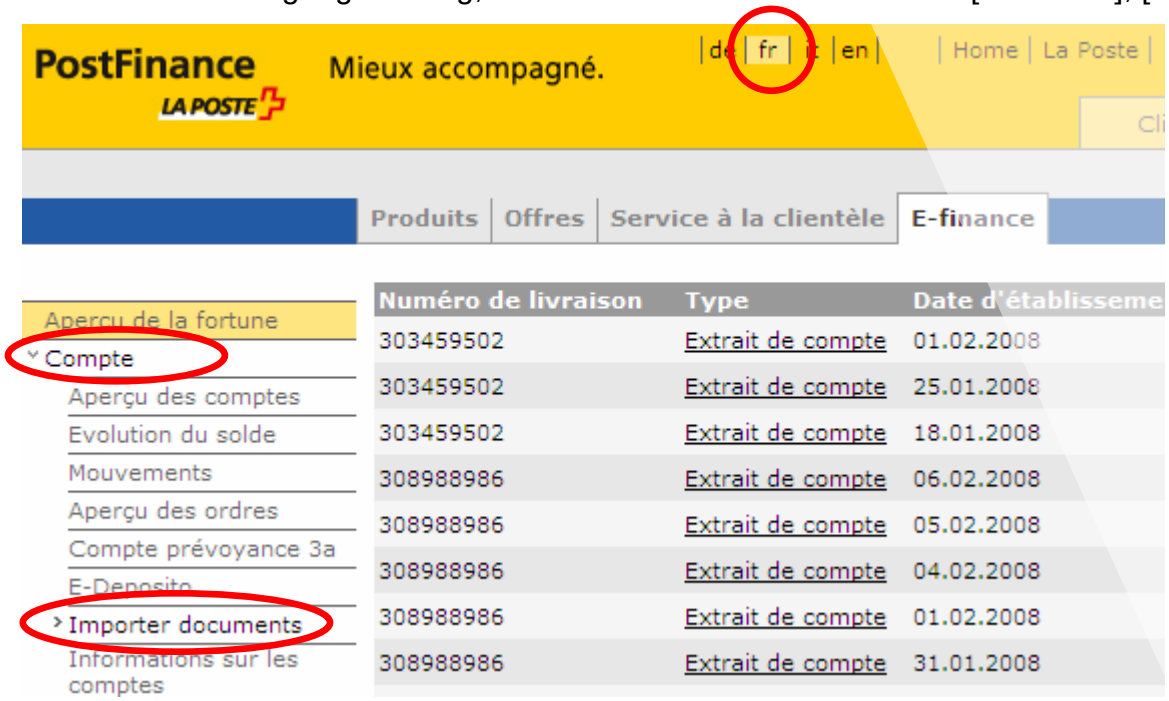
Login into the postfinance account following the procedure described in the first section of the following document [http://switzernet.com/company/071226-customer-payments-postfinance/]. You need the card reader and the green postfinance card with a microchip. If you are already familiar with the login procedure you do not need to follow the above link.

With English language settings, the XML files are accessed via [E-finance], [Account], [Download documents] menu. Our account has currently weekly and daily XML files. The weekly XML files have a delivery number equal to 303459502 and the daily XML files have a delivery number 308988986. The weekly files are generated each Friday. They contain the transactions following the closure of the previous weekly XML file (generated the previous Friday). Since the files are closed in the middle of the day (in the afternoon), the weekly files often start with the last transactions of the previous week's Friday. The daily files are generated each day in the afternoon. They contain the transactions following the closure moment of the previous day's XML file. The daily XML file often start with a last few transactions of the previous day.

Except we experience large gaps in processing of the payments, we need only the daily XML files. To download an XML file, select it with the radio-button and click on the [Download file] button at the bottom of the page. Make sure to not confound the daily XML files (delivery number: 308988986) with weekly files (delivery number: 303459502) having the same creation date:


PostFinance
Surpassing support.
SWISS POST
de | fr | it | en | Home | Swis
Products | Offers | Customer service | E-finance
different accounts.
Free "Documents Manager" software from PostFinance will enable y
documents (PDF/XML) even more efficiently.
Continue to Download/Update Software
Overview of assets
Account
Account overview-index
Balance history
Transactions
Order overview
Retirement savings account 3a
E-Deposito
Download documents
Account information
Payments
File transfer
Fund custody account
E-trading
Services
Settings
Contact / mailbox
E-finance Logout
Delivery number | Type | Creation date
303459502 | Account statement | 01.02.2008
303459502 | Account statement | 25.01.2008 | Downloaded | XML
303459502 | Account statement | 18.01.2008 | Downloaded | XML
308988986 | Account statement | 06.02.2008 | Not downloaded | XML
308988986 | Account statement | 05.02.2008 | Downloaded | XML
308988986 | Account statement | 04.02.2008 | Downloaded | XML
308988986 | Account statement | 01.02.2008 | Downloaded | XML
308988986 | Account statement | 31.01.2008 | Downloaded | XML
308988986 | Account statement | 30.01.2008 | Downloaded | XML
308988986 | Account statement | 29.01.2008 | Downloaded | XML
308988986 | Account statement | 28.01.2008 | Downloaded | XML
308988986 | Account statement | 25.01.2008 | Downloaded | XML
308988986 | Account statement | 24.01.2008 | Downloaded | XML
308988986 | Account statement | 23.01.2008 | Downloaded | XML
308988986 | Account statement | 22.01.2008 | Downloaded | XML
308988986 | Account statement | 21.01.2008 | Downloaded | XML
Date and Time Properties
Date & Time | Time Zone | Internet Time
Date
January | 2008
S M T W T F S
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
Time
17:05:32
Current time zone: W. Europe Standard Time
OK | Cancel | Apply

With the French language setting, the XML files can be found under [E-finance], [Compte], [Importer documents] menu section:

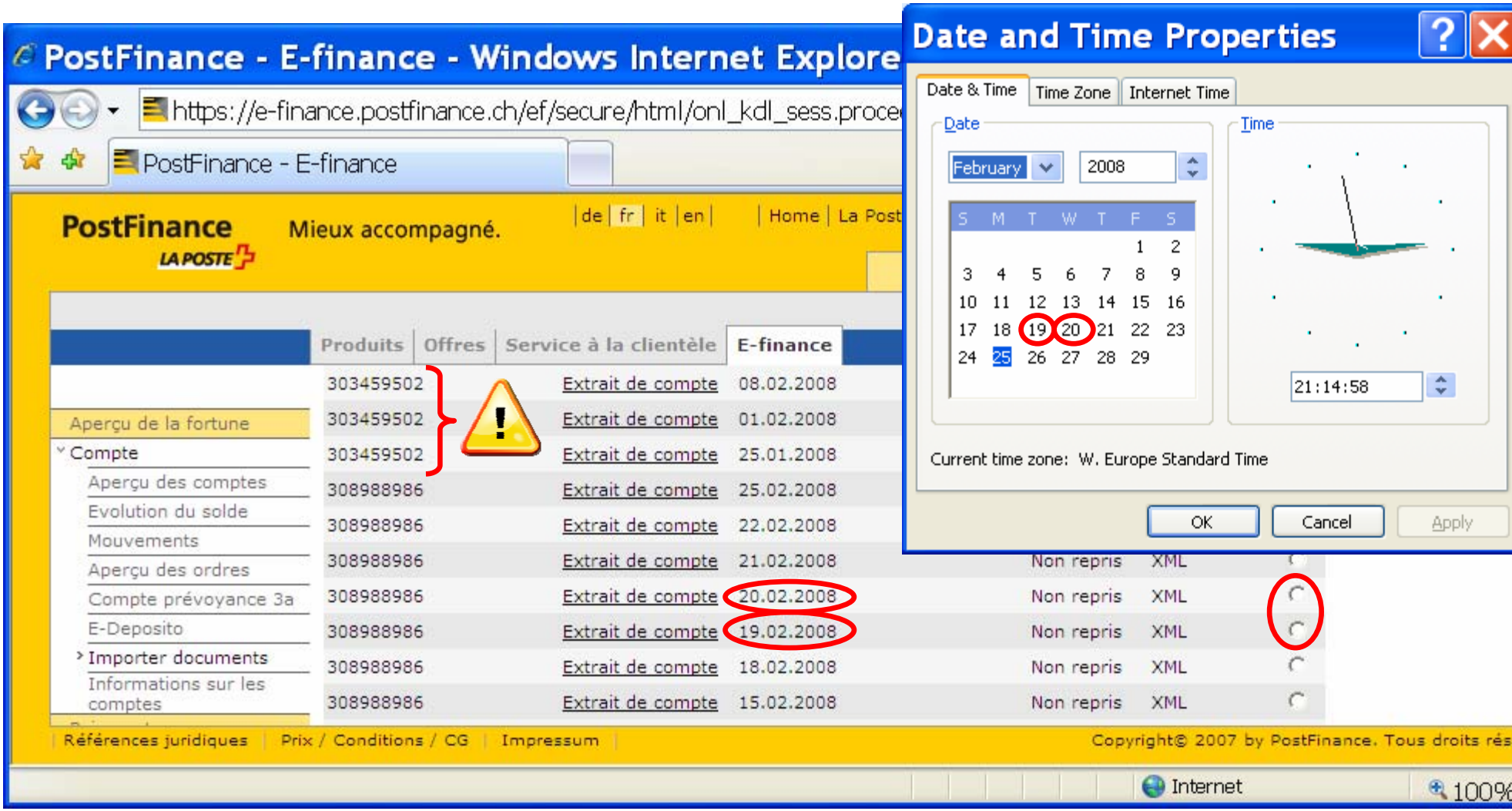


2.2. Which XML files to download for retrieving the payments of a given range of dates

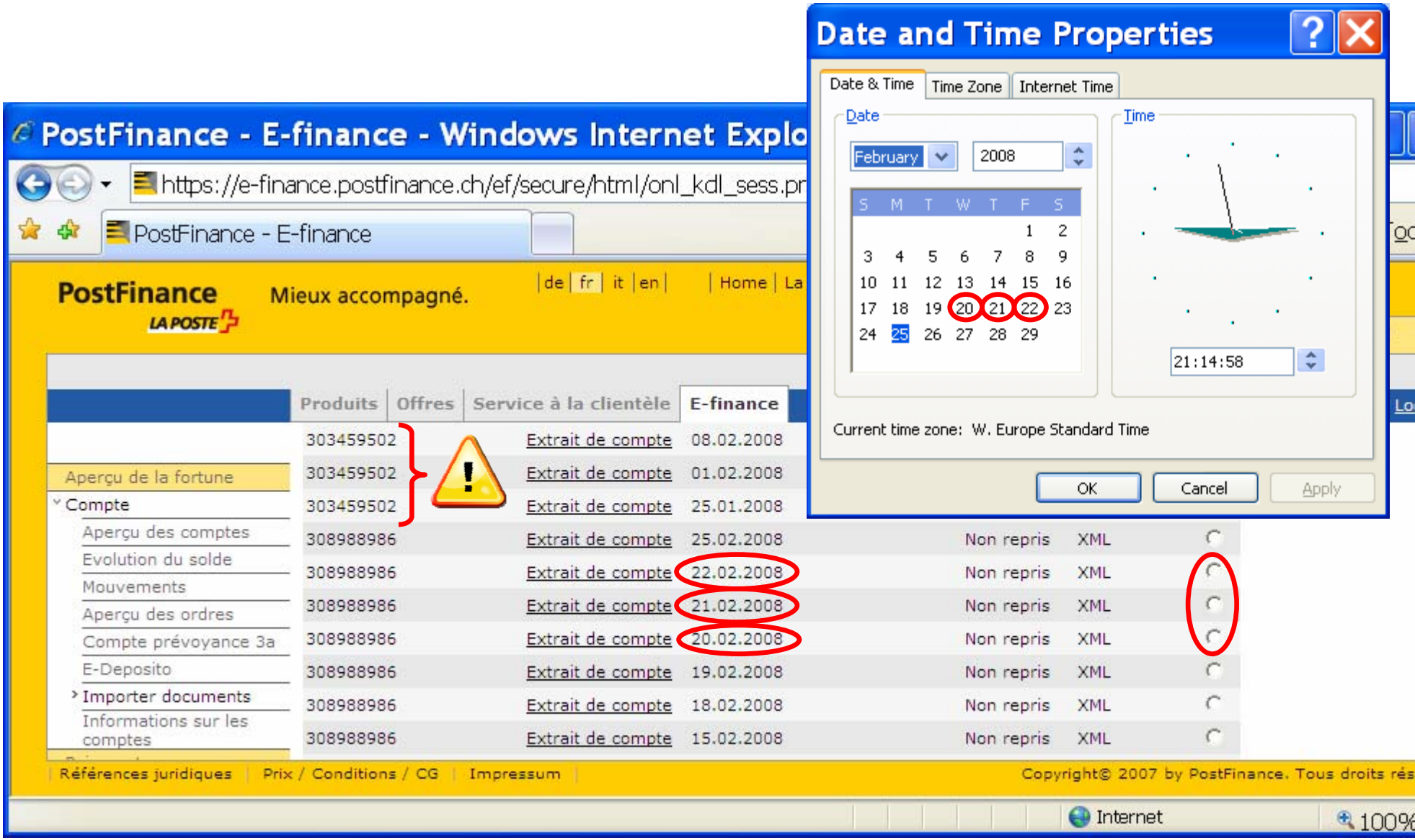
You cannot obtain all transactions of a given day by the XML file created that day. Since the XML statement file is ready in the middle of the day, it contains the transactions occurred in the beginning of the day but does not contain the transactions of the end of the day. In order to cover all transactions of the day, you need to download also the following XML file which starts with the missing transactions of the day in question.

So, the transactions of any given day can be created not earlier than the next day, somewhere in the afternoon, when you can download not only the XML file of the desired day but also the file of the following day.

For example, we can obtain the full statement of Tuesday (2008-02-19), by downloading the file created that Tuesday, and the following file created on Wednesday. The earlier moment you can do it, is the Wednesday afternoon (obviously, if we are on Wednesday, we will not see the XML files of Thursday, Friday, etc, in contrast to the screenshot below):


PostFinance - E-finance - Windows Internet Explore
https://e-finance.postfinance.ch/ef/secure/html/onl_kdl_sess.proce
PostFinance - E-finance
PostFinance
Mieux accompagné.
de | fr | it | en | Home | La Post
Produits | Offres | Service à la clientèle | E-finance
Aperçu de la fortune
Compte
Aperçu des comptes
Evolution du solde
Mouvements
Aperçu des ordres
Compte prévoyance 3a
E-Deposito
Importer documents
Informations sur les comptes
303459502 Extrait de compte 08.02.2008
303459502 Extrait de compte 01.02.2008
303459502 Extrait de compte 25.01.2008
308988986 Extrait de compte 25.02.2008
308988986 Extrait de compte 22.02.2008
308988986 Extrait de compte 21.02.2008 Non repris XML
308988986 Extrait de compte 20.02.2008 Non repris XML
308988986 Extrait de compte 19.02.2008 Non repris XML
308988986 Extrait de compte 18.02.2008 Non repris XML
308988986 Extrait de compte 15.02.2008 Non repris XML
Références juridiques | Prix / Conditions / CG | Impressum
Copyright© 2007 by PostFinance. Tous droits rés
Internet
100%
Date and Time Properties
Date & Time
Time Zone
Internet Time
Date
February
2008
S M T W T F S
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29
Time
21:14:58
Current time zone: W. Europe Standard Time
OK
Cancel
Apply

By downloading more consecutive files we can obtain transactions of two or more days. As before, we need to download one file more. For example on Friday, February 22, by downloading three consecutive files (created on February 20, 21, and 22) we can obtain transactions for days from 2008-02-20 thru 2008-02-21 inclusive:


PostFinance - E-finance - Windows Internet Explo
https://e-finance.postfinance.ch/ef/secure/html/onl_kdl_sess.pr
PostFinance - E-finance
PostFinance
Mieux accompagné.
de | fr | it | en | Home | La
Produits | Offres | Service à la clientèle | E-finance
Aperçu de la fortune
Compte
Aperçu des comptes
Evolution du solde
Mouvements
Aperçu des ordres
Compte prévoyance 3a
E-Deposito
Importer documents
Informations sur les comptes
303459502 Extrait de compte 08.02.2008
303459502 Extrait de compte 01.02.2008
303459502 Extrait de compte 25.01.2008
308988986 Extrait de compte 25.02.2008 Non repris XML
308988986 Extrait de compte 22.02.2008 Non repris XML
308988986 Extrait de compte 21.02.2008 Non repris XML
308988986 Extrait de compte 20.02.2008 Non repris XML
308988986 Extrait de compte 19.02.2008 Non repris XML
308988986 Extrait de compte 18.02.2008 Non repris XML
308988986 Extrait de compte 15.02.2008 Non repris XML
Références juridiques | Prix / Conditions / CG | Impressum
Copyright© 2007 by PostFinance. Tous droits rés
Internet
100%
Date and Time Properties
Date & Time
Time Zone
Internet Time
Date
February
2008
S M T W T F S
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29
Time
21:14:58
Current time zone: W. Europe Standard Time
OK
Cancel
Apply

Conclusion: for covering a desired range of dates, you need to download the available XML files of all days of the range plus the XML file following the last day of the range (containing the end-of-the-day transactions of the last day of the range).

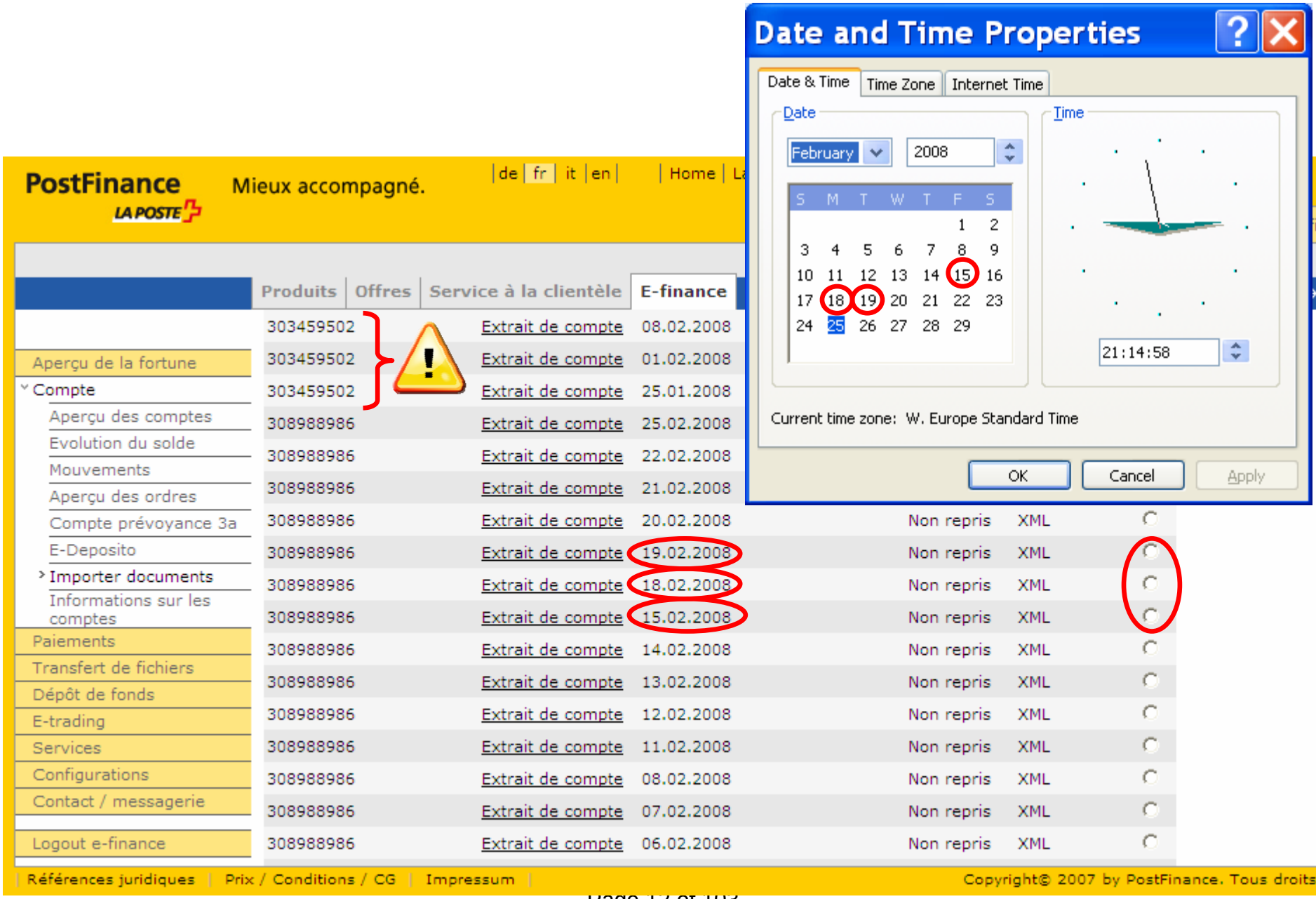
2.2.1. Weekends and holidays

No XML files are generated during weekends. The Monday XML files start with the end-of-day transactions of the last week's Friday and, in case if there were transactions also during the weekend, the Monday XML file will include them as well.

Since the weekends may occasionally contain transactions. Do not create Friday transactions and Monday transactions individually. Let us say, you attempt to do so. For only Friday transactions you will need the XML files of Friday and Monday. For only Monday transactions you will need the XML files of Monday and Tuesday. If there are any transactions occurred during weekend, they will be retrieved twice: first by the pair of Friday and Monday XML files, then by the pair of Monday and Tuesday XML files.


To avoid the retrievals of occasional weekend transactions more than one time, always retrieve the transactions of Friday and Monday together, in one go. It means you have to wait until Tuesday, in order to download three XML files (created on Friday, Monday, and Tuesday) and to retrieve the transactions from Friday thru Monday inclusive (4 consecutive but possibly containing empty days).

For example if on the Tuesday, February 19, we download the files created on Friday, Monday, and Tuesday (on the same day), we will be able to retrieve the statements of 4 days from 2008-02-15 thru 2008-02-18 inclusive:


Date and Time Properties
Date & Time
Time Zone
Internet Time
Date
February
2008
Time
21:14:58
Current time zone: W. Europe Standard Time
OK
Cancel
Apply
PostFinance
Mieux accompagné.
de | fr | it | en | Home
Produits | Offres | Service à la clientèle | E-finance
Aperçu de la fortune
Compte
Aperçu des comptes
Evolution du solde
Mouvements
Aperçu des ordres
Compte prévoyance 3a
E-Deposito
Importer documents
Informations sur les comptes
Paiements
Transfert de fichiers
Dépôt de fonds
E-trading
Services
Configurations
Contact / messagerie
Logout e-finance
303459502 Extrait de compte 08.02.2008
303459502 Extrait de compte 01.02.2008
303459502 Extrait de compte 25.01.2008
308988986 Extrait de compte 25.02.2008
308988986 Extrait de compte 22.02.2008
308988986 Extrait de compte 21.02.2008
308988986 Extrait de compte 20.02.2008 Non repris XML
308988986 Extrait de compte 19.02.2008 Non repris XML
308988986 Extrait de compte 18.02.2008 Non repris XML
308988986 Extrait de compte 15.02.2008 Non repris XML
308988986 Extrait de compte 14.02.2008 Non repris XML
308988986 Extrait de compte 13.02.2008 Non repris XML
308988986 Extrait de compte 12.02.2008 Non repris XML
308988986 Extrait de compte 11.02.2008 Non repris XML
308988986 Extrait de compte 08.02.2008 Non repris XML
308988986 Extrait de compte 07.02.2008 Non repris XML
308988986 Extrait de compte 06.02.2008 Non repris XML
Références juridiques | Prix / Conditions / CG | Impressum
Copyright© 2007 by PostFinance. Tous droits réservé


The same is valid for any holiday. In case of any closed (non-working) days, you need to create the transactions of a period including one day before and one day after the holiday. You can do it only the day following the first open day after the holidays.

3. Creating and submitting the DOC files with incoming payments

3.1. Download the XML files and convert them into DOC

Create an empty working folder. Download the script [xml2doc.zip]. Unzip its content into your working folder. Consult <payment_processing_queue> in order to find the last submission of postfinance payments. The filenames of submitted files look as follows:

```
080201-080204-post-CHF11279.36-N136.doc
```

If we are making the submission with the new filename format the very first time, then look for filenames of the previous format. They look as follows:

```
080125-080128-postfinance-xml.doc
```

Make sure that you found the last submission range. The display order may often change: an old submission thread may appear before the last submission because of fresh modifications in the old submission thread.

Find in the postfinance account the [tar.gz] XML files for the days following the last submission appearing in the <payment_processing_queue>. Follow the instructions of section 2.2 and download the required [tar.gz] XML files into your working folder.

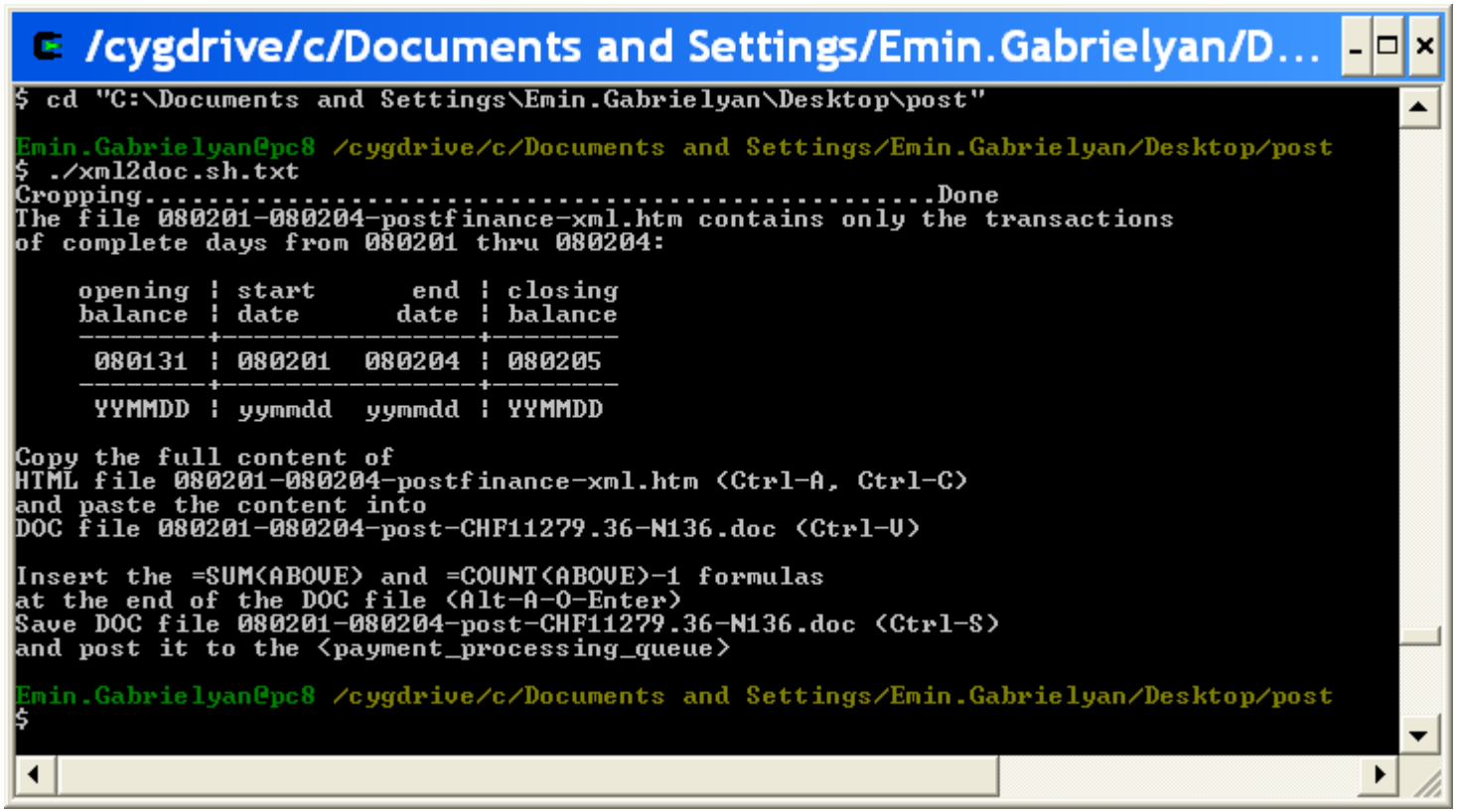
 Keep in the working folder only the [tar.gz] files required for your date range. Old [tar.gz] files may cause errors or will extend the dates beyond the limits you wish to cover.

Open a Cygwin window. Move into the working folder. Type:

```
./xml2doc.sh.txt
```

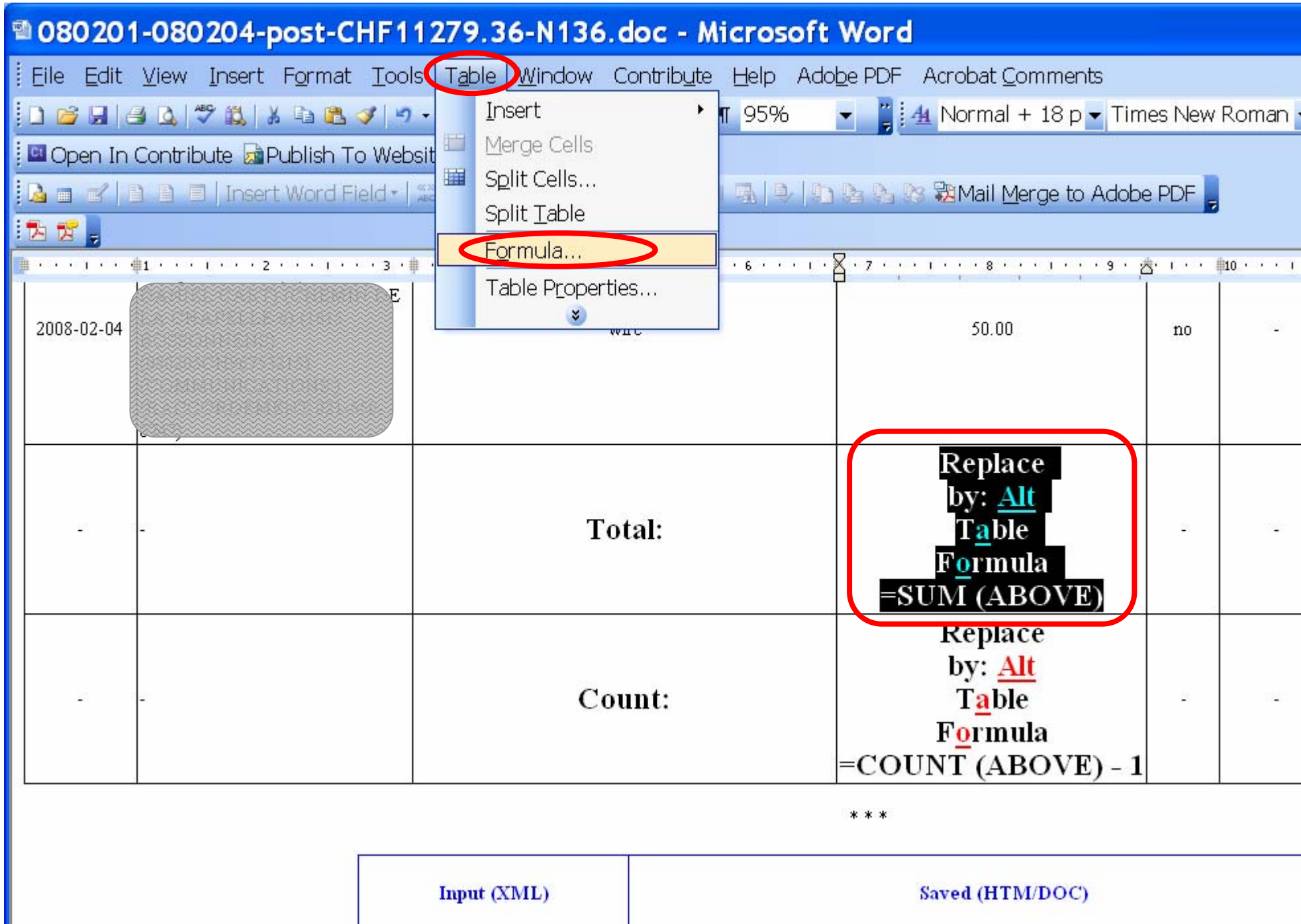
You do not need to provide any arguments, since the script will figure out the range of dates only by analyzing the [tar.gz] files available in the current folder.

The script will create an HTML file containing all transactions of full consecutive days covered by the set of [tar.gz] files available in the working directory.

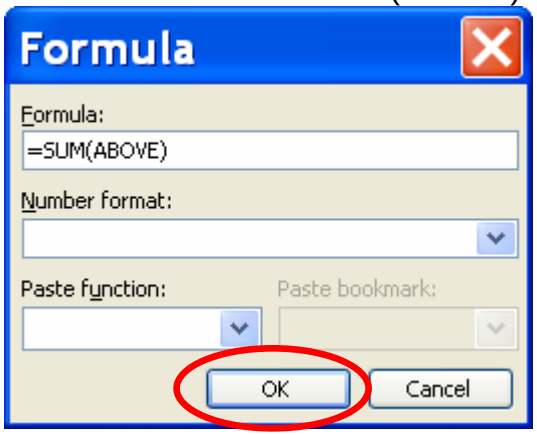


The HTML file must be saved as a DOC file. An empty yymmdd-yymmdd-post-CHFxxxxx.xx-Nxxx.doc file is created. You have to copy paste the content of the HTML file into the DOC file.

Before saving the DOC file, find in the two last rows of the table of payments the cell where you must insert the MS Word formula for displaying the total of received payments:


080201-080204-post-CHF11279.36-N136.doc - Microsoft Word
File Edit View Insert Format Tools Table Window Contribute Help Adobe PDF Acrobat Comments
Insert
Merge Cells
Split Cells...
Split Table
Formula...
Table Properties...
95%
Normal + 18 p
Times New Roman
Open In Contribute
Publish To Websit
Insert Word Field
Mail Merge to Adobe PDF
2008-02-04
50.00
no
Total:
Replace by: Alt Table Formula =SUM (ABOVE)
Count:
Replace by: Alt Table Formula =COUNT (ABOVE) - 1
* * *
Input (XML)
Saved (HTM/DOC)

Choose the formula ‘=SUM(ABOVE)’



The content of the cell will be replaced by the sum of all above listed values. Check that the total is equal to the value of the ‘save_credits’ parameter displayed at the end of the page:

080201-080204-post-CHF11279.36-N136.doc - Microsoft Word

File Edit View Insert Format Tools Table Window Contribute Help Adobe PDF Acrobat Comments

Open In Contribute Publish To Website Post To Blog

Insert Word Field Mail Merge to Adobe PDF

2008-02-04	[redacted] E	wire	50.00	no	-
-	-	**Total:**	**11279.36**	-	-
-	-	**Count:**	**Replace by: Alt Table Formula =COUNT (ABOVE) - 1**	-	-

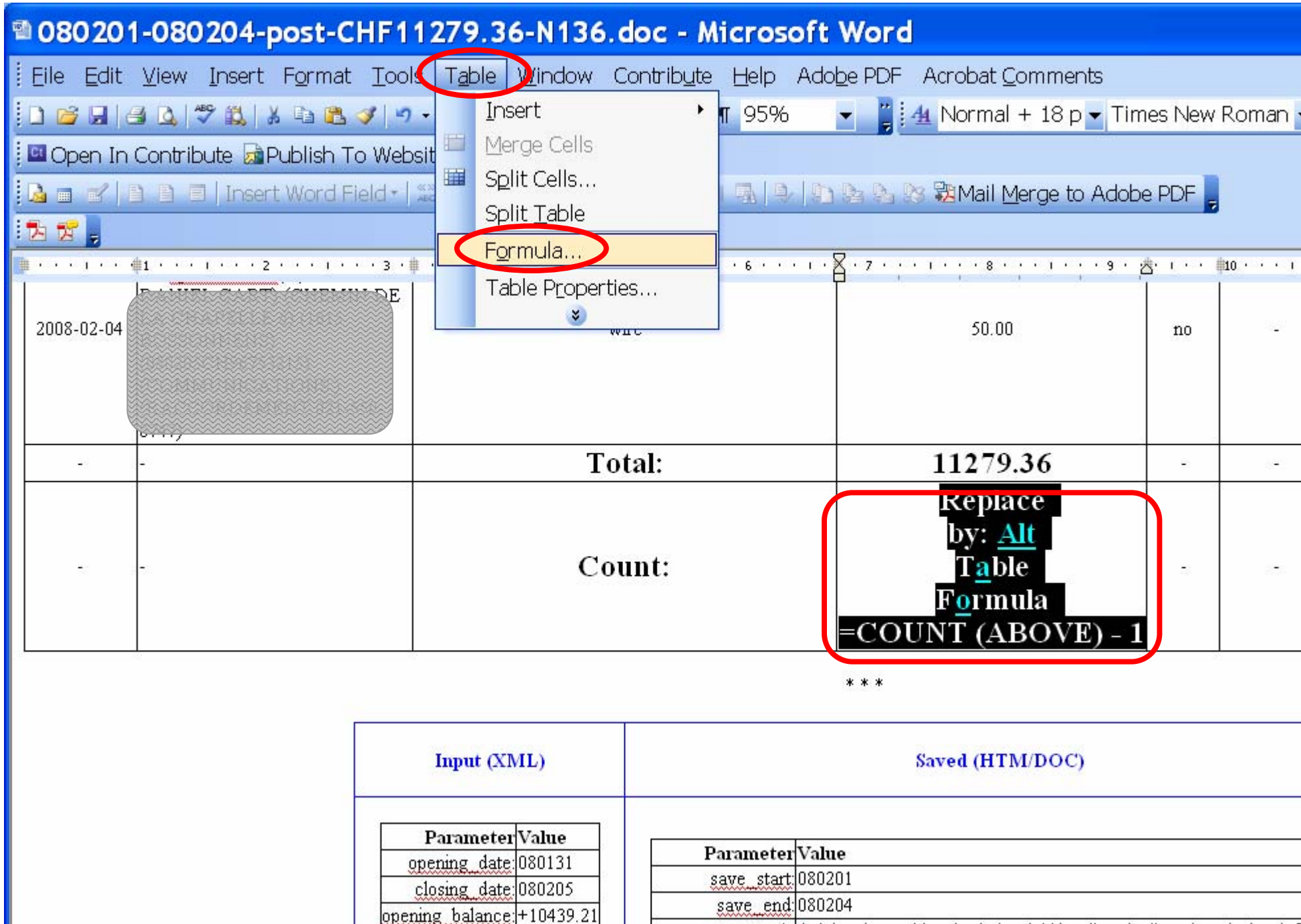
* * *

Input (XML)	Saved (HTM/DOC)

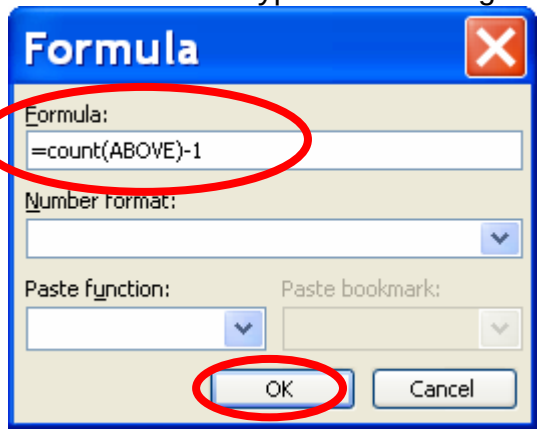
Parameter	Value
opening_date	080131
closing_date	080205
opening_balance	+10439.21

Parameter	Value
save_start	080201
save_end	080204

Now select the content of the cell where you must insert the MS Word formula for displaying the number of received payments:


080201-080204-post-CHF11279.36-N136.doc - Microsoft Word
File Edit View Insert Format Tools Table Window Contribute Help Adobe PDF Acrobat Comments
Insert
Merge Cells
Split Cells...
Split Table
Formula...
Table Properties...
Open In Contribute
Publish To Websit
Insert Word Field
Mail Merge to Adobe PDF
95%
Normal + 18 p
Times New Roman
2008-02-04
50.00
no
Total:
11279.36
Count:
Replace by: Alt Table Formula
=COUNT (ABOVE) - 1
* * *
Input (XML)
Saved (HTM/DOC)
Parameter Value
opening_date: 080131
closing_date: 080205
opening_balance: +10439.21
Parameter Value
save_start: 080201
save_end: 080204

In the formula field type the following ‘=COUNT(ABOVE)-1’



The content of the cell will be replaced by the number of all above listed numerical values minus one (because we do not want to count also the cell with the total amount). Check that the total number computed by the MS Word formula is equal to the value of the ‘save_transactions’ parameter displayed at the end of the page:

080201-080204-post-CHF11279.36-N136.doc - Microsoft Word

File Edit View Insert Format Tools Table Window Contribute Help Adobe PDF Acrobat Comments

Open In Contribute Publish To Website Post To Blog

Insert Word Field Mail Merge to Adobe PDF

2008-02-04	[redacted] 0747)	wire	50.00	no	-	-
-	-	**Total:**	**11279.36**	-	-	-
-	-	**Count:**	**136**	-	-	-

* * *

Input (XML)	Saved (HTM/DOC)

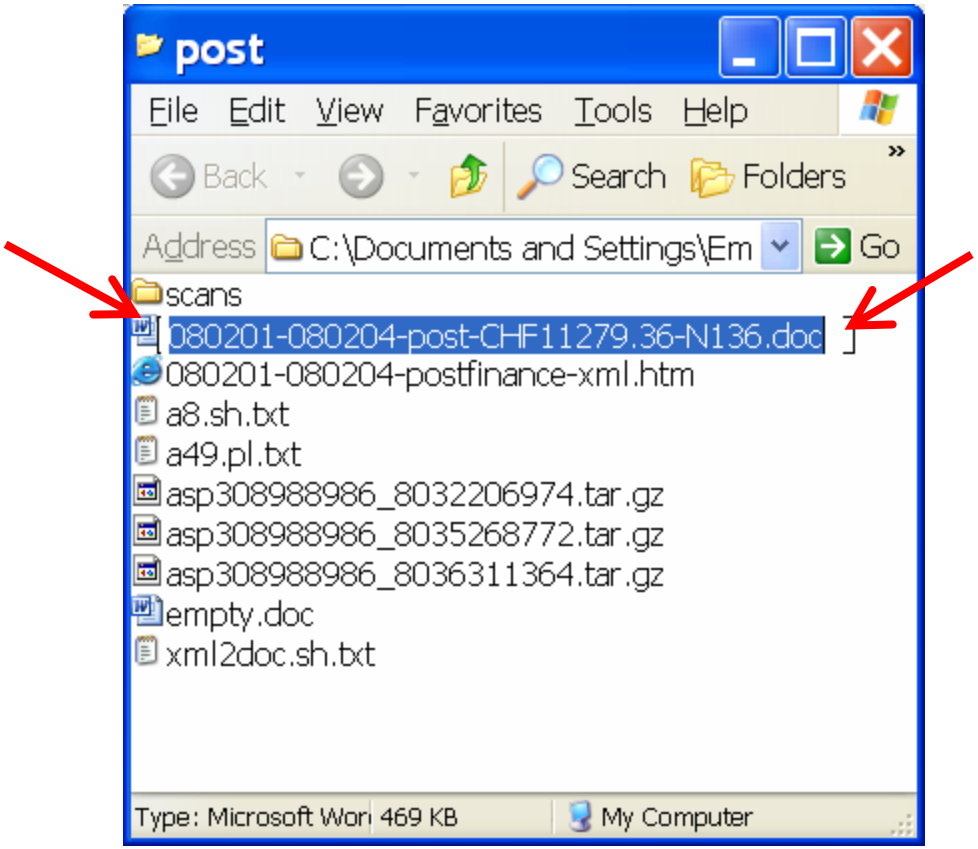
Parameter	Value
opening_date:	080131
closing_date:	080205
opening_balance:	+10439.21
closing_balance:	+12373.71
transactions:	204
credits:	15555.56

Parameter	Value
save_start:	080201
save_end:	080204
watch:	(telekurs\s+multipay|ordre\s+debit\s+direct|online-shopping\s+deferred)
exclude:	yes
save_credits:	**11279.36**

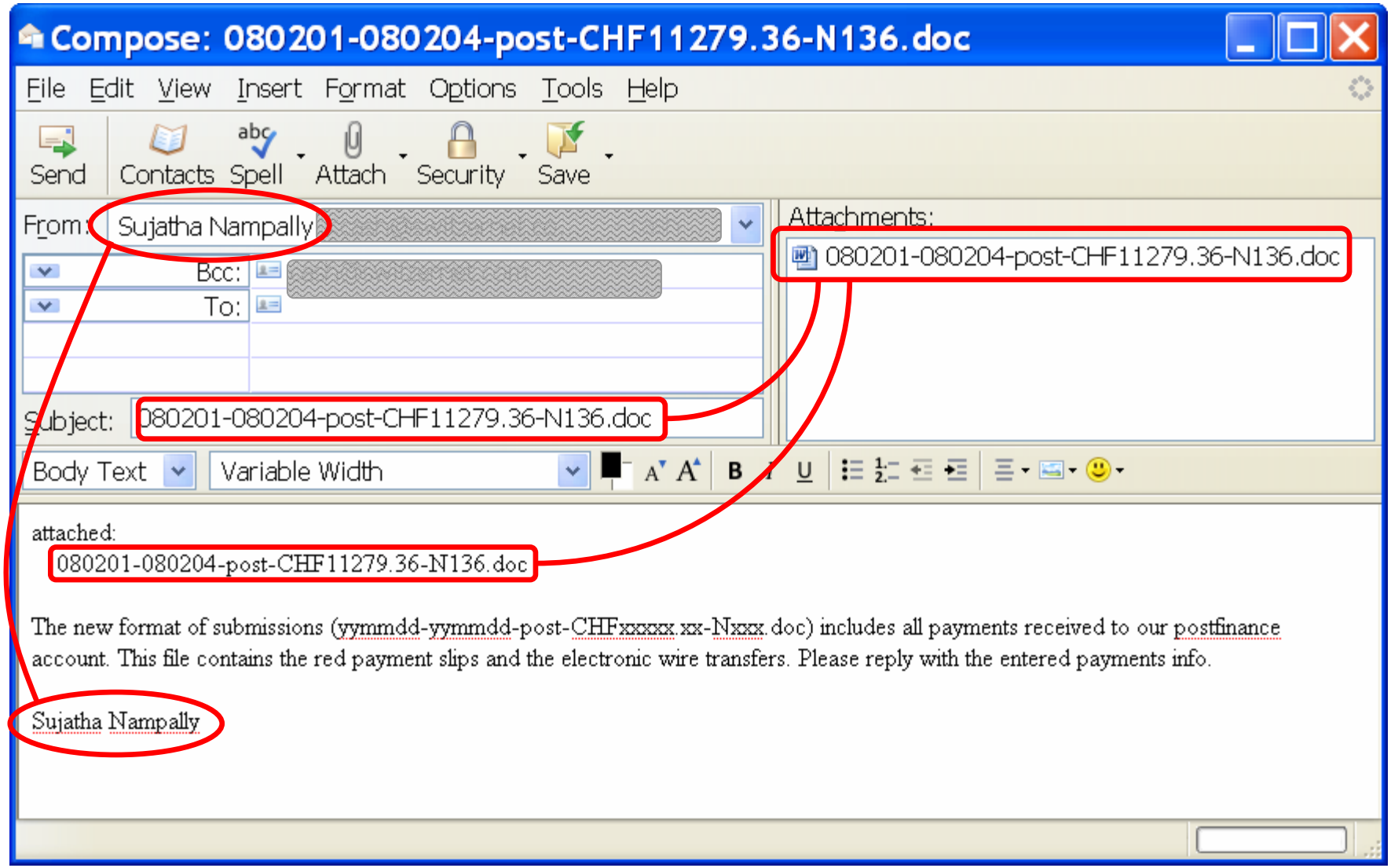
Save the DOC file (Ctrl-S) and exit.

3.2. Submit the file to <payment_processing_queue>

The DOC file with retrieved payments must be submitted to <payment_processing_queue> for further processing. The subject of the email is the name of the DOC file. To avoid errors, do not type the filename in the subject line, but copy-past it. To copy the filename, right click on the file, choose [Rename] in the pop-up menu, press on Ctrl-C to capture the text of the filename, and then click on the background to exit from the Rename mode without modifying the filename:



Paste the filename in the subject line of your e-mail (with Ctrl-V). Do not forget to sign your e-mail both in the body of the text and in the ‘From’ Display Name field (see section 3.2.1):


Compose: 080201-080204-post-CHF11279.36-N136.doc
File Edit View Insert Format Options Tools Help
Send Contacts Spell Attach Security Save
From: Sujatha Nampally
Bcc:
To:
Subject: 080201-080204-post-CHF11279.36-N136.doc
Attachments:
080201-080204-post-CHF11279.36-N136.doc
Body Text
Variable Width
attached:
080201-080204-post-CHF11279.36-N136.doc
The new format of submissions (yymmdd-yymmdd-post-CHFxxxxx.xx-Nxxx.doc) includes all payments received to our postfinance account. This file contains the red payment slips and the electronic wire transfers. Please reply with the entered payments info.
Sujatha Nampally

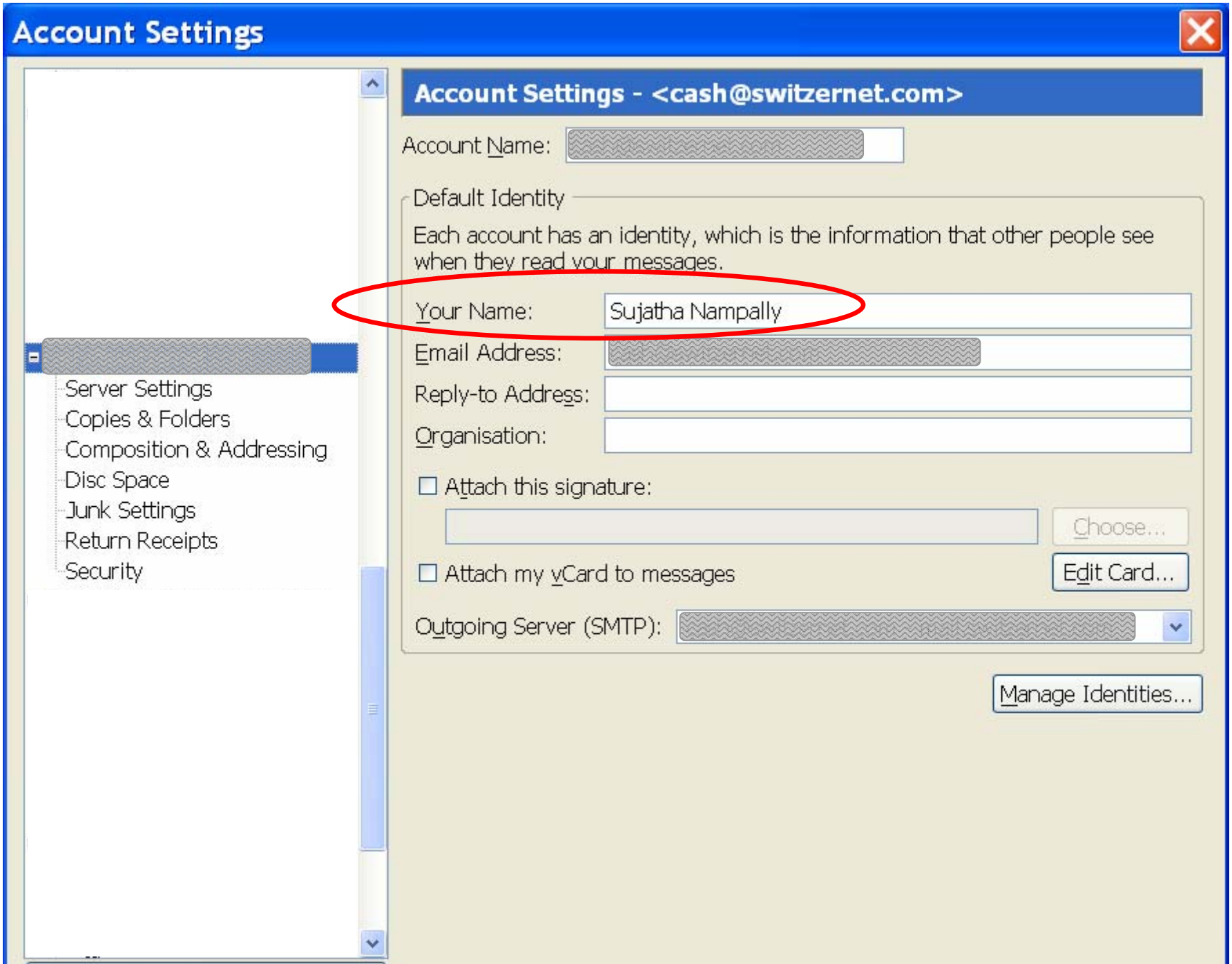
We add the following text in the body of the e-mail:

```
The new format of submissions (yymmdd-yymmdd-post-CHFxxxxx.xx-Nxxx.doc) includes all payments received to
our postfinance account. This file contains the red payment slips and the electronic wire transfers.
Please reply with the entered payments info.
```

3.2.1. Display name of your e-mails

Thee display name of the account for submitting to <payment_processing_queue> must show your name.

Make sure to properly change the display name of the account before submitting the file:


Account Settings
Server Settings
Copies & Folders
Composition & Addressing
Disc Space
Junk Settings
Return Receipts
Security
Account Settings - <cash@switzernet.com>
Account Name:
Default Identity
Each account has an identity, which is the information that other people see when they read your messages.
Your Name:
Sujatha Nampally
Email Address:
Reply-to Address:
Organisation:
Attach this signature:
Choose...
Attach my vCard to messages
Edit Card...
Outgoing Server (SMTP):
Manage Identities...

3.3. Moving from the old style (separate submissions of electronic payments and of red payment slips) to the new style (submissions combining the both types of payments)

Before switching to the new style of submissions (according to this document) the postfinance payments are divided and are submitted in two separate groups: the electronic payments (excel files) and the red payment slips (doc files).

3.3.1. Old style electronic payment submissions

The electronic payments of postfinance are currently retrieved and submitted by either of the two following procedures.

The list of postfinance payments can be retrieved into excel files by copy-pasting the data from the HTML pages [http://switzernet.com/company/080121-howto-retrieve-postfinance-payments/]. The filenames of these excel files look as follows:
`080112-080120-postfinance-credits.xls`

The list of postfinance payments can be retrieved into excel files using e-finance Java program [http://switzernet.com/company/080130-postfinance-java-excel/]. The filenames of these excel files look as follows:
`080221-080225-efinance-java.xls`

3.3.2. Old style red payment slip submissions

The current procedure retrieves the list of red slip payments into DOC files [http://switzernet.com/company/080118-red-payment-slips-doc-table/]. The filenames of these DOC files look as follows:
`080221-080225-red-slips.doc`

3.3.3. Transition to the new style submissions containing both types of postfinance payments (red slips and electronic wires) in a single file

When switching to the new style of submissions, we must make sure that we do not have gaps or overlaps with the old two flows of submissions (electronic wires and red slips).

Choose a proper date for starting new submissions. Find the most recent submission of any of the old two types (sections 3.3.1 and 3.3.2). The very first XML based submission must follow the most recent submission of any of the two old types.

If the old style submissions (electronic wires and red slips) both stopped at the same date, then there are nothing more to do, i.e. the new style is applied from the following day.

If one of the two old style flows is late, it must be continued until it gets aligned with the other one. Both flows stop aligned at the same day leaving place for new submissions from the following day.

4. Viewing the XML files and reading the XML source

From this section on, the document contains details related to the format of XML files and to the scripts processing the XML files. The user retrieving and submitting the payments to <payment_processing_queue> does not need to read the rest of this document.

The rest of this document is required for any modifications in the processing scripts. Such modifications can be required in case postfinance changes the format of its XML files or in case we encounter new elements not considered in format layouts.

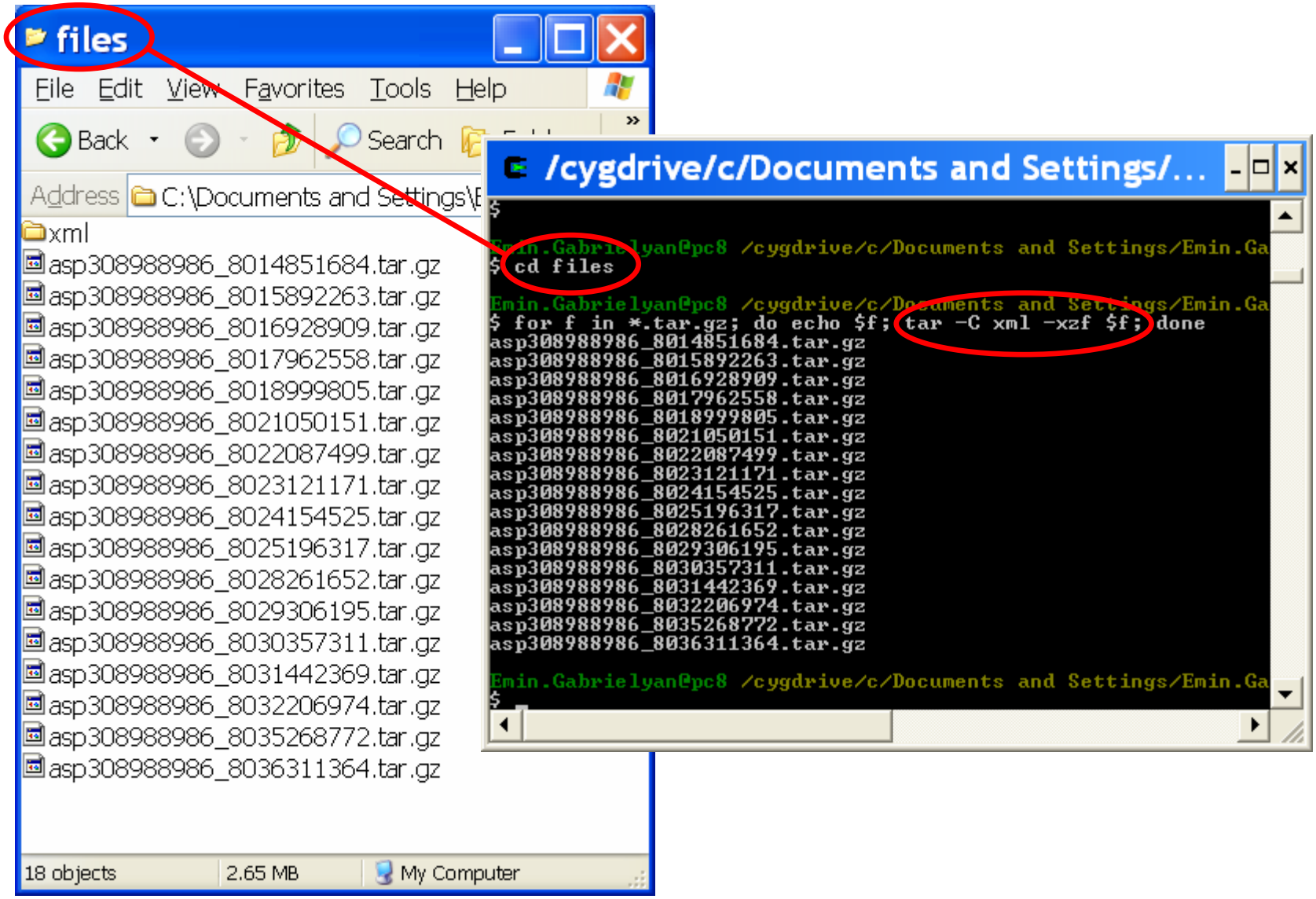
4.1. Extracting the XML files

The downloaded [tar.gz] files can be decompressed with tar program. The following command decompresses file asp308988986_8014851684.tar.gz into folder ‘xml’:

```
$ tar -C xml -xvzf asp308988986_8014851684.tar.gz
PFLogoI.gif
PFLogoD.gif
PFLogoE.gif
PFLogoF.gif
```

```
pixel.gif
lupe.gif
acc_200.dtd
acc_200.xsl
acc_200_f_8014851687.xml
20080110102701000100050.png
2008011028800l000100075.png
```

You can download and decompress several files into the same folder. The files with identical names will be overwritten without any damage, since their contents are the same (these are the DTD, XSL, and logo image files):


files
File Edit View Favorites Tools Help
Back Search
Address C:\Documents and Settings\
xml
asp308988986_8014851684.tar.gz
asp308988986_8015892263.tar.gz
asp308988986_8016928909.tar.gz
asp308988986_8017962558.tar.gz
asp308988986_8018999805.tar.gz
asp308988986_8021050151.tar.gz
asp308988986_8022087499.tar.gz
asp308988986_8023121171.tar.gz
asp308988986_8024154525.tar.gz
asp308988986_8025196317.tar.gz
asp308988986_8028261652.tar.gz
asp308988986_8029306195.tar.gz
asp308988986_8030357311.tar.gz
asp308988986_8031442369.tar.gz
asp308988986_8032206974.tar.gz
asp308988986_8035268772.tar.gz
asp308988986_8036311364.tar.gz
18 objects
2.65 MB
My Computer
/cygdrive/c/Documents and Settings/...
$ cd files
$ for f in *.tar.gz; do echo $f; tar -C xml -xzf $f; done
asp308988986_8014851684.tar.gz
asp308988986_8015892263.tar.gz
asp308988986_8016928909.tar.gz
asp308988986_8017962558.tar.gz
asp308988986_8018999805.tar.gz
asp308988986_8021050151.tar.gz
asp308988986_8022087499.tar.gz
asp308988986_8023121171.tar.gz
asp308988986_8024154525.tar.gz
asp308988986_8025196317.tar.gz
asp308988986_8028261652.tar.gz
asp308988986_8029306195.tar.gz
asp308988986_8030357311.tar.gz
asp308988986_8031442369.tar.gz
asp308988986_8032206974.tar.gz
asp308988986_8035268772.tar.gz
asp308988986_8036311364.tar.gz

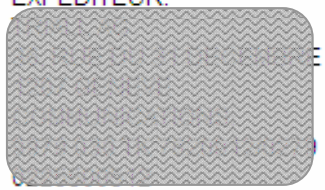
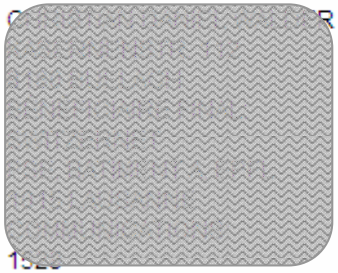
17 XML files extracted and used during our examples are listed below. The can be viewed by a browser:
acc_200_f_8014851687.xml, acc_200_f_8015892269.xml, acc_200_f_8016928919.xml, acc_200_f_8017962574.xml, acc_200_f_8018999822.xml, acc_200_f_8021050156.xml, acc_200_f_8022087517.xml, acc_200_f_8023121191.xml, acc_200_f_8024154537.xml, acc_200_f_8025196335.xml, acc_200_f_8028261714.xml, acc_200_f_8029306211.xml, acc_200_f_8030357335.xml, acc_200_f_8031442397.xml, acc_200_f_8032206992.xml, acc_200_f_8035268790.xml, acc_200_f_8036311383.xml

4.2. Postfinance XML files viewed by a browser

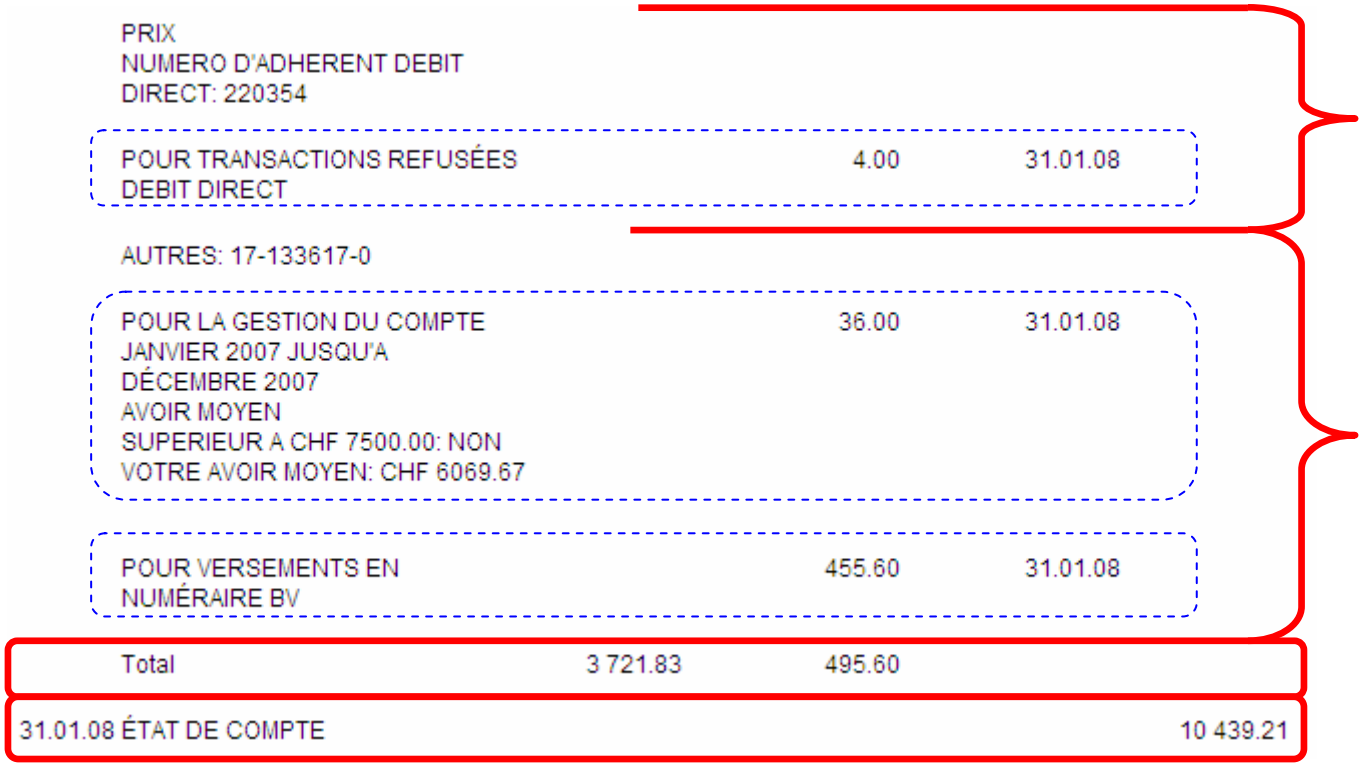
The XML files can be visualized by a browser, provided that the XSL style sheet [xsl] and the DTD type definition [dtd] files are also available. The screenshots below show the statement entries of the XML file as viewed by a browser. Each statement entry is enclosed in a red frame or is marked by curled parenthesis.

The entries enclosed in a red frame are the opening balance, the bank record of totals, and the closing balance.

The conventional credit or debit transaction entries contain one credit or debit amount. Complex transaction entries may have multiple sub entries shown in blue dashed frames:

Date	Texte	Crédit	Débit	Valeur	Solde
30.01.08	ÉTAT DE COMPTE				7 212.98
	VIREMENT DU COMPTE 80-2-2 UBS AG ZÜRICH EXPÉDITEUR: 	33.05		30.01.08	
	VIREMENT DU COMPTE 80-151-4 ZÜRCHER KANTONALBANK ZÜRICH EXPÉDITEUR: HERRN 	150.00		30.01.08	

…



4.3. The XML source format

The XML format of the statement files is preliminarily discussed in [switzernet.com/company/080129-postfinance-acc-xml-file-format/, 4z.com, unappel.ch]. The referred document provides a Perl script [a10.pl.txt] converting the text of an XML source file into a readable layout (without changing the XML properties of the file). We identified that the individual statement entries are represented by SG4 XML elements. For different types of statement entries, such as opening balance, credits, debits, totals, and closing balance, we showed the differences in the structures of the SG4 elements.

The SG4 element is composed of LIN, FTX, SG5, and SG6 elements. LIN stands for ‘line item’, FTX stands for ‘free text’, SG5 represents the account balance, and SG6 represents transactions. The structure of SG4 record changes, depending on the type of the statement entry. The table below shows the structures of all encountered types of SG4 records. Empty cells signify the absence of the element (shown at the top of the column) for the given type of SG4 entry (entitled at the right end of the row).

The type of the SG4 record	<SG4> <LIN>...</LIN> <FTX>...</FTX> <SG5>...</SG5> <SG6>...</SG6> <SG6>...</SG6> </SG4>				
	<LIN>	<FTX>	<SG5>	<SG6>	<SG6>
Opening balance [txt] (see section 8.4.1)	<LIN> <PF:D_0805 xmlns:PF="http://www.post.ch/xml" Value="**LST**"></PF:D_0805> </LIN>	<FTX> <D_4451 Value="ABN"></D_4451> <D_4453 Value="0"></D_4453> <C108> <D_4440>**ÉTAT DE COMPTE**</D_4440> </C108> </FTX>	<SG5> <MOA> <C516> <D_5025 Value="**315**"></D_5025> <D_5004>**1560.00**</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">**+**</PF:D_5003> </MOA> <DTM> <C507> <D_2005 Value="202"></D_2005> <D_2380>**20080118**</D_2380>		

			</C507> </DTM> </SG5>		
Closing balance [txt] (see section 8.4.1)	<LIN> <PF:D_0805 xmlns:PF="http://www.post.ch/xml" Value="**LEN**"></PF:D_0805> </LIN>	<FTX> <D_4451 Value="ABN"></D_4451> <D_4453 Value="0"></D_4453> <C108> <D_4440>**ÉTAT DE COMPTE**</D_4440> </C108> </FTX>	<SG5> <MOA> <C516> <D_5025 Value="**343**"></D_5025> <D_5004>**10191.53**</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">**+**</PF:D_5003> </MOA> <DTM> <C507> <D_2005 Value="202"></D_2005> <D_2380>**20080125**</D_2380> </C507> </DTM> </SG5>		
Closing balance neg	<LIN> <PF:D_0805 xmlns:PF="http://www.post.ch/xml" Value="**LEN**"></PF:D_0805>	<FTX> <D_4451 Value="ABN"></D_4451> <D_4453 Value="0">	<SG5> <MOA> <C516> <D_5025 Value="**343**"></D_5025> <D_5004>**938.70**</D_5004		

ative [txt]	</LIN>	</D_4453> <C108> <D_4440>**ÉTAT DE COMPTE**</D_4440> </C108> </FTX>	> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">- </PF:D_5003> </MOA> <DTM> <C507> <D_2005 Value="202"></D_2005> <D_2380>**20080201**</D_2380> </C507> </DTM> </SG5>		
Credit [txt]	<LIN> <PF:D_0805 xmlns:PF="http://www.post.ch/xml" Value="**LNE**"></PF:D_0805> </LIN>		<SG5> <MOA> <C516> <D_5025 Value="**15**"></D_5025> <D_5004>1590.80</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">+</PF:D_5003> </MOA> <DTM> <C507> <D_2005	<SG6> <RFF> <C506> <D_1153 Value="ACD"></D_1153> <D_1154>02</D_1154> </C506> </RFF> <DTM> <C507> <D_2005 Value="209"></D_2005> <D_2380>20080118</D_2380> </C507> </DTM>	

			Value="202"></D_2005> <D_2380>**20080118**</D_2380> </C507> </DTM> </SG5>	<MOA> <C516> <D_5025 Value="**210**"></D_5025> <D_5004>**30.80**</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">**+**</PF:D_5003> </MOA> <FTX> <D_4451 Value="ABN"></D_4451> <D_4453 Value="0"></D_4453> <C108> <D_4440>**VIREMENT DU COMPTE 80-2-2**</D_4440> <D_4440>**UBS AG**</D_4440> <D_4440>**ZÜRICH**</D_4440> <D_4440>**EXPÉDITEUR:**</D_4440> <D_4440>**XXXX COMPUTER**</D_4440> <D_4440>**SOLUTIONS SA**</D_4440> <D_4440>**XXXX**</D_4440>	

				<D_4440>**1214 VERNIER**</D_4440> <D_4440>**COMMUNICATIONS:**</D_4440> <D_4440>**XXXX**</D_4440> <D_4440>**0225500XXX**</D_4440> </C108> </FTX> </SG6>	
Redslip credit [txt]	<LIN> <PF:D_0805 xmlns:PF="http://www.post.ch/xml" Value="**LNE**"></PF:D_0805> </LIN>		<SG5> <MOA> <C516> <D_5025 Value="**15**"></D_5025> <D_5004>3305.36</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">+</PF:D_5003> </MOA> <DTM> <C507> <D_2005 Value="202"></D_2005> <D_2380>**20080121**</D_2380>	<SG6> <RFF> <C506> <D_1153 Value="ACD"></D_1153> <D_1154>01</D_1154> </C506> </RFF> <RFF> <C506> <D_1153 Value="**ZZZ**"></D_1153> <D_1154>**20080117115105000200015**</D_1154> </C506> </RFF> <DTM> <C507> <D_2005 Value="209"></D_2005>	

			</C507> </DTM> </SG5>	 <D_2380>20080121</D_2380> </C507> </DTM> <MOA> <C516> <D_5025 Value="**210**"></D_5025> <D_5004>**84.70**</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">**+**</PF:D_5003> </MOA> <FTX> <D_4451 Value="ABN"></D_4451> <D_4453 Value="0"></D_4453> <C108> <D_4440>**BULLETIN DE VERSEMENT**</D_4440> </C108> </FTX> </SG6>	
Debit [txt]	<LIN> <PF:D_0805 xmlns:PF="http://www.post.ch/xml" Value="**LNE**"></P		<SG5> <MOA> <C516> <D_5025 Value="**15**"></D_5025>	<SG6> <RFF> <C506> <D_1153 Value="ACD"></D_1153> <D_1154>00</D_1154>	

	F:D_0805> </LIN>		<D_5004>3411.05</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">+</PF:D_5003> </MOA> <DTM> <C507> <D_2005 Value="202"></D_2005> <D_2380>**20080123**</D_2380> </C507> </DTM> </SG5>	</C506> </RFF> <DTM> <C507> <D_2005 Value="209"></D_2005> <D_2380>20080123</D_2380> </C507> </DTM> <MOA> <C516> <D_5025 Value="**211**"></D_5025> <D_5004>**10000.00**</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">**+**</PF:D_5003> </MOA> <FTX> <D_4451 Value="ABN"></D_4451> <D_4453 Value="0"></D_4453> <C108> <D_4440>**E-FINANCE 80-2-2**</D_4440> <D_4440>**UBS AG**</D_4440>	

				<D_4440>**CH900024324326249801D**</D_4440> <D_4440>**SWITZERNET (UBS ACC)**</D_4440> <D_4440>**ACC. 243-262498.01D**</D_4440> <D_4440>**1015 LAUSANNE**</D_4440> </C108> </FTX> </SG6>	
Debit LNS [txt]	<LIN> <PF:D_0805 xmlns:PF="http://www.post.ch/xml" Value="**LNS**"></PF:D_0805> </LIN>	<FTX> <D_4451 Value="ABN"></D_4451> <D_4453 Value="5"></D_4453> <C108> <D_4440>**PRIX**</D_4440> <D_4440>**NUMERO D'ADHERENT DEBIT DIRECT: 220354**</D_4440>	<SG5> <MOA> <C516> <D_5025 Value="**15**"></D_5025> <D_5004>10894.81</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">+</PF:D_5003> </MOA> <DTM> <C507> <D_2005 Value="202"></D_2005> <D_2380>**20080131**</D_2380> </C507>	<SG6> <RFF> <C506> <D_1153 Value="ACD"></D_1153> <D_1154>06</D_1154> </C506> </RFF> <DTM> <C507> <D_2005 Value="209"></D_2005> <D_2380>20080131</D_2380> </C507> </DTM> <MOA> <C516> <D_5025 Value="**211**"></D_5025>	

		</C108> </FTX>	</DTM> </SG5>	<D_5004>**4.00**</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">**+**</PF:D_5003> </MOA> <FTX> <D_4451 Value="ABN"></D_4451> <D_4453 Value="0"></D_4453> <C108> <D_4440>**POUR TRANSACTIONS REFUSÉES DEBIT DIRECT**</D_4440> </C108> </FTX> </SG6>	
Debit LNS with multiple entries [txt]	<LIN> <PF:D_0805 xmlns:PF="http://www.post.ch/xml" Value="**LNS**"></PF:D_0805> </LIN>	<FTX> <D_4451 Value="ABN"></D_4451> <D_4453 Value="5"></D_4453> <C108> <D_4440>**AUTRES: 17-133617-0**</D_4440> </C108>	<SG5> <MOA> <C516> <D_5025 Value="**15**"></D_5025> <D_5004>10439.21</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">+</PF:D_5003> </MOA> <DTM>	<SG6> <RFF> <C506> <D_1153 Value="ACD"></D_1153> <D_1154>06</D_1154> </C506> </RFF> <DTM> <C507> <D_2005 Value="209"></D_2005> <D_2380>20080131</D_2380>	<SG6> <RFF> <C506> <D_1153 Value="ACD"></D_1153> <D_1154>06</D_1154> </C506> </RFF> <DTM> <C507> <D_2005 Value="209"></D_2005> <D_2380>20080131</D_2380>

		</FTX>	<C507> <D_2005 Value="202"></D_2005> <D_2380>**20080131**</D_2380> </C507> </DTM> </SG5>	</C507> </DTM> <MOA> <C516> <D_5025 Value="**211**"></D_5025> <D_5004>**36.00**</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">**+**</PF:D_5003> </MOA> <FTX> <D_4451 Value="ABN"></D_4451> <D_4453 Value="0"></D_4453> <C108> <D_4440>**POUR LA GESTION DU COMPTE**</D_4440> <D_4440>**JANVIER 2007 JUSQU'A DÉCEMBRE 2007**</D_4440> <D_4440>**AVOIR MOYEN**</D_4440> <D_4440>**SUPERIEUR A CHF 7500.00: NON**</D_4440> <D_4440>**VOTRE AVOIR MOYEN: CHF 6069.67**</D_4440>	</C507> </DTM> <MOA> <C516> <D_5025 Value="**211**"></D_5025> <D_5004>**455.60**</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">**+**</PF:D_5003> </MOA> <FTX> <D_4451 Value="ABN"></D_4451> <D_4453 Value="0"></D_4453> <C108> <D_4440>**POUR VERSEMENTS EN NUMÉRAIRE BV**</D_4440> </C108> </FTX> </SG6>

				<D_4440></D_4440> </C108> </FTX> </SG6>	
Totals [txt]	<LIN> <PF:D_0805 xmlns:PF="http://www.post.ch/xml" Value="**LTI**"></PF:D_0805> </LIN>	<FTX> <D_4451 Value="ABN"></D_4451> <D_4453 Value="0"></D_4453> <C108> <D_4440>**Total**</D_4440> </C108> </FTX>		<SG6> <MOA> <C516> <D_5025 Value="**210**"></D_5025> <D_5004>**18831.53**</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">**+**</PF:D_5003> </MOA> </SG6> <SG6>	<SG6> <MOA> <C516> <D_5025 Value="**211**"></D_5025> <D_5004>**10200.00**</D_5004> </C516> <PF:D_5003 xmlns:PF="http://www.post.ch/xml">**+**</PF:D_5003> </MOA> </SG6>

SG4 elements with LNS values were not discussed in [switzernet.com/company/080129-postfinance-acc-xml-file-format/, 4z.com, unappel.ch]. These elements probably correspond to transactions directly carried out with postfinance. The LNS types of entries contain multiple sub entries.

5. Processing the entries with a Perl script

In this section we present a Perl script which is able to read and analyze the XML files, to retrieve the values of the SG4 statement entries, and to build HTML tables with the retrieved values. A usage of the Perl script is shown at the end of this section. The table below presents the script, commented in details:

Perl code	Comments
#!/bin/perl $"=" "; $_="@ARGV";	Merging the arguments of the script into a single space-separated string.
if(!/^\s*xml=([^\s]+)(?:\s+save=(\d{6})- (\d{6})\s+htm=([^\s]+)(?:\s+watch=([^\s]+)\s+exclude=(yes|no))?)?\s*$/) { print STDERR "wrong arguments: "; print STDERR; print STDERR "\narguments: xml=<filename> [save=yymmdd-YYMMDD htm=<filename> [watch=RE exclude=[yes|no]]]\n"; print STDERR "Example: xml=in.xml save=080130-080131 htm=out.htm watch=\"(telekurs\\s+multipay|ordre\\s+debit\\s+d irect|online-shopping\\s+deferred)\" exclude=no\n"; exit 1; }	The if-statement checks the format of arguments. The regular expression checks for an obligatory occurrence of the “xml=” argument and for a possible occurrence of the pair of “save=” and “htm=” arguments, possibly followed by a pair of “watch=” and “exclude=” arguments.
$xmlfname=$1; $save_start=$2; $save_end=$3; $htmfname=$4; $watch=$5; $exclude=$6;	The filenames and date values were captured by the matching operator of the if-statement. The values extracted from the arguments’ line are assigned to corresponding variables.
if(defined($save_start) && defined($save_end)) { $save=1;	The script determines whether an output must be generated or only

`  if($save_start gt $save_end)` `  {` `  print STDERR "start date $save_start is greater than end date $save_end\n";` `  exit 1;` `  }` `}`	the scanning and checking of the input XML file is required. If the output must be generated, the text comparison in the if-statement is for making sure that the end date is greater than or equal to the start date. Note that the dates have the 6-digits format. The format is already controlled above while checking the format of arguments.
`$empty="-";`	The string that will appear in empty cells of the output HTML table.
`if(!open fh,$xmlfname)` `{` `  print STDERR "cannot open file $xmlfname\n";` `  exit 1;` `}` `@lines=<fh>;` `close fh;`	If the XML input file can be opened for reading, its lines are loaded into array @lines and the file is closed.
`$"="";` `$_="@lines";` `s/[\r\n]//sg;`	All lines are merged into the default string $_.

	Additional separators between lines are not inserted ($"=""). The carriage return and line feed symbols, which are at the ends of the lines, are removed.
@rows=split/<SG4>/;	Now when all lines of the input XML file are merged into a single string without any separators, we split that string using "<SG4>" as a separator. The string "<SG4>" is the opening tag of statement entries.
$s="\\s*"; $tag="<[^>]*>[^<]*</[^>]*>";	The Regular Expressions (RE) can be stored in strings for being used later in matching operators. Inside of the slashes of a match or a substitution variables are interpolated. The first string stores an RE for any

	number of occurrences of a whitespace character. The second string is an RE for any valid XML element including its head tag, body, and the tail tag. The backslash is used as usually when occurs in RE openly, without substitutions. However the backslash must be protected by another backslash, when attempting to store the RE in a string (the string will in fact store only one occurrence).
$sign="<PF:D_5003[^>]*>([+-])</PF:D_5003>";	We define Regular Expression (RE) for matching <PF:D_5003> tags which are storing the sign of the amount. This string helps to create a more complex RE for matching the <MOA> tag.

	<MOA> tags are dedicated for all types of amounts (debits, credits, balances, and totals). MOA stands for monetary amount.
$lin="<LIN>$s<PF:D_0805[^>]*Value=\"(\\w+)\">$s</PF:D_0805>$s</LIN>";	The <LIN> tag is at the beginning of each statement entry. See samples in section 4.3. LIN stands for line item. Via this RE we capture a value, which will help us to determine whether the entry, containing the matched <LIN> tag, is an opening or closing balance, a credit or a debit note, or a record with totals. The <LIN> tag in the Document Type Definition file[dtd.txt] is defined as follows: <!ELEMENT LIN (PF:D_0805)>
$moa="<MOA>$s<C516>$s<D_5025 Value=\"(\\d+)\"></D_5025>$s<D_5004>([\\d\\.]+)</D_5004>$s</C516>$s$sign$s</MOA>";	Here we build an RE for matching all <MOA>

	tags (monetary amount). This RE captures three values (into special variables, $1, $2 ...). The first value is the type of the amount. For example, if this value is equal to 210 then the amount is a credit, if it is equal to 211 then the amount is a debit (see the samples in section 4.3). The second value captured by RE is the amount itself. The third value is captured by the part of RE stored in string $sign (defined above). The document type definition (DTD) file [txt] describes the MOA and its nested C516 tags as follows: <!ELEMENT MOA (C516,

	PF:D_5003?)> <!ELEMENT C516 (D_5025, D_5004, D_6345?)>
$dtm="<DTM>$s<C507>$s$tag$s<D_2380>(**\\d+**)</D_2380>$s</C507>$s</DTM>";	RE expression $dtm matches all <DTM> tags for date/time/period. The date value is stored in YYYYMMDD format (see the samples in section 4.3). This RE captures a single value of the date. The <DTM> tag's hierarchy in the Document Type Definition file[dtd.txt] is as follows: <!ELEMENT DTM (C507)> <!ELEMENT C507 (D_2005, D_2380)>
$ftx="<FTX>$stagstags<C108>$s(**(?:<D_4440>[^<]*</D_4440>$s)+**)</C108>$s</FTX>";	The RE expression stored in $ftx string matches the FTX elements. FTX stands for free text. <FTX> tags contain all sorts of information

	messages, e.g. about the party issuing the payment. The $ftx RE captures a set of <D_4440> tags containing info lines. In Perl, (?:regexp) denotes non-capturing groupings [perlretut]. The <FTX> tag's hierarchy in the DTD file[dtd.txt] is as follows: <!ELEMENT FTX (D_4451, D_4453?, C108)> <!ELEMENT C108 (D_4440+)> <!ELEMENT D_4440 (#PCDATA)>
$rff="<RFF>$s<C506>$s<D_1153 Value=\"(\\w+)\"></D_1153>$s<D_1154>(\\d+)</D_1154>$s</C506>$s</RFF>";	The string $rff contains the RE for matching the RFF elements. RFF stands for reference. The $rff RE captures two values. If the first captured value is "ZZZ", then we are dealing with a red payment slip transaction. In such

	case the second captured value is a reference number of the PNG file containing the scan of the red-payment slip [example]. The <RFF> tag in the DTD file[dtd.txt] is defined as follows: <!ELEMENT RFF (C506)> <!ELEMENT C506 (D_1153, D_1154)>
#<!ELEMENT SG5 (MOA?, DTM?)> $sg5="<SG5>$smoasdtms</SG5>";	A statement entry (the SG4 element) may include zero or one <SG5> element. The <SG5> element provides the account balance (when the credit or debit associated with the same entry is accounted). SG5 stands for Segment Group 5. According to UN/EDIFACT it contains the account related details such as balances, account number and frequency of statement presentation [FINSTAD.96A].

	RE $moa is for matching the amount of the account balance (defined above). RE $dtm is for matching the date of the entry (defined above). RE $sg5 captures 4 values: first three are captured by RE $moa and the last one, the date, by RE $dtm.
#<!ELEMENT SG6 (RFF*, DTM*, MOA, FTX*)> $sg6="$s(?:rffs)*(?:dtms)?moas(?:ftxs)?</SG6>";	A statement entry (the SG4 element) may include several SG6 elements. SG6 stands for Segment Group 6. It contains the single items as they were advised to the customer by a debit/credit advice [FINSTA D.96A]. RE $sg6 matches all encountered SG6 elements. <SG6> element may contain several <RFF> elements. In case of a red payment slip payment, the last

	<RFF> element contains the filename info of the scan of the red payment slip [scan example]. For credit and debit entries the <DTM> element of the <SG6> element (<SG6>➔<DTM>) contains the value date of the amount. The date of the transaction must be retrieved from the <DTM> element of the <SG5> element (<SG5>➔<DTM>) rather than from the <DTM> of <SG6> (<SG6>➔<DTM>). The <MOA> of <SG6> element (<SG6>➔<MOA>) contains the amount of the transaction while the <SG5> element's <MOA> (<SG5>➔<MOA>) was the amount of the account's balance. RE $sg6 captures 7 values. The two first values are captured by

	$rff (only the values of the last occurrence of <RFF> tag are captured). The third one is captured by $dtm. RE $moa captures the 4th, 5th, and 6th values. Finally, RE $ftx captures the 7th value. Note that RE $sg6 does not match the full tag. The opening header string “<SG6>” of the tag is not included in the RE. This is because below we split a string containing consecutive SG6 elements using as a separator the string “<SG6>”. The header string “<SG6>” will be eaten by the split operation and RE must match only the decapitated body and the tail without the header. For matching the full string of the <SG6> tag the RE $sg6 must be preceded by the header string

	“<SG6>”, e.g: /<SG6>$sg6/
#<!ELEMENT SG4 (LIN, FTX*, SG5?, SG6*)> $sg4="$slins(?:ftxs)?(?:$sg5$s)?(?:<SG6>$sg6$s)*</SG4>";	RE $sg4 matches with a complete statement entry SG4. Its name stands for Segment Group 4 [FINSTAD.96A]. RE does not contain the opening <SG4> tag, since the XML source is split using string “<SG4>” as a separator. RE $sg4 captures 13 values. The first 6 values contain the type of the entry of the <LIN> element, an info string from <FTX> element, and the 4 values of the RE $sg5. The last 7 values are captured from the last occurrence of the <SG6> element. To capture all occurrences of <SG6>, the current instance of the <SG4> element must be further split

	using string “<SG6>” as a separator.
$verbose=0; $verbose*=$save; if($verbose) { printf STDERR "%8s %23s %3s %8s %25s\n","Date","Red slip scan","+/-","CHF","Text"; printf STDERR "%8s %23s %3s %8s %25s\n","----","------------","---","---","----"; }	If the verbose mode is requested the script will print on STDERR all transactions of the file, line by line. Here we print the header line with titles of fields.
if($save) { if(!open htm,">".$htmfname) { print STDERR "cannot open file $htmfname\n"; exit 1; } print htm "<html>\r\n"; print htm "<body>\r\n"; print htm "<table border=1>\r\n"; print htm "<tr>\r\n"; print htm " <th width=90>Date: YYYY-MM-DD</th>\r\n"; print htm " <th>Information</th>\r\n"; print htm " <th>Scan</th>\r\n"; print htm " <th>CHF</th>\r\n"; print htm " <th width=50>Billing Entered</th>\r\n"; print htm " <th width=90>Entered Date: YYYY-MM-DD (Alt-<u>I</u>nsert-<u>T</u>ime-Enter)</th>\r\n"; print htm " <th>Customer Name</th>\r\n"; print htm " <th width=110>Phone (account) number: 02x-550-xxxx</th>\r\n"; print htm " <th>Remarks</th>\r\n";	If the save option is set, we create the html output file. It contains the beginning of the table of transactions. The table will contain only the credits (i.e. the payments received from customers). Its first field is the date (<SG5>➔<DTM>), the second field is the information text, the third field is the scan of the red payment slip (if any), the forth field is the amount (<SG6>➔<MOA>).

`  print htm "</tr>\r\n";` `}`	
`foreach $_ (@rows)` `{` `  if(/$sg4/)` `  {`	In this loop we go through each element of array @rows. This array is obtained by splitting the XML source using string “<SG4>” as a separator. The if-statement checks whether the element matches with the <SG4> tag. At the same time, the values of the SG4 element are captured into special variables $1, $2, etc. The SG4 elements, i.e. the statement entries, are treated in the body of the if-statement.
`    $LIN=$1;          #<LIN>` `    $FTX=$2;          #<FTX>` `    $SG5_MOA=$3;      #<SG5><MOA><C516><D_5025>` `    $balance=$4;      #<SG5><MOA><C516><D_5004>` `    $balance_sign=$5; #<SG5><MOA><PF:D_5003>` `    $date=$6;         #<SG5><DTM>`	Assigning the captured values to variables. The $LIN variable can be equal to “LST”, “LEN”, “LNE”, “LNS”, and to “LTI”, for the opening balance, closing balance,

	transactions (credits and debits), special transactions (with post), and for totals correspondingly.
`$FTX=~s/<D_4440>/[/g;` `$FTX=~s/<\/D_4440>$s/] /g;`	The $FTX string is edited. The opening and closing tags are removed. The info lines are enclosed into square brackets and are separated by spaces.
`$balance=$balance_sign.$balance;`	The $balance variable contains the account balance with sign.
`if(defined($date) && defined($prev_date))` `{` `  if($date ne $prev_date)` `  {` `  $end_date=$prev_date;` `  if(!defined($start_date))` `  {` `  $start_date=$date;` `  }` `  }` `}` `$prev_date=$date;`	Each time a date change occurs, the $end_date variable stores the value of the ended day. Thus at the end of the loop, the $end_date variable contains the last completed day. The $start_date variable will contain at the end of the loop the first opened day. It is not sure if all records of the day before $start_date or

	all records of the day after the $end_date are in the XML file.
`if($LIN eq "LST" && $SG5_MOA == 315)` `{` `  if(!defined($opening_balance))` `  {` `  $opening_balance=$balance;` `  $opening_date=$date;` `  }` `}`	Opening balance (see section 8.4.1). Only the first occurrence of the opening balance (value and date) is memorized. Multiple opening balance entries are possible if the Perl script receives as an input a concatenation of several consecutive XML files.
`elsif($LIN eq "LEN" && $SG5_MOA == 343)` `{` `  $closing_balance=$balance;` `  $closing_date=$date;` `}`	Closing balance (see section 8.4.1). At the end of the loop we have only the value and the date of the last closing balance entry.
`else` `{` `  @subrows=split/<SG6>/;` `  foreach $_ (@subrows)` `  {` `  if(/$sg6/)` `  {`	If not opening or closing balance entry, we scan for all occurrences of <SG6> elements in the current <SG4> entry. The string of the <SG4> element is split using “<SG6>” as a

	separator. Then we process each matched <SG6> tag. The matching operator captures 7 values for each individual <SG6> tag.
$D_1153=$1; #<SG6><RFF><C506><D_1153> $D_1154=$2; #<SG6><RFF><C506><D_1154> $value=$3; #<SG6><DTM> $SG6_MOA=$4; #<SG6><MOA><C516><D_5025> $amount=$5; #<SG6><MOA><C516><D_5004> $amount_sign=$6; #<SG6><MOA><PF:D_5003> $SG6_FTX=$7; #<SG6><FTX>	Storing the captured values into variables.
$scan=$D_1153 eq "ZZZ"?$D_1154:"wire";	The first two captured values are taken from the last occurrence of the <RFF> tag. If there was no such tag, the values are undefined. The value of $scan is always “wire” except for red payment slips, in which case $scan is the filename of the image (without PNG extension).
$SG6_FTX=~s/<\/D_4440>$s<D_4440>/) (/g; $SG6_FTX=~s/<D_4440>/(/g; $SG6_FTX=~s/<\/D_4440>$s/)/g; $info=substr($SG6_FTX,0,25); $information=$FTX.$SG6_FTX;	We replace the opening and closing tags by simple brackets. The info lines are separated by spaces

	(otherwise MS-Word does not wrap the very long lines). We add the $FTX info string of the main <SG4> entry (<SG4>➔<FTX>) in front of the info string of each SG6 element (<SG4>➔<SG6>➔<FTX>). For simple transactions the info string of the main record (<SG4>➔<FTX>) is always empty. The payer's info is in (<SG4>➔<SG6>➔<FTX>).
`$amount=$amount_sign.$amount;`	The sign is added to the $amount.
`if(($LIN eq "LNE" || $LIN eq "LNS") && $SG5_MOA == 15)` `{`	The SG6 element belongs to a transaction ("LNE" in case of simple transactions and "LNS" in case of special transactions with the post office).
`if($verbose)` `{` `printf STDERR "%8s %23s %3s %8s %25s..\n",$date,$scan,$SG6_MOA==210?"cre":"DEB",$amount,$info;`	In a verbose mode we print a digest line per transaction.

} 	The output of the verbose mode is sent to STDERR so as to not affect the parameters displayed on STDOUT which are being captured by a bash script.
if($save && substr($date,2,6) ge $save_start && substr($date,2,6) le $save_end){	If the save option is set we check whether the transaction falls into the provided date range for being saved in the output HTML table.
if($exclude ne "yes" \|\| ($exclude eq "yes" && $SG6_MOA==210 && $information!~/$watch/i)) {	If the exclusion is required, then the debits are not displayed as well as all transactions whose info text matches the $watch RE string provided as a script argument. The matching is case insensitive.
printf htm "<tr>\r\n";	Opening the row of the output table.
$print_date=$date=~/^(\d{4})(\d{2})(\d{2})$/?"$1-$2-$3":$date; printf htm " <td align=center>%s</td>\r\n",$print_date;	The date is stored in the first column.
printf htm " <td width=220>"; if($information=~/$watch/i) {printf htm "%s",$information;} else {printf htm "%s",$information;} printf htm "</td>\r\n";	The information text is stored in the second column. If the text matches RE $watch, the

	information is printed in a color (in case the exclusion is sent to “no”).
printf htm " <td align=center>"; if($scan=~/^\d{23}$/) { printf htm "",$scan; printf htm "<img width=335 src=\"scans/crop.%s.png\">",$scan; printf htm ""; } else { printf htm "%s",$scan; } printf htm "</td>\r\n";	The third column contains the scans of red payment slips. For other transactions it contains a text “wire”. The scans are cropped to display only the reason of the payment with the address of the paying party [cropped example]. The displayed image is a link pointing to the full scan of the image [full scan example].
$print_amount=$amount=~/^\+([\d.]+)$/?$1:$amount; printf htm " <td align=center>"; if($SG6_MOA==211) {printf htm "-%s",$print_amount;} elsif($SG6_MOA==210) {printf htm "%s",$print_amount;} else {printf htm "<u>%s</u>",$print_amount;} printf htm "</td>\r\n";	The forth column contains the amount. The debits are shown in red with a minus sign.
printf htm " <td align=center>%s</td>\r\n","no"; printf htm " <td align=center>%s</td>\r\n",$empty; printf htm " <td align=center>%s</td>\r\n",$empty; printf htm " <td align=center>%s</td>\r\n",$empty;	The five following columns are for the personnel processing the payment. The titles of these

printf htm " <td>%s</td>\r\n",$empty;	columns are the following: Billing Entered, Entered Date, Customer Name, Phone (account) number, Remarks
printf htm "</tr>\r\n";	Closing the current row of the HTML table.
if($SG6_MOA==210){$save_credits+=$amount;} elsif($SG6_MOA==211){$save_debits+=$amount;} $save_transactions++;	The number of transactions and the total amounts of credits and debits **saved into the HTML table** are stored for reporting at the end of the HTML file.
}	End of the if-block: the record is not excluded
}	End of the if-block: the save option is set and the entry is in the requested date range
if($SG6_MOA==210) { $sum_credits+=$amount; } elsif($SG6_MOA==211) { $sum_debits+=$amount; } **$cnt**++;	The number of transactions and the total amounts of credits and debits read from the input XML file are computed and stored for reports and for comparing with the bank totals.
}	If the entry was a

	transaction
elsif($LIN eq "LTI") { if($SG6_MOA==210) { **$bank_credits**+=$amount; } elsif($SG6_MOA==211) { **$bank_debits**+=$amount; } }	If the entry is a bank record of totals, we sum up the total bank credits into $bank_credits and the total bank debits into $bank_debits. Multiple total records are possible if the input file of the script is a concatenation of several consecutive XML files. At the end of processing, these totals must match the totals computed with the amounts of individual transactions.
} } }	End of the body of the loop treating all <SG6> elements of the <SG4> entry, if the entry is not an opening or a closing balance.
} }	End of the body of the loop treating all <SG4> entries.

if(!defined($cnt)) { print STDERR "Error: no transaction is found\n"; exit 2; } if(!defined($bank_credits) || !defined($bank_debits)) { print STDERR "Error: no bank's total record is found\n"; exit 2; } if(!defined($opening_date) || !defined($closing_date) \|\| !defined($opening_balance) || !defined($closing_balance)) { print STDERR "Error: opening and closing balance entries are not found\n"; exit 2; }	An error is occurred, if no transaction is found, if no bank entry with totals is found, if no opening balance entry is found, or if no closing balance entry is found.
$err_tolerance=0.01;	Tolerance of errors for amount comparison is set to 1 centime.
if(abs($sum_credits-$bank_credits)>$err_tolerance \|\| abs($sum_debits- $bank_debits)>$err_tolerance) { print STDERR "[ERROR] the summed credits and debits ($sum_credits, $sum_debits) do not match the bank totals ($bank_credits, $bank_debits)\n"; exit 3; }	If the sums of individual credits and debits are not equal to the total credits and debits of the bank, an error is reported and the script interrupts.
$computed_balance=$opening_balance+$sum_credits-$sum_debits; if(abs($computed_balance-$closing_balance)>$err_tolerance) { print STDERR "[ERROR] the opening balance ($opening_balance) plus[+] credits	If the closing balance value is not equal to the opening balance value plus credits and minus debits, then an

($sum_credits) minus[-] debits ($sum_debits) is equal[=] to $computed_balance, which is NOT the closing balance ($closing_balance)\n"; exit 3; }	error is reported.
if(!defined($start_date) \|\| !defined($end_date)) { print STDERR "[ERROR] at least one date change must occur in the statement (the opening and closing dates cannot be the same)\n"; exit 4; }	If $start_date or $end_date values are not defined, it means that no date change occurs in the XML file, which is an error, since in postfinance files the opening balance is the date of the closing balance of the previous statement.
if($start_date!~/^\d{8}$/ || $end_date!~/^\d{8}$/) { print STDERR "[ERROR] start end date format error\n"; exit 4; } $start_date=substr($start_date,2,6); $end_date=substr($end_date,2,6); if($opening_date!~/^\d{8}$/ || $closing_date!~/^\d{8}$/) { print STDERR "[ERROR] opening closing date format error\n"; exit 4; } $opening_date=substr($opening_date,2,6); $closing_date=substr($closing_date,2,6);	The date values are changed from the 8-digit YYYYMMDD format to the 6-digit YYMMDD format.
print "opening_date=**$opening_date**\n"; print "closing_date=**$closing_date**\n"; print "opening_balance=**$opening_balance**\n";	The script prints on the standard output the parameters

```
print "closing_balance=$closing_balance\n";
print "transactions=$cnt\n";
print "credits=$bank_credits\n";
print "debits=$bank_debits\n";
print "start_date=$start_date\n";
print "end_date=$end_date\n";
```

retrieved from the XML file. A bash script gets these parameters by capturing and evaluating the output.

A bash script can scan the file for retrieving the range of dates and can re-launch the Perl script again for properly saving the selected days.

The $start_date is computed when the first time the date field changes its value and $end_date is computed when the date field changes its value the last time.

If $end_date is greater than or equal to the $start_date, then we can say that the XML file covers all transactions of days from the $start_date through the $end_date inclusive.

	The bash script should retrieve the $start_date and $end_date values and parse them as save options to the Perl script called the second time for creating an output HTML file.
if($save) {	If the save option was set, we start closing the HTML file.
print htm "<tr>\r\n"; print htm " <td align=center>$empty</td>\r\n"; print htm " <td>$empty</td>\r\n"; print htm " <td align=center>Total:</td>\r\n"; print htm " <td align=center>Replace by: <u>Alt</u> T<u>a</u>ble F<u>o</u>rmula =SUM (ABOVE)</td>\r\n"; print htm " <td align=center>$empty</td>\r\n"; print htm " <td align=center>$empty</td>\r\n"; print htm " <td align=center>$empty</td>\r\n"; print htm " <td align=center>$empty</td>\r\n"; print htm " <td>$empty</td>\r\n"; print htm "</tr>\r\n"; print htm "</table>\r\n";	Adding the last row of the table. After copy-pasting the HTML file into a DOC file, user must insert in the last row, under the column of amounts, an MS-Word formula computing the sum of all above values. This sum will be equal to the $save_credits minus $save_debits, both parameters being reported at the end of the HTML file.
print htm " * * * \r\n"; print htm " \r\n";	Headers of two columns of the output HTML

print htm "<table border=1>\r\n"; print htm "<tr><th><u>Input (XML)</u></th><th><u>Saved (HTM/DOC)</u></th></tr>\r\n"; print htm "<tr valign=center>\r\n"; print htm "<td>\r\n";	report table showing at the end of the page the parameters of the input and the output files.
print htm " <table border=1>\r\n"; print htm " <tr>\r\n"; print htm " <th align=right>Parameter</th>\r\n"; print htm " <th align=left>Value</th>\r\n"; print htm " </tr>\r\n"; print htm " <tr><td align=right>opening_date:</td><td>**$opening_date**</td></tr>\r\n"; print htm " <tr><td align=right>closing_date:</td><td>**$closing_date**</td></tr>\r\n"; print htm " <tr><td align=right>opening_balance:</td><td>**$opening_balance</**td></tr>\r\n"; print htm " <tr><td align=right>closing_balance:</td><td>**$closing_balance**</td></tr>\r\n"; print htm " <tr><td align=right>transactions:</td><td>**$cnt**</td></tr>\r\n"; print htm " <tr><td align=right>credits:</td><td>**$bank_credits**</td></tr>\r\n"; print htm " <tr><td align=right>debits:</td><td>**$bank_debits**</td></tr>\r\n"; print htm " <tr><td align=right>start_date:</td><td>**$start_date**</td></tr>\r\n"; print htm " <tr><td align=right>end_date:</td><td>**$end_date**</td></tr>\r\n"; print htm " </table>\r\n";	First column contains the table of parameters and values related to the input XML file.
print htm "</td>\r\n"; print htm "<td>\r\n";	The boundary of columns.
print htm " <table border=1>\r\n"; print htm " <tr>\r\n"; print htm " <th align=right>Parameter</th>\r\n"; print htm " <th align=left>Value</th>\r\n"; print htm " </tr>\r\n"; print htm " <tr><td align=right>save_start:</td><td>$save_start</td></tr>\r\n"; print htm " <tr><td align=right>save_end:</td><td>$save_end</td></tr>\r\n"; print htm " <tr><td align=right>watch:</td><td>$watch</td></tr>\r\n";	The second column contains the table of parameters related to the output HTML file.

`  print htm "  <tr><td align=right>exclude:</td><td>$exclude</td></tr>\r\n";` `  print htm "  <tr><td align=right>save_credits:</td><td><font size=5><b>$save_credits</b><font></td></tr>\r\n";` `  print htm "  <tr><td align=right>save_debits:</td><td>$save_debits</td></tr>\r\n";` `  print htm "  <tr><td align=right>save_transactions:</td><td>$save_transactions</td></tr>\r\n";` `  print htm "  </table>\r\n";`	
`  print htm "</td>\r\n";` `  print htm "</tr>\r\n";` `  print htm "</table>\r\n";` `  print htm "</p>\r\n";`	End of the columns with the parameters of the input and output files.
`  print htm "</body>\r\n";` `  print htm "</html>\r\n";` `  close htm;` `}`	End of the save and close block.

[a47.pl.txt] the version documented above
[a48.pl.txt] new version since 2008-03-19 (new border style)
[a49.pl.txt] new version since 2008-03-20 (for having the total amount in the DOC filename)

5.1. Example of a standalone usage of the Perl script

The following bash script shows an example of how the Perl script can be used:

The bash script	**Comments**
`#!/bin/bash` `tarid=69` `perl_script=a47.pl.txt` `rm *.xml 2> /dev/null`	This part of the script extracts the PNG files from the .tar.gz file downloaded from postfinance interface into a folder “scans”. It extracts the XML file into the current directory.

rm -r scans 2> /dev/null mkdir scans 2>/dev/null if [$? -ne 0] then echo "Cannot create folder \"scans\"" exit 1 fi tar -C scans --wildcards -x "*.png" -vzf asp*_*$tarid.tar.gz tar --wildcards -x "*.xml" -vzf asp*_*$tarid.tar.gz	$tarid contains the last two symbols of the file. This example used file asp308988986_8031442369.tar.gz.
echo -ne "Cropping." for f in scans/??????????????????????.png do if [-f $f] then ref=`basename $f .png` convert -crop 460x220+475+28 $f scans/crop.$ref.png echo -ne "." fi done echo "Done"	We use Imagemagick's convert tool to cut out from the scans the sections reserved for the purpose of the payment. In this example we cut out also the address of the payer [example of the cropped image].
xml=`echo acc_200_f_*.xml` echo -e "\nChecking:" eval `**./$perl_script xml=$xml**` echo opening_date=$opening_date echo closing_date=$closing_date	The first execution of the Perl script analyzes the XML file and figures out the opening and closing dates of the records.
echo -e "\nSaving:" watch="(telekurs\s+multipay|ordre\s+debit\s+direct|online-shopping\s+deferred)" **./$perl_script xml=$xml save=$opening_date-$closing_date htm=out.htm watch=$watch exclude=no**	At the second execution, we request saving of the output into an HTML file. We specify as a range, the period from the opening balance through the closing balance. Remember that the day is closed somewhere in

	afternoon missing the last transactions of the day. The day of the opening balance and the day of the closing balance are not complete. In this example we do not select the complete days but we select the entire range of the XML file (save=$opening_date-$closing_date). The main bash script described in section 6 makes sure to extract only the transactions of full days. Argument watch specifies a Regular Expression for coloring matched transactions without excluding them from the output file. See the output html file [htm].

[crop.sh.txt]

Download the script [crop.sh.txt]. Before its execution you may need to convert the text of the script into a Unix format:

```
$ d2u crop.sh.txt ; ./crop.sh.txt ; u2d crop.sh.txt
crop.sh.txt: done.
20080129103902000200270.png
20080129114609000100006.png
20080129252501000200064.png
20080129248302000100097.png
20080129264602000100151.png
20080129287212000100081.png
acc_200_f_8031442397.xml
Cropping.......Done

Checking:
opening_date=080130
closing_date=080131

Saving:
opening_date=080130
```

```
closing_date=080131
opening_balance=+7212.98
closing_balance=+10439.21
transactions=38
credits=3721.83
debits=495.6
start_date=080131
end_date=080130
crop.sh.txt: done.
```

The script will generate you the following output file [out.htm] (this example uses the data of asp308988986_8031442369.tar.gz input file).

6. The bash script processing sets of consecutive [tar.gz] files downloaded from postfinance

This section presents the bash script, which takes as an input a set of [tar.gz] files downloaded from the postfinance interface and generates a single output table for saving in a DOC format. The script saves only the dates for which the input XML files ensure a full coverage. Partially covered days are not considered for saving. The dates with missing transactions can be completed and saved when additional postfinance files are provided. With only single daily postfinance file where the opening and closing times are the afternoons of two consecutive days, the output table will be empty. For generating an output, at least two consecutive postfinance files are needed. Before processing the bash script checks that the input files are consecutive, without gaps, and that there are no weekly input files overlapping with daily files. Such overlaps would cause some payments to be considered twice.

The bash script	Comments
`#!/bin/bash` `verbose=0` `xml_suffix="range.xml"` `merge_suffix="merge-xml.txt"`	`xml_suffix` is used for names of XML files extracted from the input [tar.gz] postfinance files, once the date ranges of the XML files is known.

htm_suffix="postfinance-xml.htm" doc_suffix="postfinance-xml.doc" perl_script="a47.pl.txt"	merge_suffix is used for the name of a file containing the concatenation of all input XML files. htm_suffix and doc_suffix are used for the names of final output files.
if [! -f $perl_script] then echo "the perl script \"$perl_script\" is not found" exit 1 fi ls *.tar.gz > /dev/null 2>&1 if [! $? -eq 0] then echo "No *.tar.gz file found" exit 1 fi if [! "`perl -e 'print 1+2' 2>/dev/null`" = "3"] then echo "Perl is not installed" exit 1 fi tar --version > /dev/null 2>&1 if [$? -ne 0] then echo "tar program is not installed" exit 1 fi awk --version > /dev/null 2>&1 if [$? -ne 0]	If the Perl script file is not found, if input [tar.gz] files are not found, if Perl is not installed, if the tar tool is not installed, if the awk tool is not installed, or if the convert tool of Imagemagick is not installed, the script prints an error message and interrupts.

then echo "awk program is not installed" exit 1 fi convert --version 2> /dev/null | grep -q ImageMagick if [$? -ne 0] then echo "ImageMagick (http://www.imagemagick.org) is not installed" exit 1 fi	
if ["$verbose" = "1"] then echo "Removing the \"scans\" folder and all XML files, if any, in 5 seconds..." echo "[Press Ctrl-C to Stop]" sleep 5 fi rm --recursive scans 2> /dev/null rm *.xml 2> /dev/null mkdir scans 2>/dev/null if [$? -ne 0] then echo "Cannot create folder \"scans\"" exit 1 fi	Creating the folder “scans” for storing the images of red payment slips.  Removing all XML files in the current folder.
for f in *.tar.gz do tar --wildcards -z -x "*.xml" -f $f 2> /dev/null if [$? -ne 0] then echo "No XML file is found in $f"	Extracting from all [tar.gz] files the XML files. If a [tar.gz] file contains no XML file, an error is reported and the script is terminated. All PNG files are extracted into the

exit 1 fi tar --wildcards --directory=scans -z -x "????????????????????.png" -f $f 2> /dev/null done	“scans” folder.
for f in *.xml do params=`./$perl_script xml=$f` if [$? -ne 0] then echo "the XML file $f contains error" exit 1 fi eval $params mv $f $opening_date-$closing_date-$xml_suffix done	All extracted XML files are analyzed with the Perl script. If the Perl script does not recognize the XML files as postfinance statement files, the bash script will report an error and will interrupt. The extracted XML files are renamed so as their file names contain an yymmdd-YYMMDD prefix, according to their opening and closing balance dates.
ls ??????-??????-$xml_suffix > /dev/null 2>&1 if [$? -ne 0] then echo "No ??????-??????-$xml_suffix file found" exit 1 fi ls *-$xml_suffix | sort | awk -v FS=- ' BEGIN{err=0} closing_date!=""{if($1!=closing_date){ print "Error: The range " $1 "-" $2 " does not follow the range " opening_date "-" closing_date "."; err=1 }} {closing_date=$2; opening_date=$1} END{exit err} ' if [$? -ne 0]	Here the script checks the consecutiveness of the input XML files. The opening balance date of each XML file must be equal to the closing balance date of the previous XML file. If the files are not consecutive (i.e. there are gaps or overlaps), the bash script reports an error and interrupts the further processing.

then echo "The files [tar.gz] are not consecutive. Check for gaps or overlaps." echo "You probably missed a file or downloaded a weekly file together with daily files." exit 1 fi	
open_date=`ls ??????-??????-$**xml_suffix** \| head -1 \| cut -d- -f1` close_date=`ls ??????-??????-$**xml_suffix** | tail -1 | cut -d- -f2` merge_xml=$open_date-$close_date-$merge_suffix cat ??????-??????-$**xml_suffix** > $merge_xml rm ??????-??????-$**xml_suffix**	We are replacing all XML files by a single file containing their concatenations.
params=`**./$perl_script xml=$merge_xml**` if [$? -ne 0] then echo "the merged XML file $merge_xml contains error" exit 1 fi eval "$params" perl -e "if(\"$**start_date**\" le \"$**end_date**\"){exit 0}else{exit 1}" if [$? -ne 0] then echo "The entries of the merged XML file ($opening_date-$closing_date) do not cover one full day." echo "The balance is opened on $opening_date and is closed on $closing_date." echo "This file contains the end of $**end_date** day, but you need the preceding [tar.gz] file for the beginning." echo "It contains also the beginning of $**start_date** day, but	Analyzing the concatenation of all XML files with a Perl script. By evaluating the parameters sent by the Perl script to the standard output, we retrieve the **start_date** and **end_date** values of a period fully covered by the concatenated file. Script terminates if no sufficient records are available for fully covering at least one day.

you need the following [tar.gz] file for completing the day." exit 1 fi	
echo -ne "Cropping." for f in scans/??????????????????????.png do if [-f $f] then ref=`basename $f .png` convert -crop 670x620+475+28 $f scans/crop.$ref.png echo -ne "." fi done echo "Done"	Cropped copies of red payment slips are created which contain only the sections of the reason of the payment and of the address of the paying party. We use the convert tool of the Imagemagick package.
file_htm=$**start_date**-$**end_date**-$htm_suffix file_doc=$**start_date**-$**end_date**-$doc_suffix watch="(telekurs\s+multipay|ordre\s+debit\s+direct|online-shopping\s+deferred)" params=`**./$perl_script xml=$merge_xml** save=$**start_date**-$**end_date** htm=$file_htm watch=$watch exclude=yes` if [$? -ne 0] then echo "error generating the HTML file $file_htm from $merge_xml" exit 1 fi rm ??????-??????-$merge_suffix date1=$opening_date date2=$**start_date** date3=$**end_date** date4=$closing_date	We invoke a call to the Perl script the third and the last time. The Perl script saves the transactions into the HTML table. The **start_date** and **end_date** values refer to the first and last complete days.

`echo "The file $file_htm contains only the transactions of complete days from $date2 thru $date3:"` `echo` `echo "    opening | start      end | closing"` `echo "    balance | date      date | balance"` `echo "    --------+----------------+--------"` `echo "     $date1 | $date2  $date3 | $date4 "` `echo "    --------+----------------+--------"` `echo "     YYMMDD | yymmdd  yymmdd | YYMMDD "` `echo` `echo "Save the content of the HTML file as a DOC file $file_doc for posting to <payment_processing_queue>"`	
`msdir=\`ls -d /cygdrive/*/Program\ Files/Microsoft\ Office 2>/dev/null | head -1\`` `if [ -z "$msdir" ]` `then` `  echo Microsoft Office is not found` `  exit 1` `fi`  `msword=\`find "$msdir" -iname winword.exe 2>/dev/null | head -1\`` `if [ -z "$msword" ]` `then` `  echo Microsoft Word is not found` `  exit 1` `fi` `"$msword" $file_htm` `if [ -f $file_doc ]` `then` `  echo Now post the file $file_doc to <payment_processing_queue>` `else` `  echo The file $file_doc is not yet created`	Searching the drive, where Microsoft Office is installed. Searching the location of the MS Word program (winword.exe). Opening the HTML file with MS Word. The user must insert the formula of the total (at the end of the table according to the instructions in the file) and must save the file in the DOC format.

`  exit 1` `fi`	

[a6.sh.txt] the version documented above
[a7.sh.txt] new version since 2008-03-07
[a8.sh.txt] new version since 2008-03-20 (the total number and amount of transactions in the output filename)

7. New versions release

All releases are available in the following folder.

7.1. Modification of the main bash script (2008-03-07)

New version of the main bash script is released. In the old version the scripts finds the location of the MS Word program and opens the HTML file with MS Word. The document requires the user to save the HTML file as a DOC file. The problem is that the images were not embedded in the DOC file. The DOC file showed the images as far as they are available on the disk at the same location. We need to open the HTML file with internet explorer and an empty DOC file. Then the content of the HTML file must be copied and pasted into the empty DOC file. If we save now the DOC file it contains all images embedded (naturally its size is also larger). The table below shows the last part of the script that undergoes changes.

The bash script	**Comments**
`date1=$opening_date` `date2=$start_date` `date3=$end_date` `date4=$closing_date` `echo "The file $file_htm contains only the transactions"` `echo "of complete days from $date2 thru $date3:"` `echo` `echo "    opening | start      end |`	`The HTML file is created`

closing" echo " balance | date date | balance" echo " --------+----------+--------" echo " $date1 | $date2 $date3 | $date4 " echo " --------+----------+--------" echo " YYMMDD | yymmdd yymmdd | YYMMDD " echo echo "Copy the full content of" echo "HTML file $file_htm (Ctrl-A, Ctrl-C)"	
if [! -f empty.doc] then echo the file empty.doc is not found exit 1 fi cp empty.doc $file_doc echo "and paste the content into" echo "DOC file $file_doc (Ctrl-V)"	If “empty.doc” file exists (unzipped from xml2doc.zip), we create an empty DOC file which must contain the copy of the content of the HTML file. The DOC file has a correct name, representing the ranges of dates.
programsdir=`ls -d /cygdrive/*/Program\ Files 2>/dev/null | head -1` if [-z "$programsdir"] then echo Program Files folder is not found exit 1 fi	Searching the location of the “Program Files” folder.
officedir=`ls -d "$programsdir/Microsoft Office" 2>/dev/null \| head -1` if [-z "$officedir"]	Searching the location of the “Microsoft Office” folder.

then echo Microsoft Office is not found exit 1 fi	
msword=`find "$officedir" -iname winword.exe 2>/dev/null \| head -1` if [-z "$msword"] then echo Microsoft Word is not found exit 1 fi	Searching the location of the Microsoft Word program.
iedir=`ls -d "$programsdir/Internet Explorer" 2>/dev/null \| head -1` if [-z "$iedir"] then echo Internet Explorer folder is not found exit 1 fi	Searching the location of the “Internet Explorer” folder.
iexplore=`find "$iedir" -iname iexplore.exe 2>/dev/null \| head -1` if [-z "$iexplore"] then echo Internet Explorer program is not found exit 1 fi	Searching the location of the Internet Explorer program
pwddos=`pwd | perl -ne '/^\/cygdrive\/(\w+)\/(.*)$/; print "$1:/$2"'`	Computing the current directory name in DOS format.
"$iexplore" "file://$pwddos/$file_htm" &	Opening the HTML file with Internet Explorer (in the background mode so the script continues)
"$msword" $file_doc &	Opening the empty DOC file (with correct name) with Microsoft Word (in the background mode)
echo ""	Final instructions to the user.

echo "Insert the =SUM(ABOVE) formula" echo "at the end of the DOC file (Alt-A-O-Enter)" echo "Save DOC file $file_doc (Ctrl-S)" echo "and post it to <payment_processing_queue>"	

[a7.sh.txt]

7.2. Modification of the Perl script for the new style of table borders (2008-03-19)

The modifications from the version a47.pl.txt to a48.pl.txt can be followed in the table below:

The modification log	**Comments**
$ d2u a47.pl.txt a48.pl.txt a47.pl.txt: done. a48.pl.txt: done. $ cp a47.pl.txt a47-48.txt $ diff a47-48.txt a48.pl.txt | wc -l **143**	Copying version a47.pl.txt to a temporary file a47-48.txt. The temporary file will be gradually modified so as to show the upgrade steps up to the next version a48.pl.txt. For the moment the diff shows 143 lines.
$ echo -e '1,$s/<th/<th $td_border1/g'"\nwq" | ed --quiet a47-48.txt; diff a47-48.txt a48.pl.txt | wc -l **111**	We add $td_border1 style variable in <th> table header cell tags
$ echo -e '1,$s/<td/<td $td_border1/g'"\nwq" | ed --quiet a47-48.txt; diff a47-48.txt a48.pl.txt | wc -l **27**	We add $td_border1 style variable in <td> table data cell tags
$ echo -e '1,$s/<table border=1/<table $table_border1/g'"\nwq" | ed --quiet a47-48.txt; diff a47-48.txt a48.pl.txt | wc -l **19**	We add $table_border1 style variable in <table> tags
$ diff a47-48.txt a48.pl.txt	The remaining differences are the definition of the

33a34,37 > $table_border1="border=1 cellspacing=0 cellpadding=1 style=\'border-collapse:collapse;border:solid black 1\'"; > $table_border2="border=1 cellspacing=0 cellpadding=20 style=\'border-collapse:collapse;border:solid blue 1\'"; > $td_border1="style=\'border:solid black 1\'"; > $td_border2="style=\'border:solid blue 1\'"; 326,327c330,331 < print htm "<table $table_border1>\r\n"; < print htm "<tr><th $td_border1><u>Input (XML)</u></th><th $td_border1><u>Saved (HTM/DOC)</u></th></tr>\r\n"; --- > print htm "<table $table_border2>\r\n"; > print htm "<tr><th $td_border2><u>Input (XML)</u></th><th $td_border2><u>Saved (HTM/DOC)</u></th></tr>\r\n"; 329c333 < print htm "<td $td_border1>\r\n"; --- > print htm "<td $td_border2>\r\n"; 346c350 < print htm "<td $td_border1>\r\n"; --- > print htm "<td $td_border2>\r\n";	table styles and table cell styles. The second table at the bottom of the page, containing the conversion statistics, uses a different style.
$ rm a47-48.txt $ u2d a47.pl.txt a48.pl.txt a47.pl.txt: done. a48.pl.txt: done.	The temporary file for demonstration of the upgrade from version a47.pl.txt to a48.pl.txt is deleted. The files are converted back to DOS format.

7.3. *Modifications in the Perl script for parsing the number and the amount of saved transactions to the bash script (2008-03-20)*

The modifications from version a48.pl.txt to version a49.pl.txt are in the diff output shown below:

Diff output	**Comments**
`$ diff a48.pl.txt a49.pl.txt`	Difference
`311a312,316` `> if($save)` `> {` `>  print "save_credits=$save_credits\n";` `>  print "save_transactions=$save_transactions\n";` `> }`	Print if called for saving the output file. We print the total number and amount of credits saved in the HTML file.
`325a331,341` `>  print htm "<tr>\r\n";` `>  print htm "  <td $td_border1 align=center>$empty</td>\r\n";` `>  print htm "  <td $td_border1>$empty</td>\r\n";` `>  print htm "  <td $td_border1 align=center><font size=5><b>`**`Count:`**`</b></font></td>\r\n";` `>  print htm "  <td $td_border1 align=center><font size=5><b>Replace by: <font color=red><u>Alt</u></font> T<font color=red><u>a</u></font>ble F<font color=red><u>o</u></font>rmula `**`=COUNT (ABOVE) - 1`**`</b></font></td>\r\n";` `>  print htm "  <td $td_border1 align=center>$empty</td>\r\n";` `>  print htm "  <td $td_border1 align=center>$empty</td>\r\n";` `>  print htm "  <td $td_border1 align=center>$empty</td>\r\n";` `>  print htm "  <td $td_border1 align=center>$empty</td>\r\n";` `>  print htm "  <td $td_border1>$empty</td>\r\n";` `>  print htm "</tr>\r\n";`	We add a new row at the bottom of the table of transactions. A cell below the column of amounts of the last row must be replaced by an MS Word formula.
`331c347,350` `<  print htm "<tr><th $td_border2><u>Input (XML)</u></th><th $td_border2><u>Saved (HTM/DOC)</u></th></tr>\r\n";` `---`	Changing the header style of the table of the statistics.

`>  print htm "<tr>\r\n";` `>  print htm "  <th $td_border2><font color=blue>Input (XML)</font></th>\r\n";` `>  print htm "  <th $td_border2><font color=blue>Saved (HTM/DOC)</font></th>\r\n";` `>  print htm "</tr>\r\n";`	
`360c379` `<  print htm "  <tr><td $td_border1 align=right>save_credits:</td><td $td_border1><font size=5><b>$save_credits</b><font></td></tr>\r\n";` `---` `>  print htm "  <tr><td $td_border1 align=right>save_credits:</td><td $td_border1><font size=5><b>$save_credits</b></font></td></tr>\r\n";`	Fixing a bug.
`362c381` `<  print htm "  <tr><td $td_border1 align=right>save_transactions:</td><td $td_border1>$save_transactions</td></tr>\r\n";` `---` `>  print htm "  <tr><td $td_border1 align=right>save_transactions:</td><td $td_border1><font size=5><b>$save_transactions</b></font></td></tr>\r\n";`	Making bold the number of saved transactions in the table of statistics. This value must match the MS Word formula computing the number of table elements.
`$`	End of diff

7.4. New bash script saving the output DOC file with a name containing the total amount

The difference between version a7.sh.txt and a8.sh.txt shows the modifications made in the bash script:

Diff output	**Comments**
`$ diff a7.sh.txt a8.sh.txt`	Difference
`8,9c8,11` `< doc_suffix="postfinance-xml.doc"` `< perl_script="a47.pl.txt"` `---` `> doc_amount_prefix="-post-CHF"`	The Perl script version is different. The elements of the

`> doc_count_prefix="-N"` `> doc_suffix=".doc"` `> perl_script="a49.pl.txt"`	filename are also different. The new format is: yymmdd-yymmdd-post-CHFxx.xx-Nxx.doc
`153d154` `< file_doc=$start_date-$end_date-$doc_suffix` `188a190,191` `> eval "$params"` `> file_doc=$start_date-` `$end_date$doc_amount_prefix$save_credits$doc_count_prefix$save_transactions$doc_suffix`	The parameters received from the Perl script (at its last execution for saving the output HTML file) are evaluated. The retrieved $save_credits and $save_transactions parameters are used for creating the filename of the output DOC file.
`236c239` `< echo "Insert the =SUM(ABOVE) formula"` `---` `> echo "Insert the =SUM(ABOVE) and =COUNT(ABOVE)-1 formulas"`	Reminds the user about the two MS Word formulas. They must be inserted manually
`$`	End of diff

8. References

8.1. Previous procedures for retrieving the lists of payments

The current procedure for retrieving the list of postfinance payments into an excel file: http://switzernet.com/company/080121-howto-retrieve-postfinance-payments/

The procedure for retrieving the list of postfinance payments into an excel file using e-finance Java program: http://switzernet.com/company/080130-postfinance-java-excel/

The procedure for retrieving the BMP files of scans of red payment slips: http://switzernet.com/company/071226-customer-payments-postfinance/

The current procedure for retrieving the red payment slips into a DOC file: http://switzernet.com/company/080118-red-payment-slips-doc-table/

8.2. Preliminary analysis of XML source formats of postfinance

The XML format of the statement files, a conversion of the XML source into a readable layout, the hierarchical structures of the statement records (i.e. of SG4 elements of XML files) is discussed in a previous report [http://www.switzernet.com/company/080129-postfinance-acc-xml-file-format/], [http://4z.com/company/080129-postfinance-acc-xml-file-format/], [http://www.unappel.ch/company/080129-postfinance-acc-xml-file-format/]

8.3. Related links at postfinance

The “Document manager” is software which can convert the XML files of postfinance account into a CSV format [http://www.postfinance.ch/pf/content/fr/seg/priv/customer/download/sw.html]

Sur demande, les écritures des documents de compte électroniques peuvent vous être fournies au format XML, ce qui vous permet de les traiter ultérieurement à l'aide d'un logiciel de e-finance ou de comptabilité apte à le lire

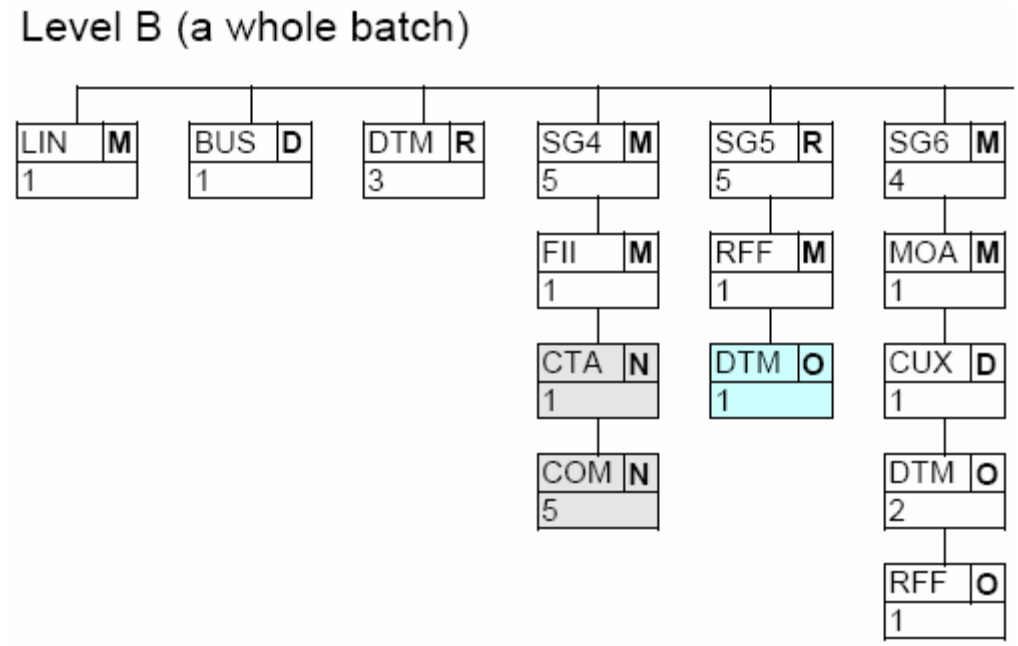
[http://www.postfinance.ch/help/content/fr/help/edoc.html], [http://www.postfinance.ch/help/content/fr/help/edoc/faq.html], [https://www.postfinance.ch/pf/content/fr/seg/biz/product/eserv/spec/edoc.html]

8.4. References to the message formats

The tag names of XML files of postfinance are based on the names of message segments specifications of UN/CEFACT TBG5, where UN CEFACT TBG5 is the standards organization responsible for financial services under the United Nations Centre for Trade facilitation and Electronic Business (UN/CEFACT) and under the United Nations Economic Commission for Europe (UNECE). In 2004 SWIFT and UN/CEFACT TBG5 signed a convergence support agreement [news], [cached].

United Nations Directories for Electronic Data Interchange for Administration, Commerce and Transport [http://www.unece.org/trade/untdid/], [http://www.unece.org/trade/untdid/d00a/trmd/stlrpt_c.htm]

FINPAY, Message Implementation Guideline, Recommendation of UN/CEFACT (TBG5 Working Groups) Finance Domain [http://www.unece.org/trade/untdid/mig/finpay/tbg5_finpay_2002_2-0-0.pdf]:



Message Implementation Guides (MIGs) published by TBG Working Groups [http://www.unece.org/trade/untdid/mig/migs.htm]

FINSTA D.96A, Financial statement of an account message, Recommendation for the domain of Finances for the use of the UN/EDIFACT-Messages [http://www.unece.org/trade/untdid/mig/finsta/finsta_v1-3-2.pdf]:

1.3. Data Segment Clarification

This section should be read in conjunction with the Branching Diagram and the Segment Table which indicate mandatory, conditiona
repeating requirements.

The following semantic principles apply to the message and are intended to facilitate the understanding and implementation of the me
Financial Statement of an Account message is structured in three levels: A,B and C. Level A starts with the UNH segment, level B st
the LIN segment in Segment Group 4 and level C starts with the SEQ segment in Segment Group 6.

- Level A contains general data related to the whole message and compraises segments UNH, BGM, DTM, Segment Groups 1, 2, 3
  segments CNT and UNT. Furthermore, this level describes the kind of statement.
- Level B contains the account related details such as balances, account number and frequency of statement presentation.
- Level C contains the single items as they were advised to the customer by a debit/credit advice. Furthermore, this level contains a
  of a multiple message such as DEBMUL or CREMUL. In some special cases it is possible, that several B levels are contained in a
  without any dependent C levels (balance confirmation for auditing purposes).

Financial statement of an account message, Recommendation of Swiss Financial Institutions for the use of the UN/EDIFACT-Messages [http://www.telekurs.com/dl_tkicch_finsta131.pdf]:

Financial Statement Level B.

SG4	M	9999	LIN-FII-RFF-FTX-SG5-SG6
LIN	M	1	Line item
FII	M	1	Financial institution information
RFF	M	1	Reference
FTX	C	1	Free text
SG5	M	99	MOA-DTM
MOA	M	1	Monetary amount
DTM	C	1	Date/time/period

Financial Statement Level C.

SG6	C	9999	SEQ-RFF-DTM-BUS-MOA-FTX
SEQ	M	1	Sequence details
RFF	M	5	Reference
DTM	M	2	Date/time/period
BUS	M	1	Business function
MOA	M	1	Monetary amount
FTX	C	1	Free text

Direct debit message, Recommendation of Swiss Financial Institutions for the use of the UN/EDIFACT Message [http://www.telekurs.com/dirdeb_lsv_v10.pdf]

Banking status message, Recommendation of Swiss Financial Institutions for the use of the UN/EDIFACT-Message as Message acknowledgment and/or status information [http://www.telekurs.com/dl_tkicch_bansta13.pdf], [cached]:

Banking status Level B.

SG4	M	99	LIN-SG5-SG6
LIN	M	1	Line item
SG5	C	5	RFF-DTM
RFF	M	1	Reference
DTM	C	1	Date/time/period

Banking status Level C.

SG6	C	99	SEQ-GIS-DTM-MOA-CUX-PCD-FTX-DOC-SG7-SG8
SEQ	M	1	Sequence details
GIS	M	1	General indicator
DTM	C	2	Date/time/period
MOA	C	1	Monetary amount
CUX	C	1	Currencies
PCD	C	1	Percentage details
FTX	C	1	Free text
DOC	C	5	Document/message details
SG7	C	1	FII-CTA-COM
FII	M	1	Financial institution information
CTA	C	1	Contact information
COM	C	5	Communication contact
SG8	C	1	NAD-CTA-COM
NAD	M	1	Name and address
CTA	C	1	Contact information
COM	C	5	Communication contact

Swiss Interbank Clearing, the hub for payment traffic, operates the payment systems SIC and Euro SIC in Switzerland and across its borders [http://www.sic.ch/]

ISO/TC 154-UN/CEFACT, Joint Syntax Working Group (JSWG), Database EDIFACT Syntax [http://www.gefeg.com/jswg/v41/data/v41.html]

International Organization for Standardization, International Standards for Business, Government and Society [http://www.iso.ch/cate/cat.html]

UN/EDIFACT, United Nations Standard Message (UNSM), Balance of payment customer transaction report message [http://www.stylusstudio.com/edifact/d04b/BOPCUS.htm]:

ATT	Attribute
BGM	Beginning of message
CNT	Control total
COM	Communication contact
CTA	Contact information
CUX	Currencies
DTM	Date/time/period
FII	Financial institution information
FTX	Free text
GIR	Related identification numbers
LIN	Line item
LOC	Place/location identification
MOA	Monetary amount
NAD	Name and address
PRI	Price details
QTY	Quantity
RCS	Requirements and conditions
RFF	Reference
UNH	Message header
UNT	Message trailer

EAN International [http://www.gs1.nl/hb_eancom/cremul/sc1.htm]

EAN International, Direct debit message [http://www.gs1.se/eancom_2002/ean02s4/user/part2/dirdeb/041.htm]

EDIFACT D6 Finance Working Group Meeting, Stockholm (June 19th-20th, 2000) [http://crg.tbg5-finance.org/minutes/mn000619.doc] [cached]:

```
In Switzerland there are currently 2 projects in XML:
- Postfinance want to develop a statement of account based on FINSTA (the customer doesn’t know what syntax is behind the browser), including image references.
- Swiss banks are also developing an XML message based on CREMUL (this is an interim solution and will move to an international standard as soon as there is one).
```

8.4.1. C516 values of opening and closing balance entries

FINSTA D.96A, Financial statement of an account message, Recommendation of D6 EWG sub-working group Finance for the use of the UN/EDIFACT-Message, Version 1.3.1 from February 16th, 2001 [http://www.tbg5-finance.org/unece/mig/finsta/finsta_v1-3-1.pdf]:

SG4 - 9999 LIN-FII-RFF-FTX-SG5-SG6							
SG5 - 99 MOA-DTM							
UN/EDIFACT Directory 96.A M			D6 M			CRG M	
A group of segments indicating the balance of the account, its type and the respective dates (e.g. opening balance, final closing balance, etc.).							
MOA - 1 Monetary amount							
UN/EDIFACT Directory 96.A M			D6 M			CRG M	
A segment indicating the balance of the account and its type.			Within each B level all DE6345 must contain the same currency code.				
Segment number : 16							
UN/EDIFACT Directory 96.A			D6 Description			CRG Description	
C516 Monetary amount	M		M				
5025 Monetary amount type qualifier	M an..3		M		**One of the following codes must be used when code 54 is used in DE 1001 in BGM:** 315 =Opening balance 343 = Closing balance 344 = Value date balance - or available balance 345 = Cost information for providing the statement		

8.5. References to other developers

Marc SCHAEFER Software, Import XML data from Yellownet/The Post Office, This implements a quick and simple way to import Yellownet accounting data into our accounting system [http://www.alphanet.ch/pub/Main/MarcSCHAEFERSoftware/xml-post-import-records.pl.txt], [cached]

Request to explain the format of postfinance XML files [http://www.developpez.net/forums/archive/index.php/t-251418.html]

A remark about XML, DTD, and XSLT files based on the example of a Yellownet XML file [http://lists.alphanet.ch/pipermail/gull/2004-January/001848.html]

8.6. Perl links

An historical Perl tutorial [http://www.comp.leeds.ac.uk/Perl/], [cached]

Perl functions A-Z [http://perldoc.perl.org/index-functions.html]

Perl regular expressions quick start [http://perldoc.perl.org/perlrequick.html]

Perl regular expressions tutorial [http://perldoc.perl.org/perlretut.html]

A brief introduction and overview of Perl [http://perldoc.perl.org/perlintro.html]

8.7. Packages

You need to install Cygwin (preferably completely including Perl) and Imagemagick.

Cygwin is a Linux-like environment for Windows [http://cygwin.com/]

ImageMagick is a software suite to create, edit, and compose bitmap images [http://www.imagemagick.org]

9. Files and links

9.1. Files of the project

Download the current release of the scripts converting the postfinance XML to DOC format [xml2doc.zip]

All releases are available [folder]

Examples of [tar.gz] XML files used in this project: asp308988986_8014851684.tar.gz (49818 bytes), asp308988986_8015892263.tar.gz (78977 bytes), asp308988986_8016928909.tar.gz (108786 bytes), asp308988986_8017962558.tar.gz (141612 bytes), asp308988986_8018999805.tar.gz (132982 bytes), asp308988986_8021050151.tar.gz (174989 bytes), asp308988986_8022087499.tar.gz (255322 bytes), asp308988986_8023121171.tar.gz (165734 bytes), asp308988986_8024154525.tar.gz (138293 bytes), asp308988986_8025196317.tar.gz (148804 bytes), asp308988986_8028261652.tar.gz (135194 bytes), asp308988986_8029306195.tar.gz (258063 bytes), asp308988986_8030357311.tar.gz (276861 bytes), asp308988986_8031442369.tar.gz (94597 bytes), asp308988986_8032206974.tar.gz (149968 bytes), asp308988986_8035268772.tar.gz (184084 bytes), asp308988986_8036311364.tar.gz (286797 bytes)

The main bash script [a6.sh.txt], [a7.sh.txt] (new version since 2008-03-07), [a8.sh.txt] (new version since 2008-03-20)

Example of an output HTML and DOC files [htm], [doc] (where the input files are the following: asp308988986_8025196317.tar.gz, asp308988986_8028261652.tar.gz, asp308988986_8029306195.tar.gz)

The Perl script for processing the XML files [a47.pl.txt], [a48.pl.txt] (new version since 2008-03-19), [a49.pl.txt] (new version since 2008-03-20)

A bash script for demonstrating a ‘standalone’ use of the Perl script [crop.sh.txt], the output HTML file: [out.htm] (the input file: asp308988986_8031442369.tar.gz)

A Perl script converting the XML source into a readable layout [a10.pl.txt]

9.2. This document

In PDF format [pdf]

In MS Word format [doc]

In HTML format [htm]

Internal version mirrors [http://switzernet.com/company/080302-postfinance-xml2doc/], [http://4z.com/company/080302-postfinance-xml2doc/], and [http://www.unappel.ch/company/080302-postfinance-xml2doc/]

Public version mirrors [http://switzernet.com/public/080302-postfinance-xml2doc/], [http://4z.com/public/080302-postfinance-xml2doc/], and [http://www.unappel.ch/public/080302-postfinance-xml2doc/]

* * *