[1'cbq'1 mr24 import]

SCRIPTS FOR UploadING MR18 customers to  
 the new MR24 database

Created on 2012-11-16  
Updated on 2012-11-26  
Updated on 2012-12-05  
Updated on 2013-03-05  
Nicolas Bondier

[[pdf](index.pdf)][[html](index.htm)][[docx](index.docx)]

\* \* \*

Switzernet logo

Copyright © 2012 by Switzernet

Contents

[Introduction 2](#_Toc341707613)

[Scripts 3](#_Toc341707614)

[Download customers 3](#_Toc341707615)

[Import customers 10](#_Toc341707616)

[Download accounts 11](#_Toc341707617)

[Import accounts 18](#_Toc341707618)

[Import old i\_customer fields 18](#_Toc341707619)

[Import customer sites 20](#_Toc341707620)

[Download Follow-me 23](#_Toc341707621)

[Import follow me 26](#_Toc341707622)

[Imports subscriptions 26](#_Toc341707623)

[How-to use 31](#_Toc341707624)

[Import customers 31](#_Toc341707625)

[Import accounts 37](#_Toc341707626)

[Import old i\_customer fields 46](#_Toc341707627)

[Import customer sites 47](#_Toc341707628)

[Import follow me 51](#_Toc341707629)

[Imports subscriptions 56](#_Toc341707630)

# Introduction

This document describes in the first part the scripts created for downloading customers’ data and uploading this data to the new version of porta-billing. Most of the scripts provided by PortaOne uses excel file for uploading data. Excel files are generated by our own scripts.

In most of the code fields of the tables below, the functions are hyperlinked to easily retrieve their content.

Second part of this document is a how-to use these scripts for migrate all customers to the new billing.

# Scripts

## Download customers

This script is executed on the server with replication of the main porta-billing database.

|  |  |
| --- | --- |
| Code | Comments |
| *#!/usr/bin/perl*  *#*  *# Nicolas Bondier*  *# Switzernet 2012*  *#*    **use** warnings;  **use** strict;  **use** DBI;  **use** Spreadsheet::WriteExcel;  **use** POSIX *qw/strftime/*;  **use** File::Spec::Functions [qw](http://perldoc.perl.org/functions/qw.html)(rel2abs);  **use** File::Basename;  **use** Unicode::Map();  **use** Encode;  **use** List::Util [qw](http://perldoc.perl.org/functions/qw.html)(first);  **use** Number::Latin;     my $test\_mode = 1;     my $dirname = dirname(rel2abs($0));   my $db   = "porta-billing";   my $host = "xxxxxxxxxxxx";   my $user = "xxxxxxxxxxxx";   my $pass = "xxxxxxxxxxxx";     my $dbh = DBI->[connect](http://perldoc.perl.org/functions/connect.html)( "dbi:mysql:dbname=$db;host=$host;", $user, $pass ) or [die](http://perldoc.perl.org/functions/die.html) "Connexion  impossible � la base de donn�es $db !";     my @cols;   my %columns; | Initialization of the different global vars.  We connect to the local database, which is the replication of the whole old master database. |
| my ($sta, $sto) = get\_i\_customer\_range(); | We get the range of the i\_customer we need to upload. |
| my @i\_customers\_list = get\_i\_customers ($sta, $sto); | We get the i\_customer list from the range passed as parameter. See the subroutine. |
| my ( %customers ) = [get\_customer\_data](#get_customer_data)(@i\_customers\_list); | All data of the concerned customers is downloaded from the database and inserted in a hash.  See the subroutine. |
| %customers = [add\_services](#add_services)(%customers); | The customer hash is completed with services subscribed by customers from external tables. |
| to\_excel(%customers); | Finally the hash is written to the excel file. |
| **sub** get\_i\_customer\_range{  [print](http://perldoc.perl.org/functions/print.html) "**\n**Select a range of i\_customer to import.**\n**";  [print](http://perldoc.perl.org/functions/print.html) "First i\_customer:**\n**";     my $first\_c = <>;  [chomp](http://perldoc.perl.org/functions/chomp.html)($first\_c);  *#print '\''.$first\_c.'\'';*      while (  !isint($first\_c) ){  [print](http://perldoc.perl.org/functions/print.html) "Error ! Give a correct value.**\n**";  [print](http://perldoc.perl.org/functions/print.html) "First i\_customer:**\n**";       $first\_c = <>;  [chomp](http://perldoc.perl.org/functions/chomp.html)($first\_c);     }  [print](http://perldoc.perl.org/functions/print.html) "**\n**Last i\_customer:**\n**";     my $last\_c = <>;  [chomp](http://perldoc.perl.org/functions/chomp.html)($last\_c);       while ( !$last\_c || !isint($last\_c) || $last\_c < $first\_c ){  [print](http://perldoc.perl.org/functions/print.html) "Error ! Give a correct value.**\n**";  [print](http://perldoc.perl.org/functions/print.html) "Last i\_customer:**\n**";       $last\_c = <>;  [chomp](http://perldoc.perl.org/functions/chomp.html)($last\_c);     }    **sub** isint{       my $val = [shift](http://perldoc.perl.org/functions/shift.html);  [return](http://perldoc.perl.org/functions/return.html) ($val =~ *m/^\d+$/*);     }    [print](http://perldoc.perl.org/functions/print.html) "**\n**First i\_customer : ".$first\_c."**\n**";  [print](http://perldoc.perl.org/functions/print.html) "Last i\_customer  :  ".$last\_c."**\n**";  [return](http://perldoc.perl.org/functions/return.html) ($first\_c,$last\_c);   } | This subroutine is called in first. It asks to the user, the range of i\_customer we need to download. |
| **sub** get\_i\_customers {     my $start = [shift](http://perldoc.perl.org/functions/shift.html);     my $stop = [shift](http://perldoc.perl.org/functions/shift.html);     my @ret;     my $req = "SELECT        c.i\_customer      FROM        Customers c      INNER JOIN        Accounts a      ON        c.i\_customer=a.i\_customer      WHERE        c.iso\_4217 = 'CHF'      AND        a.id vREGEXP '^41'      AND        i\_rep = '3'      AND  c.i\_customer >= ".$start."      AND        c.i\_customer <= ".$stop."  AND  bill\_status='O';";     my $sth = $dbh->prepare($req);     $sth->execute();     my @row;     while ( @row=$sth->fetchrow\_array ){  [push](http://perldoc.perl.org/functions/push.html)(@ret, $row[0]);     }  [return](http://perldoc.perl.org/functions/return.html) @ret;   } | Subroutine to get only the customers we want to add in the new billing. According to our system of sorting customer by i\_rep, we get only the billable Swiss customers (for the moment). |
| **sub** get\_customer\_data {     my @i\_cus = @\_;     my %hash;     my $req1 = "SELECT      cus.i\_customer                 as OldICustomer,      cus.name                       as CustomerName,      cus.balance                    as Balance,      cus.iso\_4217                   as Currency,      cus.companyname                as CompanyName,      cus.salutation                 as Salutation,      cus.firstname                  as FirstName,      cus.midinit                    as MI,      cus.lastname                   as LastName,      cus.baddr1                     as Address1,  cus.baddr2                     as Address2,      cus.baddr3                     as Address3,      cus.baddr4                     as Address4,      cus.baddr5                     as Address5,      cus.city                       as City,      cus.state                      as ProvinceState,      cus.zip                        as Zip,      cus.country                    as CountryRegion,      cus.note                       as Note,      cus.cont1                      as Contact,      cus.phone1                     as Phone,      cus.faxnum                     as Fax,      cus.phone2                     as AltPhone,      cus.cont2                      as AltContact,      cus.email                      as Email,      cus.bcc                        as BCC,      cus.send\_statistics            as SendStatistics,      cus.login                      as Login,      cus.password                   as Password,      cus.i\_customer\_type            as CustomerType,      cus.i\_billing\_period           as BillingPeriod,      cus.credit\_limit               as CreditLimit,      cus.i\_tariff                   as Tariff,      cus.i\_time\_zone                as TimeZone,      cus.i\_credit\_card              as CreditCard,      cus.i\_env                      as Env,      cus.i\_template                 as Template,      cus.tax\_id                     as TaxID,      cus.blocked                    as Blocked,      cus.ppm\_enabled                as PPMEnabled,      cus.i\_rep                      as Representative,      cus.drm\_enabled                as DRMEnabled,      cus.max\_abbreviated\_length     as AbbreviatedNumberLength,      cus.password\_timestamp         as PasswordTimestamp,      cus.out\_date\_format            as OutDateFormat,      cus.out\_time\_format            as OutTimeFormat,      cus.out\_date\_time\_format       as OutDateTimeFormat,      cus.in\_date\_format             as InDateFormat,      cus.in\_time\_format             as InTimeFormat,      cus.i\_online\_payment\_processor as OnlinePaymentProcessor,      cus.reccuring\_enabled          as ReccuringEnabled,      cus.min\_allowed\_payment        as MinAllowedPayment,      cus.i\_acl                      as ACL,      cus.opening\_balance            as OpeningBalance,      cus.cld\_translation\_rule       as CLDDialingRule,      cus.cli\_in\_translation\_rule    as CLIDialingRule,      cus.i\_lang                     as PreferredLanguage,      cus.credit\_limit\_warning       as BalanceWarningThreshold,      cus.callshop\_enabled           as CallShopEnabled,      cus.billed\_to                  as BilledTo,  cus.i\_routing\_plan             as RoutingPlan,      cus.i\_vd\_plan                  as DiscountPlan,      cus.i\_moh                      as MOH,      cus.i\_customer\_class           as CustomerClass,      cus.bp\_charge\_cc               as BpChargeCc,      cus.unallocated\_payments       as UnallocatedPayments,      cus.bill\_status                as BillStatus,      cus\_notepad.notepad            as Notepad  FROM      Customers cus  INNER JOIN      Customer\_Notepad cus\_notepad  ON      cus\_notepad.i\_customer = cus.i\_customer  WHERE cus.i\_customer = '";     my $n = 0;     while ($i\_cus[$n]){           my $req = $req1.$i\_cus[$n]."';";           my $sth = $dbh->prepare($req) or [die](http://perldoc.perl.org/functions/die.html) DBI->errstr();           $sth->execute() or [die](http://perldoc.perl.org/functions/die.html) DBI->errstr();           @cols = @{ $sth->{NAME} };           $hash{$i\_cus[$n]} = $dbh->selectrow\_hashref($req);           $n++;     }  [return](http://perldoc.perl.org/functions/return.html) %hash;   } | This subroutine collects all required data from each customer and adds it in a hash.  In this hash, each key (ex: OldICustomer, CustomerName …) is the column name of the excel file we will generate.  Each customer’s data hash is inserted in a hash with i\_customer as key.  This hash can be returned once all customers’ data has been downloaded. |
| **sub** add\_services {     my %hash = @\_;  [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvSimcallsLimit');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvSimcallsLimit'} = [get\_srvSimcallsLimit](#get_srvSimcallsLimit)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvCLI');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvCLI'} = [get\_srvCLI](#get_srvCLI)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvCLIR');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvCLIR'} = [get\_srvCLIR](#get_srvCLIR)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvCLIRHide');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvCLIRHide'} = [get\_srvCLIRHide](#get_srvCLIRHide)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvCLIRShow');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvCLIRShow'} = [get\_srvCLIRShow](#get_srvCLIRShow)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvFirstLoginGreeting');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvFirstLoginGreeting'} = [get\_srvFirstLoginGreeting](#get_srvFirstLoginGreeting)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvDistinctiveRingVpn');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvDistinctiveRingVpn'} = [get\_srvDistinctiveRingVpn](#get_srvDistinctiveRingVpn)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvLegalIntercept');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {  $hash{$c}{'srvLegalIntercept'} = [get\_srvLegalIntercept](#get_srvLegalIntercept)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvCallRecording');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvCallRecording'} = [get\_srvCallRecording](#get_srvCallRecording)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvCallParking');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvCallParking'} = [get\_srvCallParking](#get_srvCallParking)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvCentrex');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvCentrex'} = [get\_srvCentrex](#get_srvCentrex)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvCliTrust');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvCliTrust'} = [get\_srvCliTrust](#get_srvCliTrust)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvPaging');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvPaging'} = [get\_srvPaging](#get_srvPaging)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvGroupPickup');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvGroupPickup'} = [get\_srvGroupPickup](#get_srvGroupPickup)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvIpCentrexCare');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvIpCentrexCare'} = [get\_srvIpCentrexCare](#srvIpCentrexCare)($c);     }    [push](http://perldoc.perl.org/functions/push.html) (@cols, 'srvRtppLevel');     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'srvRtppLevel'} = [get\_srvRtppLevel](#get_srvRtppLevel)($c);     }       %hash = [set\_defaults\_values](#set_defaults_values)(%hash);  [return](http://perldoc.perl.org/functions/return.html) %hash;   } | This subroutine add, for all customers, services settings such as the displayed CLI for the account if it is different that account user name, hiding CLI …  Customers’ hash is returned when all new keys for all customers are set.  For each one of the loop, click on the called function. |
| **sub** get\_srvSimcallsLimit {     my $i\_cus = [shift](http://perldoc.perl.org/functions/shift.html);     my $sql = "SELECT        value    FROM        Service\_Attribute\_Values srv    WHERE        i\_foreign = '".$i\_cus."'    AND        i\_sattribute = (SELECT                            sra.i\_sattribute as i\_a                        FROM                            Service\_Attributes sra                        INNER JOIN                            Services sr                        ON                            sr.i\_service=sra.i\_service                        WHERE                            sr.name='sim\_calls\_limit'                        AND                            sr.level = 'Customers'                        AND                            sra.name='max\_calls'    );";     my $sth = $dbh->prepare($sql);     $sth->execute();     my @result = $sth->fetchrow\_array();     my $ret = $result[0];     $ret = '' if (!$ret);     if ($ret eq ''){       $ret = 'N';     } else {       $ret = 'Y';     }  [return](http://perldoc.perl.org/functions/return.html) $ret;   } | This function is used to get the value of the ‘SimcallsLimit’ service attribute, ‘Y’ or ‘N’.  This has been done before seeing the simultaneous calls must be set in Customer Site level. |
| **sub** get\_srvCLI {   my $i\_cus = [shift](http://perldoc.perl.org/functions/shift.html);       my $req = "select service\_flags from Customers where i\_customer = ".$i\_cus." LIMIT 1;";       my $v = $dbh->selectrow\_array($req, [undef](http://perldoc.perl.org/functions/undef.html));       $v = [substr](http://perldoc.perl.org/functions/substr.html)($v, 2, 1);       $v = 'A' if (!$v || $v ne 'Y');  [return](http://perldoc.perl.org/functions/return.html) $v;   } | CLI service, on the customer level, has many options.  We choose between only two according to our current settings.  - ‘A’: means the CLI is the account’s CLI.  - ‘Y’: means the CLI is set in the customer level. We use this feature to display another CLI than account ID for our business customer. |
| **sub** get\_srvCLIR {  [return](http://perldoc.perl.org/functions/return.html) 'P';   } | CLIR service is the default rule for hiding numbers.  3 options are available :  - ‘Y’: Always hide CLI.  - ‘N’: Never hide CLI.  - ‘P’: Automatic.  We only set the setting to this value. This way, customers will use a prefix before the CLD to hide the CLI. |
| **sub** get\_srvCLIRHide {  [return](http://perldoc.perl.org/functions/return.html) '\*81';   } | This is the prefix to add for hiding numbers. The prefix \*81 is the only value we accept for number hiding.  See past researches [[I](http://switzernet.com/3/public/110530-astradV7/)] |
| **sub** get\_srvCLIRShow {  [return](http://perldoc.perl.org/functions/return.html) '';   } | Default is showing, we do not need a prefix for showing CLI. |
| **sub** get\_srvFirstLoginGreeting {  [return](http://perldoc.perl.org/functions/return.html) 'N';   } | As we are uploading customer who are not new, I disabled this feature. |
| **sub** get\_srvDistinctiveRingVpn {  [return](http://perldoc.perl.org/functions/return.html) 'N';   } | We do not support. This is for a distinctive ring when receiving call from an external network. |
| **sub** get\_srvLegalIntercept {  [return](http://perldoc.perl.org/functions/return.html) 'N';   } | No legal intercept. |
| **sub** get\_srvCallRecording {  [return](http://perldoc.perl.org/functions/return.html) 'N';   } | No recording service is implemented on Astrad servers as far. |
| **sub** get\_srvCallParking {  [return](http://perldoc.perl.org/functions/return.html) 'N';   } | Not implemented. |
| **sub** get\_srvCentrex {   my $i\_cus = [shift](http://perldoc.perl.org/functions/shift.html);       my $req = "select value from Service\_Attribute\_Values where i\_foreign = ".$i\_cus." and i\_sattribute = '3' LIMIT 1;";       my $v = $dbh->selectrow\_array($req, [undef](http://perldoc.perl.org/functions/undef.html));       $v = '' if (!$v);  [return](http://perldoc.perl.org/functions/return.html) $v;   } | The srvCentrex value is the number to display in case of the CLI must be different than the account ID (if srvCLI is set to ‘Y’).  On old billing, we used to create an account with the same batch number, but this is no more required in new version of PB. |
| **sub** get\_srvCliTrust {  [return](http://perldoc.perl.org/functions/return.html) 'N';   } | Correspond to the ‘Accept Caller Identity’ setting in service feature. This is not used by our Astrad servers. |
| **sub** get\_srvPaging {  [return](http://perldoc.perl.org/functions/return.html) 'N';   } | Paging service disabled. |
| **sub** get\_srvGroupPickup {  [return](http://perldoc.perl.org/functions/return.html) 'N';   } | Not implemented in Astrad servers. |
| **sub** get\_srvIpCentrexCare {  [return](http://perldoc.perl.org/functions/return.html) 'N';   } | This is the new centrex customer care interface. It is not well implemented on the new MR24. There are some bugs, it is complicated and only English is available. It is disabled by default. |
| **sub** get\_srvRtppLevel {  [return](http://perldoc.perl.org/functions/return.html) 'N';   } | Deactivation of RTP proxy. |
| **sub** set\_defaults\_values {     my %hash = @\_;  *# Access list*     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'ACL'} = '10011';     }  *# Customer Class*     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $hash{$c}{'CustomerClass'} = '2';     }  [return](http://perldoc.perl.org/functions/return.html) %hash;   } | Access list (‘ACL’) is set to the custom access list for all our Swiss customers.  Privileges for accessing customer data from the customer care web interface are defined in the access list with id ‘10011’.  Default created CustomerClass for Swiss customers has id ‘2’. |
| **sub** to\_excel {   my %hash = @\_;     my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = [localtime](http://perldoc.perl.org/functions/localtime.html)([time](http://perldoc.perl.org/functions/time.html));     my $filename = strftime('%y-%m-%d\_%Hh%Mm%Ss',[localtime](http://perldoc.perl.org/functions/localtime.html)).'\_Customers.xls';     my $workbook  = Spreadsheet::WriteExcel->**new**($filename);     my $fl = 1;     my $x;     my $y = 1;     my $worksheet = $workbook->add\_worksheet();     my $temp = '';       for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $x = 0;       foreach (@cols){         if ( $y == 1 ){ $worksheet->[write](http://perldoc.perl.org/functions/write.html)(0, $x, $\_); $columns{$\_} = [get\_column\_AA](#get_column_AA)($x+1);}         $temp = decode("utf8", $hash{$c}{$\_});         $temp = '' if ( !$temp || ![defined](http://perldoc.perl.org/functions/defined.html)($temp) );  *#print $temp;*         $temp = [test\_mode](#test_mode)( $temp, $\_ ) if $test\_mode;         $worksheet->[write](http://perldoc.perl.org/functions/write.html)($y, $x, $temp);         $x++;       }       $y++;     }    [print](http://perldoc.perl.org/functions/print.html) "**\n\n**";     foreach (@cols){  [print](http://perldoc.perl.org/functions/print.html) $\_."=".$columns{$\_}."**\n**";     }    [print](http://perldoc.perl.org/functions/print.html) "**\n\n** # Copy the list above in the importCustomers\_SwitzernetXls.cfg configuration file in the slave **\n**";  [print](http://perldoc.perl.org/functions/print.html) "**\n** # Download and verify created excel file :";  [print](http://perldoc.perl.org/functions/print.html) "**\n** # scp root**\@**pbs1.switzernet.com:".$dirname."/".$filename." . ;  cygstart ./$filename **\n**";  [print](http://perldoc.perl.org/functions/print.html) "**\n** # Import in new master (to execute on slave) :";  [print](http://perldoc.perl.org/functions/print.html) "**\n** # cd /home/porta-admin/importCustomers; scp root**\@**pbs1.switzernet.com:".$dirname."/".$filename." .; ./importCustomers\_SwitzernetXls.pl -v -x ".$filename." -c importCustomers\_SwitzernetXls.cfg**\n**";       } | The final subroutine which write the content of the customers’ hash to the excel file.  At first loop, the script adds the columns names from keys of the hash of customers’ hashes.  We do not forget to decode utf8 values of hash from the database, in order to keep names with accents.  At the end, the script displays the new correspondences between columns letters and names.  This output must be past in the configuration file used by the script for uploading customers.  Ex :  CustomerName=A  Balance=B  …  InDateFormat=BC  … |
| **sub** test\_mode {     my $par1 = [shift](http://perldoc.perl.org/functions/shift.html);     my $par2 = [shift](http://perldoc.perl.org/functions/shift.html);     my $ret = $par1;     if ( $par2 eq 'CustomerName' ){       $par1 = $par1.' '.`date +%y%m%d%H%M%S`;     }     if ( $par2 eq 'Login' ){       $par1 = $par1.'['.`date +%y%m%d%H%M%S`.']';     }     if ( $par2 eq 'Email' ){       $par1 = 'xxxxxxxxx.xxxxxxxx@switzernet.com';     }     $par1 =~ *s/\n//g*;  [return](http://perldoc.perl.org/functions/return.html) $par1;   } | A simple function that some of values unique for testing.  This is important to do not have duplicates values that can break the upload. |
| **sub** get\_column\_AA {     my $index = [shift](http://perldoc.perl.org/functions/shift.html);     my $col = int2latin($index);     $col = [uc](http://perldoc.perl.org/functions/uc.html) $col;  [return](http://perldoc.perl.org/functions/return.html) $col;   } | Return ‘A’ for 1, ‘B’ for 2, … |
| [[full\_version](http://switzernet.com/3/public/121016-import-customer-doc/syntax.php?file=files/pbs1/customers_download.pl.txt&lang=perl&code_display=table)] |  |

## Import customers

The script used for inserting the customers in the database is provided by portaone. For legacy reasons you can only view the full version in the protected folder [[I](http://switzernet.com/3/public/121016-import-customer-doc/protected/syntax.php?file=files/slave/importCustomers/importCustomers_SwitzernetXls.pl.txt&lang=perl)].

The only modifications we had to do on this script, is to get back the new i\_customer of the customer we just inserted in the database. This is essential for linking the accounts with the right customer in the next steps.

|  |  |
| --- | --- |
| Code | Comments |
| my $old\_i\_customer = ''; | Setting a global variable old\_i\_customer for further use. |
| my $i\_cus\_corr\_file = 'i\_cust\_corr.csv';  if ([unlink](http://perldoc.perl.org/functions/unlink.html)($i\_cus\_corr\_file) == 0) {  [print](http://perldoc.perl.org/functions/print.html) "File ".$i\_cus\_corr\_file." deleted successfully.";  } else {  [print](http://perldoc.perl.org/functions/print.html) "File ".$i\_cus\_corr\_file." was not deleted.";  } | Checking if the file of correspondences exists. If yes deleting it in order to create a new one. |
| old\_i\_customer => [q](http://perldoc.perl.org/functions/q.html){OldICustomer}, | This is one of the settings that have been added. OldICustomer is the column of the excel file containing the old i\_customer. |
| if ($k =~ */^old\_i\_customer$/*){             $old\_i\_customer = $object{$k};  [delete](http://perldoc.perl.org/functions/delete.html) $object{$k};           } | Part of the subroutine ‘doRow’ where we get all values of the excel file. The i\_customer is taken and its value is set in the old\_i\_customer global variable. |
| $op->{after\_update\_hook} = **sub** {      *# Here we can do some specific manipulations after object creation*       my ($op, $customer, $subtables) = @\_;  [print](http://perldoc.perl.org/functions/print.html) **STDERR** "Created customer with i\_customer=$customer->{\_\_i\_object}**\n**" if $op->{verbose};  [print](http://perldoc.perl.org/functions/print.html) **STDERR** "Old i\_customer billing :".$old\_i\_customer.".";    [open](http://perldoc.perl.org/functions/open.html) (BILL\_CORR, '>>'.$i\_cus\_corr\_file);  [print](http://perldoc.perl.org/functions/print.html) BILL\_CORR $customer->{\_\_i\_object}.",".$old\_i\_customer."**\n**";  [close](http://perldoc.perl.org/functions/close.html) (BILL\_CORR);   }; | Here, according to the comments, we can place code after the object has been created. We just get the new i\_customer, and then write it in the correspondences file with the old\_i\_cusotmer. |
| [[full\_version](http://switzernet.com/3/public/121016-import-customer-doc/protected/syntax.php?file=files/slave/importCustomers/importCustomers_SwitzernetXls.pl.txt&lang=perl)] |  |

## Download accounts

|  |  |
| --- | --- |
| Code | Comments |
| *#!/usr/bin/perl*  *#*  *# Nicolas Bondier*  *# Switzernet 2012*  *#*    **use** warnings;  **use** strict;  **use** DBI;  **use** Spreadsheet::WriteExcel;  **use** POSIX *qw/strftime/*;  **use** File::Spec::Functions [qw](http://perldoc.perl.org/functions/qw.html)(rel2abs);  **use** File::Basename;  **use** Text::CSV;  **use** Encode;  **use** Number::Latin;    *# Options*   my $test\_mode = 0;   my $print\_hash\_ref = 0;   my $write\_to\_excel = 1; | Includes and testing options. |
| my $dirname = dirname(rel2abs($0));   my $db   = "porta-billing";   my $host = "xxxxxxxxxxxx";   my $user = "xxxxxxxxxxxx";   my $pass = "xxxxxxxxxxxx";   my $dbh  = DBI->[connect](http://perldoc.perl.org/functions/connect.html)( "dbi:mysql:dbname=$db;host=$host;", $user, $pass ) or [die](http://perldoc.perl.org/functions/die.html) "Connexion impossible à la base de données $db !";   my $i\_customer\_file = 'i\_cust\_corr.csv';   my @cols;   my $init\_col = 0; | Connection to the database and initialization of variables. |
| *# Creating hash of accounts*   my %accounts\_list = [get\_account\_list](#get_account_listn2)($i\_customer\_file);   %accounts\_list = [verify\_accounts](#verify_accountsn2)(%accounts\_list);     my %accounts = [get\_account\_data](#get_account_datan2)(%accounts\_list);    *# Fixing id for external tables*   %accounts = [get\_new\_i\_product](#get_new_i_productn2)(%accounts);   %accounts = [get\_new\_i\_access\_level](#get_new_i_access_leveln2)(%accounts);    *# Adding all other fields*   %accounts = [get\_batch](#get_batchn2)(%accounts);   %accounts = [get\_srvCentrex](#get_srvCentrexn2)(%accounts);   %accounts = [get\_srvCLI](#get_srvCLIn2)(%accounts);   %accounts = [get\_srvCLIR](#get_srvCLIRn2)(%accounts);   %accounts = [get\_srvDistinctiveRingVPN](#get_srvDistinctiveRingVPNn2)(%accounts); %accounts = get\_srvLegalIntercept(%accounts);   %accounts = [get\_srvCallRecording](#get_srvCallRecordingn2)(%accounts);   %accounts = [get\_srvEmergency](#get_srvEmergencyn2)(%accounts);   %accounts = [get\_srvAnsweringMode](#get_srvAnsweringModen2)(%accounts);   %accounts = [get\_FollowMeMode](#get_FollowMeModen2)(%accounts);   %accounts = [get\_FollowMeSequence](#get_FollowMeSequencen2)(%accounts);   %accounts = [get\_FollowMeTimeout](#get_FollowMeTimeoutn2)(%accounts);   %accounts = [get\_FollowMeMaxForwards](#get_FollowMeMaxForwardsn2)(%accounts);      *# Write to excel file*  [to\_excel](#to_exceln2)(%accounts) if $write\_to\_excel;    *# printing the list*  [print\_hash\_ref](#print_hash_refn2)(%accounts) if $print\_hash\_ref; | Main manipulation of the account list and account’s hash.  Then we write the final hash to an Excel file. |
| **sub** get\_account\_list {     my $file = [shift](http://perldoc.perl.org/functions/shift.html);     my $csv = Text::CSV->**new**();     my %corr;     my $req = '';    [open](http://perldoc.perl.org/functions/open.html) (CSV, "<", $file) or [die](http://perldoc.perl.org/functions/die.html) $!;     while (<CSV>) {         if ($csv->parse($\_)) {           my @columns = $csv->fields();  *#print "Searching for i\_customer : ".$columns[1]."\n";*           $req  = "SELECT i\_account FROM Accounts WHERE i\_customer = ".$columns[1].";";           my $sth = $dbh->prepare($req) or [die](http://perldoc.perl.org/functions/die.html) DBI->errstr();           $sth->execute();           while (my @results = $sth->fetchrow\_array()) {  *# print "Let's insert the account ".$results[0]." for customer ".$columns[0]." (old icustomer ".$columns[1].")\n";*             $corr{ $results[0] } = $columns[0];           }         } else {           my $err = $csv->error\_input;  [print](http://perldoc.perl.org/functions/print.html) "Failed to parse line: $err";         }     }  [close](http://perldoc.perl.org/functions/close.html) CSV;  [return](http://perldoc.perl.org/functions/return.html) %corr;   } | This function read the list of customers from i\_cust\_corr.csv file created when downloading the customers.  For each of the customer we get the i\_account fields of owned accounts. |
| **sub** get\_account\_data {     my %i\_acc = @\_;    my %hash;     my $req1 = "SELECT      acc.issue\_date               as IssueDate,      acc.iso\_4217                 as Currency,      acc.iso\_639\_1                as PreferredLanguage,      acc.activation\_date          as ActivationDate,      acc.expiration\_date          as ExpirationDate,      acc.life\_time                as LifeTime,      acc.id                       as ID,      acc.i\_product                as Product,      acc.balance                  as Balance,      acc.blocked                  as Blocked,      acc.first\_usage              as FirstUsage,      acc.credit\_limit             as CreditLimit,      acc.billing\_model            as BillingModel,      acc.login                    as Login,      acc.password                 as Password,      acc.i\_env                    as Env,      acc.follow\_me\_enabled        as FollowMe,      'N'                          as UM\_Enabled,      acc.opening\_balance          as OpeningBalance,      acc.control\_number           as ControlNumber,      acc.redirect\_number          as RedirectNumber,      acc.email                    as Email,      acc.i\_lang                   as PreferredLanguage,      acc.ecommerce\_enabled        as EcommerceEnabled,      acc.password\_timestamp       as PasswordTimestamp,      acc\_n.notepad                as Notepad,      acc.out\_date\_format          as OutDateFormat,      acc.out\_time\_format          as OutTimeFormat,      acc.out\_date\_time\_format     as OutDateTimeFormat,      acc.in\_date\_format           as InDateFormat,      acc.in\_time\_format           as InTimeFormat,      acc.i\_vd\_plan                as DiscountPlan,      acc.i\_acl                    as ACL,      acc.i\_time\_zone              as TimeZone,      acc.h323\_password            as VoIPPassword,      acc.i\_account                as OldAccount,      acc.i\_customer               as OldCustomer  FROM      Accounts acc  LEFT JOIN      Account\_Notepad acc\_n  ON      acc.i\_account=acc\_n.i\_account  WHERE      acc.i\_account = '";       foreach my $k ([keys](http://perldoc.perl.org/functions/keys.html) %i\_acc){       my $req = $req1.$k."';";       my $sth = $dbh->prepare($req) or [die](http://perldoc.perl.org/functions/die.html) DBI->errstr();       $sth->execute() or [die](http://perldoc.perl.org/functions/die.html) DBI->errstr();       $hash{$k} = $dbh->selectrow\_hashref($req);       if ($init\_col == 0) {         @cols = @{ $sth->{NAME} };       $init\_col++;       }       %hash = [add\_field\_and\_value](#add_field_and_value)($k,'Customer',$i\_acc{$k},%hash);     }  [return](http://perldoc.perl.org/functions/return.html) %hash;   } | For each i\_account, we get the Account data from the database.  Each value is stored in a hash reference with i\_account as key and the Excel column name as reference. |
| **sub** print\_hash\_ref {     my %hash = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       foreach (@cols){         $hash{$c}{$\_} = '' if (!$hash{$c}{$\_});  [print](http://perldoc.perl.org/functions/print.html) $\_ . "=>" . $hash{$c}{$\_} . "**\n**";       }  [print](http://perldoc.perl.org/functions/print.html) "-----------------------------------------------------------------------**\n**";     }   } | Printing the main account hash reference to view all values. |
| **sub** get\_batch {     my %acc = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       my $req = "SELECT b.name FROM Accounts a INNER JOIN Batch b ON a.i\_batch = b.i\_batch WHERE a.i\_account = ".$acc{$c}{'OldAccount'}." LIMIT 1;";  *# print $req;*       my $v = $dbh->selectrow\_array($req, [undef](http://perldoc.perl.org/functions/undef.html));       %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'Batch',$v,%acc);     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | We get account’s batch. |
| **sub** get\_srvCLI {     my %acc = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       my $req = "select service\_flags from Accounts where i\_account = ".$acc{$c}{'OldAccount'}." LIMIT 1;";       my $v = $dbh->selectrow\_array($req, [undef](http://perldoc.perl.org/functions/undef.html));       $v = [substr](http://perldoc.perl.org/functions/substr.html)($v, 1, 1);       $v = 'A' if (!$v || ($v ne 'Y' && $v ne '^'));       %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'srvCLI',$v,%acc);     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | Accounts have possibility to show a different CLI than their number. There are many possibilities:  - ‘A’ is for displaying the account ID.  - ‘Y’ is for a custom CLI to fill (See srvCentrex bellow).  - ‘^’ mean to take the customers settings for choosing (which can be ‘A’ for account ID). |
| **sub** get\_srvCentrex {     my %acc = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       my $req = "select value from Service\_Attribute\_Values where i\_foreign = ".$acc{$c}{'OldAccount'}." and i\_sattribute = '4' LIMIT 1;";       my $v = $dbh->selectrow\_array($req, [undef](http://perldoc.perl.org/functions/undef.html));       $v = '' if (!$v);       %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'srvCentrex',$v,%acc);     }  [return](http://perldoc.perl.org/functions/return.html) %acc; } | srvCentrex is the CLI to display if is different than account ID. |
| **sub** get\_srvCLIR {     my %acc = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'srvCLIR','^',%acc);     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | We have set the srv\_CLIR in the customer level.  We set here the option to ‘^’, meaning that the value must be found under the customer level.  We do not need to define a value for each account. |
| **sub** get\_srvDistinctiveRingVPN {     my %acc = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'srvDistinctiveRingVPN','^',%acc);     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | We do not support. This is for a distinctive ring when receiving call from an external network.  We refer to customer level. |
| **sub** get\_srvLegalIntercept {     my %acc = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'srvLegalIntercept','^',%acc);     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | Astrad servers do not support this feature.  We refer to customer level. |
| **sub** get\_srvCallRecording {     my %acc = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'srvCallRecording','^',%acc);     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | No call recording.  We refer to customer level. |
| **sub** get\_srvEmergency {     my %acc = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {         %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'srvEmergency','N',%acc);     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | Not activated. |
| **sub** get\_srvAnsweringMode {     my %acc = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'srvAnsweringMode','7',%acc);  *# TODO :*  *# test with no voicemail : %acc =* [add\_field\_and\_value](#add_field_and_value) *($c,'srvAnsweringMode','3',%acc);*     }  [return](http://perldoc.perl.org/functions/return.html) %acc; } | Voicemail answer. |
| **sub** get\_FollowMeMode {     my %acc = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'FollowMeMode','',%acc);     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | The value of the follow me modes are still the same, always, never, …  We just select. |
| **sub** get\_FollowMeSequence {     my %acc = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'FollowMeSequence','Order',%acc);     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | Sequence value does not change, we keep the same. |
| **sub** get\_FollowMeTimeout {     my %acc = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       my $req = "select timeout from Follow\_Me where i\_account = '".$acc{$c}{'OldAccount'}."' LIMIT 1;";       my $v = $dbh->selectrow\_array($req, [undef](http://perldoc.perl.org/functions/undef.html));       %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'FollowMeTimeout',$v,%acc);     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | Timeout before going to the followme, in seconds. |
| **sub** get\_FollowMeMaxForwards {     my %acc = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'FollowMeMaxForwards','20',%acc);     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | Maximum number of forwards. The limit will never be reached. |
| **sub** get\_new\_i\_product{     my %acc = @\_;     my $prods = {                   '97'=>'7',                   '96'=>'8',                   '95'=>'9',                   '69'=>'5',                   '70'=>'4',                   '68'=>'3'                 };     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'Product',$prods->{$acc{$c}{'Product'}},%acc);     }   [return](http://perldoc.perl.org/functions/return.html) %acc;   } | We set correspondences between the products in the new billing and the ones of the old billing.  Here I only made these correspondences with the i\_product, which is safer than the products names. |
| **sub** get\_new\_i\_access\_level{     my %acc = @\_;  *# my $acl = {*  *# '155'=>'10007'*  *# };*     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       %acc = [add\_field\_and\_value](#add_field_and_value) ($c,'ACL','10007',%acc);     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | Access levels for our customers have been set-up. Value ‘10007’ is the i\_access\_level for Switzernet customers. |
| **sub** add\_field\_and\_value {     my $a = [shift](http://perldoc.perl.org/functions/shift.html);     my $n = [shift](http://perldoc.perl.org/functions/shift.html);     my $v = [shift](http://perldoc.perl.org/functions/shift.html);     my %acc = @\_;  *# print "| account : ".$a."|\n";*  *# print "| name    : ".$n."|\n";*  *# print "| value   : ".$v."|\n";*       $acc{$a}{$n} = $v;     if ( ![exist\_in\_array](#exist_in_array)($n,@cols) ){  [push](http://perldoc.perl.org/functions/push.html) (@cols,$n);     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | Subroutine for adding values to the hash more quickly. |
| **sub** exist\_in\_array {     my $value = [shift](http://perldoc.perl.org/functions/shift.html);     my @array = @\_;     my $ret = 0;     my %hash;     %hash = [map](http://perldoc.perl.org/functions/map.html) { $\_ => 1 } @array;     if ($hash{$value}){       $ret = 1;     }  [return](http://perldoc.perl.org/functions/return.html) $ret;   } | Checking if a column exists in the column list. |
| **sub** to\_excel {     my %hash = @\_;     my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = [localtime](http://perldoc.perl.org/functions/localtime.html)([time](http://perldoc.perl.org/functions/time.html));     my $filename = strftime('%y-%m-%d\_%Hh%Mm%Ss',[localtime](http://perldoc.perl.org/functions/localtime.html)).'\_Accounts.xls';     my $workbook  = Spreadsheet::WriteExcel->**new**($filename);     my $fl = 1;     my $x;     my $y = 1;   my $worksheet = $workbook->add\_worksheet();     my $temp = '';     my %columns;    [print](http://perldoc.perl.org/functions/print.html) "**\n\n** Starting account excel file creation ... **\n\n**";       for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $x = 0;       foreach (@cols){         if ( $y == 1 ){ $worksheet->[write](http://perldoc.perl.org/functions/write.html)(0, $x, $\_); $columns{$\_} = get\_column\_AA($x+1);}         $temp = decode("utf8", $hash{$c}{$\_});         $temp = '' if ( !$temp || ![defined](http://perldoc.perl.org/functions/defined.html)($temp) );         $temp = [test\_mode](#test_moden2)( $temp, $\_ ) if $test\_mode;         $worksheet->[write](http://perldoc.perl.org/functions/write.html)($y, $x, $temp);         $x++;       }       $y++;     }  [print](http://perldoc.perl.org/functions/print.html) "**\n** Please copy to upload configuration file, above [Columns], following lines :**\n\n**";     foreach (@cols){  [print](http://perldoc.perl.org/functions/print.html) $\_."=".$columns{$\_}."**\n**";     }  [print](http://perldoc.perl.org/functions/print.html) "**\n\n** Created new excel file : ".$dirname."/".$filename."**\n\n**";   } | This function write the final content to the excel file, put the column names in the top of the Excel file and then display the information to past in the configuration file of the script we will use to upload accounts. |
| **sub** get\_column\_AA {     my $index = [shift](http://perldoc.perl.org/functions/shift.html);     my $col = int2latin($index);     $col = [uc](http://perldoc.perl.org/functions/uc.html) $col;  [return](http://perldoc.perl.org/functions/return.html) $col;   } | Return ‘A’ for 1, ‘B’ for 2, … |
| **sub** test\_mode {  *# For formating a field if testing*     my $par1 = [shift](http://perldoc.perl.org/functions/shift.html);     my $par2 = [shift](http://perldoc.perl.org/functions/shift.html);  *# if ( $par2 eq 'CustomerName' ){*  *# $par1 = $par1.'['.`date +%y%m%d%H%M%S`.']';*  *# }*  [return](http://perldoc.perl.org/functions/return.html) $par1;   } | Testing options. This function has also been used in the customer import script. |
| **sub** verify\_accounts {     my %acc = @\_;     my %centrex\_i\_acc;     my @ret;     my %del;     my %prefs = ( '4121550' => 1,                   '4121504' => 1,                   '4121509' => 1,                   '4121999' => 1,                   '4122550' => 1,                   '4122509' => 1,                   '4122504' => 1,                   '4124504' => 1,                   '4124509' => 1,                   '4126504' => 1,                   '4126509' => 1,                   '4127504' => 1,                   '4127509' => 1,                   '4131504' => 1,                   '4131509' => 1,                   '4132504' => 1,                   '4132509' => 1,                   '4133504' => 1,                   '4133509' => 1,                   '4134504' => 1,                   '4134509' => 1,                   '4141509' => 1,                   '4141504' => 1,                   '4143509' => 1,                   '4143504' => 1,                   '4144504' => 1,                   '4152509' => 1,                   '4152504' => 1,                   '4155509' => 1,                   '4155504' => 1,                   '4156509' => 1,                   '4156504' => 1,                   '4161504' => 1,                   '4161509' => 1,                   '4162504' => 1,                   '4162509' => 1,                   '4171509' => 1,                   '4171504' => 1,                   '4181509' => 1,                   '4181504' => 1,                   '4191209' => 1,                   '4181204' => 1     );    *## DELETING Accounts that are only used for CLI display*  *# We now use new CLI display feature*       my $req = "select acc.i\_account, acc.id from Service\_Attribute\_Values sav INNER JOIN Accounts acc ON sav.value=acc.id where (i\_sattribute = '4' or i\_sattribute = '3') and value != '';";          my $sth = $dbh->prepare($req);          $sth->execute();          my @row;          while ( @row=$sth->fetchrow\_array ){       $centrex\_i\_acc{$row[0]}=$row[1];        }       for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       if ($centrex\_i\_acc{$c}){  [delete](http://perldoc.perl.org/functions/delete.html) $acc{$c} if (!$prefs{[substr](http://perldoc.perl.org/functions/substr.html)($centrex\_i\_acc{$c},0,7)});       }     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | I used this function to filter accounts we use for only displaying different CLI (business customers).  We must not insert them, as from know we will not use anymore this kind of set-up. |
| [[full version](http://switzernet.com/3/public/121016-import-customer-doc/syntax.php?file=files/pbs1/account_download.pl.txt&lang=perl&code_display=table)] |  |

## Import accounts

Import accounts script hasn’t changed, we only need to place the configuration file and the new created excel file. Visit this link to view protected script file [[II](http://switzernet.com/3/public/121016-import-customer-doc/protected/syntax.php?file=files/slave/importAccounts/importAccounts_SwitzernetXls.pl.txt&lang=perl)] and this one [[III](http://switzernet.com/3/public/121016-import-customer-doc/protected/syntax.php?file=files/slave/importAccounts/importAccounts_SwitzernetXls.cfg.txt&lang=perl)] for the configuration file.

## Import old i\_customer fields

Before uploading customer sites, as portaone hasn’t provided any script for this purpose, we had to create one. For more convenience, I created a new field for each customer called old\_i\_customer. This way, it is easier to link old and new i\_customer. I used the SOAP appy in order to insert the new field values.

|  |  |
| --- | --- |
| Code | Comments |
| *#!/usr/bin/perl*  *#*  *# Nicolas Bondier*  *# Switzernet 2012*  *#*  **use** Encode;  **use** strict;  **use** warnings;  **use** DBI;  **use** Data::Dumper;  **use** Text::CSV;  **use** SOAP::Lite  *# +trace=>'debug'*   ;     my $i\_customer\_file = 'i\_cust\_corr.csv';   my $customers\_corr = [get\_customer\_list](#get_customer_listn6)($i\_customer\_file); | Includes and global vars  Getting the i\_customer and old\_i\_customer list to import from the i\_cust\_corr.csv file. |
| my $db="xxxxxxxxxxxx";   my $host="xxxxxxxxxxxx";   my $user = "xxxxxxxxxxxx";   my $pass = "xxxxxxxxxxxx";   my $dbh = DBI->[connect](http://perldoc.perl.org/functions/connect.html)( "dbi:mysql:dbname=$db;host=$host;", $user, $pass ) or [die](http://perldoc.perl.org/functions/die.html) "Connexion impossible à la base de données $db !"; | Setting the mysql connection to the new master server. |
| [binmode](http://perldoc.perl.org/functions/binmode.html)(**STDOUT**, ':utf8');   my $proxy\_host = 'https://xxxxxxxxxxxx'; *# Porta-Billing Admin Server*   my $proxy\_port = 'xxxxxxxxxxxx';   my $login = 'xxxxxxxxxxxx';   my $password = 'xxxxxxxxxxxx';   my $uri\_base = 'http://portaone.com/Porta/SOAP';   my $proxy = "$proxy\_host:$proxy\_port/soap/";   my %uris = (     'Session' => "$uri\_base/Session",     'Account' => "$uri\_base/Account",     'Customer' => "$uri\_base/Customer",   );  **sub** [fault\_handler](#fault_handlern6) {     my ($soap, $res) = @\_;  [die](http://perldoc.perl.org/functions/die.html) "SOAP Fault: $!, " . ([ref](http://perldoc.perl.org/functions/ref.html) $res ? $res->faultstring : $soap->transport->status);   }   my $session\_service = SOAP::Lite     ->uri($uris{'Session'})     ->proxy($proxy)     ->on\_fault(\&fault\_handler)   ;     my $customer\_service = SOAP::Lite     ->uri($uris{'Customer'})     ->proxy($proxy)     ->on\_fault(\&fault\_handler) ;   my $account\_service = SOAP::Lite     ->uri($uris{'Account'})     ->proxy($proxy)     ->on\_fault(\&fault\_handler)   ;  *# required to support dateTime type*   $session\_service->serializer()     ->xmlschema('http://www.w3.org/2001/XMLSchema');   $customer\_service->serializer()     ->xmlschema('http://www.w3.org/2001/XMLSchema');   $account\_service->serializer()     ->xmlschema('http://www.w3.org/2001/XMLSchema');   my $LoginResponse = $session\_service->login($login, $password);   my $session\_id = $LoginResponse->result();  [print](http://perldoc.perl.org/functions/print.html) "Logged in with session $session\_id**\n**";   my $header = SOAP::Header->name('auth\_info')->value({ session\_id => $session\_id }); | Setting connection to the SOAP interface. Here we can access to Sessions, Customers, and Account. |
| foreach ( @$customers\_corr ){     if ([customer\_exist\_in\_new\_billing](#customer_exist_in_new_billingn6)($\_->{i\_customer})){       my $hash = {         i\_customer => $\_->{i\_customer},         old\_i\_customer => $\_->{old\_i\_customer}       };    [update\_custom\_old\_i\_customer\_field](#update_custom_old_i_customer_fieldn6)($hash);     }   } | For all customers of the i\_customer correspondences csv file, we verify the existence in the new billing and then add old\_i\_customer the new billing. |
| **sub** update\_custom\_old\_i\_customer\_field {     my $i\_custs = [shift](http://perldoc.perl.org/functions/shift.html);       my $GetCustomerCustomFieldsValuesRequest = {       i\_customer => $i\_custs->{i\_customer}     };       my $GetCustomerCustomFieldsValuesResponse = $customer\_service->get\_custom\_fields\_values($header, $GetCustomerCustomFieldsValuesRequest)->result;     if (! $GetCustomerCustomFieldsValuesResponse) {  [die](http://perldoc.perl.org/functions/die.html) "No customer found**\n**";     }       my $updates = [       {         text\_value => $i\_custs->{old\_i\_customer},         db\_value => $i\_custs->{old\_i\_customer},         name => 'old i\_customer'       }   ];       my $UpdateCustomerCustomFieldsValuesRequest = {       i\_customer => $i\_custs->{i\_customer},       custom\_fields\_values => $updates     };       my $UpdateCustomerCustomFieldsValuesResponse = $customer\_service->update\_custom\_fields\_values($header, $UpdateCustomerCustomFieldsValuesRequest)->result;           if ( $i\_custs->{old\_i\_customer} == $UpdateCustomerCustomFieldsValuesResponse->{'custom\_fields\_values'}->[0]->{text\_value}){  [print](http://perldoc.perl.org/functions/print.html) $UpdateCustomerCustomFieldsValuesResponse->{'custom\_fields\_values'}->[0]->{text\_value}."**\n**";  [print](http://perldoc.perl.org/functions/print.html) "Updating i\_customer **\'**".$i\_custs->{i\_customer}."**\'** with **\'**old i\_customer**\'**=**\'**".$i\_custs->{old\_i\_customer}."**\'** : OK**\n**"     } else {  [print](http://perldoc.perl.org/functions/print.html) "Updating i\_customer **\'**".$i\_custs->{i\_customer}."**\'** with **\'**old i\_customer**\'**=**\'**".$i\_custs->{old\_i\_customer}."**\'** : FAILED !**\n**"     }   } | Udating the field with the value and verifying the results. |
| **sub** get\_customer\_list {     my $file = [shift](http://perldoc.perl.org/functions/shift.html);     my $csv = Text::CSV->**new**();     my $arr = [];     my $i = 0;    [open](http://perldoc.perl.org/functions/open.html) (CSV, "<", $file) or [die](http://perldoc.perl.org/functions/die.html) $!;     while (<CSV>) {         if ($csv->parse($\_)) {           my @columns = $csv->fields();  [print](http://perldoc.perl.org/functions/print.html) "NEW : ".$columns[0]." OLD i\_customer : ".$columns[1]."**\n**";           $arr->[$i]->{i\_customer} = $columns[0];           $arr->[$i]->{old\_i\_customer} = $columns[1];           $i++;         } else {           my $err = $csv->error\_input;  [print](http://perldoc.perl.org/functions/print.html) "Failed to parse line: $err";         }     }  [close](http://perldoc.perl.org/functions/close.html) CSV;  [return](http://perldoc.perl.org/functions/return.html) $arr;   } | Parsing the excel file containing correspondences between old and new i\_customer.  (Format :  new\_i\_customer1, old\_i\_customer1  new\_i\_customer2, old\_i\_customer2  … , …  ) |
| **sub** customer\_exist\_in\_new\_billing {     my $i\_cus = [shift](http://perldoc.perl.org/functions/shift.html);     my $ret = 0;     my $sth = $dbh->prepare('SELECT COUNT(1) FROM Customers WHERE i\_customer=**\'**'.$i\_cus.'**\'**;');     $sth->execute();     if ($sth->fetch()->[0]) {       $ret = 1;     }   [return](http://perldoc.perl.org/functions/return.html) $ret;   } | Checking if the i\_customer exist in new billing. |
| if ($@) {  [print](http://perldoc.perl.org/functions/print.html) "An error occured: $@**\n**";   }  *# logging out is required*   $session\_service->logout($session\_id);  [exit](http://perldoc.perl.org/functions/exit.html) 0; | Loging out of SOAP. |
| [[full version](http://switzernet.com/3/public/121016-import-customer-doc/syntax.php?file=files//slave/fill_old_i_customer.pl.txt&lang=perl&code_display=table)] |  |

## Import customer sites

This script was not provided by portaone, we have mentioned it.

|  |  |
| --- | --- |
| Code | Comments |
| *#!/usr/bin/perl*    **use** warnings;  **use** strict;  **use** DBI;  **use** Spreadsheet::ParseExcel; | Includes |
| my $account\_data = [get\_accounts](#get_accountsn7)();   $account\_data = [set\_limits](#set_limitsn7)($account\_data);  [main\_site\_upload](#main_site_uploadn7)($account\_data);  [print\_list](#print_listn7)($account\_data);   $dbh->disconnect(); | Main routine. Get the accounts date, setting limits and uploading. |
| my $default\_limits = {     '120701+60 Business Promo'  => 3,     '120701+60 Prepaid Promo'   => 3,     '120701+60 Private Promo'   => 1,     'Business'                  => 3,     'Prepaid'                   => 3,     'Private'                   => 1   };     my $default\_prefixe\_name = {     '120701+60 Business Promo'  => 'business',     '120701+60 Prepaid Promo'   => 'prepaid',     '120701+60 Private Promo'   => 'private',     'Business'                  => 'business',     'Prepaid'                   => 'prepaid',     'Private'                   => 'private'   }; | Definition of many parameters.  First one is the defaults calls limit for each product.  Second hash is the type (business, private, prepaid) of product for each imported product.  We will use this description in the customer site name. |
| my $db="xxxxxxxxxxxxxxxxx";   my $host="xxxxxxxxxxxxxxxxx";   my $user = "xxxxxxxxxxxxxxxxx";   my $pass = "xxxxxxxxxxxxxxxxx";   my $dbh = DBI->[connect](http://perldoc.perl.org/functions/connect.html)( "dbi:mysql:dbname=$db;host=$host;", $user, $pass ) or [die](http://perldoc.perl.org/functions/die.html) "Connexion impossible à la base de données $db !";   my $test\_mode = 1; | Connecting to the new master database. |
| **sub** get\_accounts {     my $req = 'SELECT a.i\_account, a.id, a.i\_customer, p.name as product from Accounts a INNER JOIN Products p ON a.i\_product=p.i\_product WHERE a.id REGEXP **\'**^41[0-9]{9}$**\'**;';     my $sth = $dbh->prepare($req);     $sth->execute();     my $h = $sth->fetchall\_hashref('i\_account');  [print](http://perldoc.perl.org/functions/print.html) $req."**\n**";     if ($test\_mode){       for ([keys](http://perldoc.perl.org/functions/keys.html) %$h) {  [delete](http://perldoc.perl.org/functions/delete.html) $h->{$\_};       }     }  [return](http://perldoc.perl.org/functions/return.html) $h;   } | Select i\_account, numer and i\_customer from Swiss accounts (Do not take Verizon Inbound for ex) and returning a hash with the data. |
| **sub** set\_limits {     my $h = [shift](http://perldoc.perl.org/functions/shift.html);     foreach my $k ( [keys](http://perldoc.perl.org/functions/keys.html)( %$h ) ) {       $h->{$k}->{'max\_simultaneous\_calls'} = $default\_limits->{$h->{$k}->{'product'}};       $h->{$k}->{'max\_incoming\_calls'} = $default\_limits->{$h->{$k}->{'product'}};       $h->{$k}->{'max\_outgoing\_calls'} = $default\_limits->{$h->{$k}->{'product'}};       $h->{$k}->{'max\_forwarded\_calls'} = 5;     }  [return](http://perldoc.perl.org/functions/return.html) $h;   } | Here, we set the value for each one of the limits we want to set in the customer site. Values come from the default\_limits hash initialized at the beginning of the script. |
| **sub** main\_site\_upload {     my $h = [shift](http://perldoc.perl.org/functions/shift.html);     my $cur\_site = '';     foreach my $k ( [keys](http://perldoc.perl.org/functions/keys.html)( %$h ) ) {       my $req = 'SELECT a.i\_customer\_site, cs.name FROM Accounts a INNER JOIN Customer\_Sites cs ON a.i\_customer\_site=cs.i\_customer\_site WHERE id = **\'**'.$h->{$k}->{'id'}.'**\'**';       my $row = $dbh->selectrow\_arrayref($req);       $cur\_site = $row->[0];       my $cur\_name = $row->[1];         if ( $cur\_name && ! $cur\_name =~ */\[\$default\_prefixe\_name->\{\$h->\{\$k\}->\{'product'\}\}\]\$h->\{\$k\}->\{'id'\}/* ){         if ( $cur\_site && 0 ){ *# FOR UPDATE. BUT DISABLED BECAUSE Service\_Attribute\_Values MUST BE UPDATED TOO. OR IT WILL BREAK EVERYTHING*  [update\_customer\_site\_name](#update_customer_site_namen7)("[".$default\_prefixe\_name->{$h->{$k}->{'product'}}."]".$h->{$k}->{'id'}, $cur\_site);         } else {           $cur\_site = [add\_new\_customer\_site](#add_new_customer_siten7)( "[".$default\_prefixe\_name->{$h->{$k}->{'product'}}."]".$h->{$k}->{'id'}, $h->{$k}->{'i\_customer'} );         }         $h->{$k}->{'i\_customer\_site'} = $cur\_site;  [insert\_service\_attribute\_values](#insert_service_attribute_valuesn7)($cur\_site , $h->{$k}->{'max\_simultaneous\_calls'},$h->{$k}->{'max\_incoming\_calls'},$h->{$k}->{'max\_outgoing\_calls'},$h->{$k}->{'max\_forwarded\_calls'});  [update\_account\_customer\_site](#update_account_customer_siten7)($h->{$k}->{'i\_account'},$cur\_site);  [print](http://perldoc.perl.org/functions/print.html) "**\n**";       } elsif (! $cur\_name) {  [print](http://perldoc.perl.org/functions/print.html) "Creating site for $h->{$k}->{'id'}...**\n**";         $cur\_site = [add\_new\_customer\_site](#add_new_customer_siten7)( "[".$default\_prefixe\_name->{$h->{$k}->{'product'}}."]".$h->{$k}->{'id'}, $h->{$k}->{'i\_customer'} );         $h->{$k}->{'i\_customer\_site'} = $cur\_site;  [insert\_service\_attribute\_values](#insert_service_attribute_valuesn7)($cur\_site , $h->{$k}->{'max\_simultaneous\_calls'},$h->{$k}->{'max\_incoming\_calls'},$h->{$k}->{'max\_outgoing\_calls'},$h->{$k}->{'max\_forwarded\_calls'});  [update\_account\_customer\_site](#update_account_customer_siten7)($h->{$k}->{'i\_account'},$cur\_site);  [print](http://perldoc.perl.org/functions/print.html) "**\n**";       } else {  [print](http://perldoc.perl.org/functions/print.html) "Site $cur\_name exist.**\n**";     }     }   } | Main function for uploading the site.  First check if the account has a site.  Then we check if the name of the site correspond to our format : ‘[product]41XXXXXXXXX’  Then we add the new customer site and its services attribute values (values of each option) too. |
| **sub** get\_current\_site {     my $id = [shift](http://perldoc.perl.org/functions/shift.html);     my $ics = '';     my $req = 'SELECT i\_customer\_site FROM Accounts WHERE id = **\'**'.$id.'**\'**';     $ics = $dbh->selectrow\_array($req);  [return](http://perldoc.perl.org/functions/return.html) $ics;   } | Getting the current customer site selected on an account. |
| **sub** add\_new\_customer\_site {     my $name = [shift](http://perldoc.perl.org/functions/shift.html);     my $i\_customer = [shift](http://perldoc.perl.org/functions/shift.html);     my $req = '';     $req = 'INSERT into Customer\_Sites (name,i\_env,service\_flags,i\_customer) values (**\'**'.$name.'**\'**,**\'**1**\'**,**\'**Y**\'**,**\'**'.$i\_customer.'**\'**);';  [print](http://perldoc.perl.org/functions/print.html) $req."**\n**";     $dbh->do($req);     my $new\_i\_cust\_site = $dbh->last\_insert\_id([undef](http://perldoc.perl.org/functions/undef.html), [undef](http://perldoc.perl.org/functions/undef.html), [qw](http://perldoc.perl.org/functions/qw.html)(Customer\_Sites i\_customer\_site));  [return](http://perldoc.perl.org/functions/return.html) $new\_i\_cust\_site;   } | Creation of the customer site, with the first settings :  - name = this format ‘[product]412XXXXXXXX’  - i\_env = always ‘1’  - service\_flags = ‘Y’ for activation  - i\_customer = i\_customer owning this site. |
| **sub** insert\_service\_attribute\_values {     my $i\_cus\_site = [shift](http://perldoc.perl.org/functions/shift.html);     my @params = @\_;     my $req = '';     $req = 'INSERT into Service\_Attribute\_Values (i\_sattribute,value,i\_foreign) values (**\'**46**\'**,**\'**'.$params[0].'**\'**,**\'**'.$i\_cus\_site.'**\'**);';     my $sth = $dbh->prepare($req);     $sth->execute();  [print](http://perldoc.perl.org/functions/print.html) $req."**\n**";     $req = 'INSERT into Service\_Attribute\_Values (i\_sattribute,value,i\_foreign) values (**\'**47**\'**,**\'**'.$params[1].'**\'**,**\'**'.$i\_cus\_site.'**\'**);';  [print](http://perldoc.perl.org/functions/print.html) $req."**\n**";     $sth = $dbh->prepare($req);     $sth->execute();  [print](http://perldoc.perl.org/functions/print.html) $req."**\n**";     $req = 'INSERT into Service\_Attribute\_Values (i\_sattribute,value,i\_foreign) values (**\'**48**\'**,**\'**'.$params[2].'**\'**,**\'**'.$i\_cus\_site.'**\'**);';     $sth = $dbh->prepare($req);     $sth->execute();  [print](http://perldoc.perl.org/functions/print.html) $req."**\n**";       $req = 'INSERT into Service\_Attribute\_Values (i\_sattribute,value,i\_foreign) values (**\'**49**\'**,**\'**'.$params[3].'**\'**,**\'**'.$i\_cus\_site.'**\'**);';     $sth = $dbh->prepare($req);     $sth->execute();   [print](http://perldoc.perl.org/functions/print.html) $req."**\n**";  [return](http://perldoc.perl.org/functions/return.html) 1;   } | Insertion of the options to the customer site.  i\_sattribute are the service id :  - 46 for max\_simultaneous\_calls  - 47 for max\_incoming\_calls  - 48 for max\_outgoing\_calls  - 49 for max\_forwarded\_calls |
| **sub** update\_customer\_site\_name {     my $name = [shift](http://perldoc.perl.org/functions/shift.html);  [print](http://perldoc.perl.org/functions/print.html) "**\n** NAME : ".$name."**\n**";     my $i\_site = [shift](http://perldoc.perl.org/functions/shift.html);     my $req = 'UPDATE Customer\_Sites SET name = **\'**'.$name.'**\'** WHERE i\_customer\_site = **\'**'.$i\_site.'**\'**;';     my $sth = $dbh->prepare($req);     $sth->execute();  [print](http://perldoc.perl.org/functions/print.html) $req."**\n**" if ($verbose);   } | Update a customer site name if the name doesn’t correspond to the template. |
| **sub** update\_account\_customer\_site {     my $i\_acc = [shift](http://perldoc.perl.org/functions/shift.html);     my $i\_site = [shift](http://perldoc.perl.org/functions/shift.html);     my $req = 'UPDATE Accounts SET i\_customer\_site = **\'**'.$i\_site.'**\'** WHERE i\_account = **\'**'.$i\_acc.'**\'**;';     my $sth = $dbh->prepare($req);     $sth->execute();  [print](http://perldoc.perl.org/functions/print.html) $req."**\n**" if ($verbose);   } | Update the account’s to set the new i\_customer\_site to it. |
| **sub** print\_list {     my $h = [shift](http://perldoc.perl.org/functions/shift.html);     my $c = 0;     foreach my $k ( [keys](http://perldoc.perl.org/functions/keys.html)( %$h ) ) {  [print](http://perldoc.perl.org/functions/print.html) "**\n**> I\_ACCOUNT ".$k.":**\n**";       foreach my $k2 ( [keys](http://perldoc.perl.org/functions/keys.html) ( %{$h->{$k}} ) ){  [print](http://perldoc.perl.org/functions/print.html) '- '.$k2.' : '.$h->{$k}->{$k2}."**\n**";       }       $c++;     }  [print](http://perldoc.perl.org/functions/print.html) "**\n** Total : ".$c." Accounts.**\n\n**"   }  **\_\_END\_\_** | Printing the list of accounts. |
| [[full version](http://switzernet.com/3/public/121016-import-customer-doc/protected/syntax.php?file=files%2Fslave%2FimportCustomerSites%2Fimport.pl.txt&lang=PERL&code_display=table)] |  |

## Download Follow-me

|  |  |
| --- | --- |
| Code | Comments |
| *#!/usr/bin/perl*  *#*  *# Nicolas Bondier*  *# Switzernet 2012*  *#*    **use** warnings;  **use** strict;  **use** DBI;  **use** Spreadsheet::WriteExcel;  **use** POSIX *qw/strftime/*;  **use** File::Spec::Functions [qw](http://perldoc.perl.org/functions/qw.html)(rel2abs);  **use** File::Basename;  **use** Text::CSV;  **use** Encode;  **use** Number::Latin;  **use** List::Util [qw](http://perldoc.perl.org/functions/qw.html)(max); | Includes for the script. |
| # Options   my $test\_mode = 0;   my $print\_hash\_ref = 1;   my $write\_to\_excel = 1; | Debug options. |
| # Vars   my $dirname = dirname(rel2abs($0));   my $db   = "porta-billing";   my $host = "xxxxxxxxxxxx";   my $user = "xxxxxxxxxxxx";   my $pass = "xxxxxxxxxxxx";   my $dbh = DBI->[connect](http://perldoc.perl.org/functions/connect.html)( "dbi:mysql:dbname=$db;host=$host;", $user, $pass ) or [die](http://perldoc.perl.org/functions/die.html) "Connexion impossible à la base de données $db !";     my $dbh2 = DBI->[connect](http://perldoc.perl.org/functions/connect.html)( "dbi:mysql:dbname=porta-billing;host=xxxxxxxxxxxx;port=xxxxxxxxxxxx", "xxxxxxxxxxxx", "xxxxxxxxxxxx" ) or [die](http://perldoc.perl.org/functions/die.html) "Connexion impossible à la base de données $db !";   my $i\_customer\_file = 'i\_cust\_corr.csv';     my @cols;   my $init\_col = 0; | Global variables and databases connections.  First MySQL connection is with the new master, second one with the old master. |
| # Creating hash of accounts   my %accounts\_list = [get\_account\_list](#get_account_listn8)($i\_customer\_file);     my %follow\_mes = [get\_follow\_me\_data](#get_follow_me_datan8)(%accounts\_list);     #%follow\_mes = [get\_new\_i\_account](#get_new_i_accountn8)(%follow\_mes);     # Write to excel file  [to\_excel](#to_exceln8)(%follow\_mes) if $write\_to\_excel;     # printing the list  [print\_hash\_ref(%](#print_hash_refn8)follow\_mes) if $print\_hash\_ref;     $dbh->disconnect();   $dbh2->disconnect(); | Main procedures.  We get the list of account,then we add the data from old master and write it to the excel file.  We optionally print the hash.  Disconnection of the MySQL connections. |
| **sub** get\_account\_list {     my $file = [shift](http://perldoc.perl.org/functions/shift.html);     my $csv = Text::CSV->**new**();     my %corr;     my $req = '';    [open](http://perldoc.perl.org/functions/open.html) (CSV, "<", $file) or [die](http://perldoc.perl.org/functions/die.html) $!;     while (<CSV>) {         if ($csv->parse($\_)) {           my @columns = $csv->fields();           #print "Searching for i\_account : ".$columns[1]."\n";           $req  = "SELECT i\_account, id FROM Accounts WHERE i\_customer = ".$columns[1].";";           my $sth = $dbh->prepare($req) or [die](http://perldoc.perl.org/functions/die.html) DBI->errstr();           $sth->execute();           while (my @results = $sth->fetchrow\_array()) {  [print](http://perldoc.perl.org/functions/print.html) "Let's insert i\_account ".$results[0]." for account ".$results[1]."**\n**";             $corr{ $results[0] } = $results[1];           }         } else {           my $err = $csv->error\_input;  [print](http://perldoc.perl.org/functions/print.html) "Failed to parse line: $err";         }     }  [close](http://perldoc.perl.org/functions/close.html) CSV;  [return](http://perldoc.perl.org/functions/return.html) %corr;   } | We get the list of Customers we have imported from the CSV file.  Then we get the account ids and i\_account owned by each customer.  We return the hash. |
| **sub** get\_follow\_me\_data {  my %i\_acc = @\_;     my %hash;     my $key = 1;  my $req1 = "SELECT        fmn.i\_follow\_me\_number  as `FollowMeID`,        a.id                    as `AccountID`,        fmn.i\_follow\_order      as `Order`,        fmn.name                as `Name`,        fmn.active              as `Active`,        fmn.period              as `Period`,        fmn.period\_description  as `PeriodDescription`,        fmn.redirect\_number     as `RedirectNumber`,        fmn.timeout             as `NumberTimeout`    FROM        Follow\_Me\_Numbers fmn    INNER JOIN       Accounts a    ON        fmn.i\_account=a.i\_account    WHERE        fmn.i\_account = '";            my $req2 = "'    ORDER BY      AccountID,      `Order`;";       foreach my $k ([keys](http://perldoc.perl.org/functions/keys.html) %i\_acc){       my $req = $req1.$k.$req2;  [print](http://perldoc.perl.org/functions/print.html) $req."**\n**";       my $sth = $dbh->prepare($req) or [die](http://perldoc.perl.org/functions/die.html) DBI->errstr();       $sth->execute();       while(my $ref = $sth->fetchrow\_hashref) {         if ($init\_col == 0) {           @cols = [keys](http://perldoc.perl.org/functions/keys.html) %$ref;           $init\_col++;         }         foreach my $t ([sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %{$ref}){           $hash{$key}{$t} = $ref->{$t};         }         $key++;       }     }  [return](http://perldoc.perl.org/functions/return.html) %hash;   } | For each of the account, we get the data of the FollowMe and insert it in the excel file. |
| **sub** print\_hash\_ref {     my %hash = @\_;     for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       foreach (@cols){         $hash{$c}{$\_} = '' if (!$hash{$c}{$\_});  [print](http://perldoc.perl.org/functions/print.html) $\_ . "=>" . $hash{$c}{$\_} . "**\n**";       }  [print](http://perldoc.perl.org/functions/print.html) "-----------------------------------------------------------------------**\n**";     }   } | Printing the hash for testing. |
| **sub** add\_field\_and\_value {     my $a = [shift](http://perldoc.perl.org/functions/shift.html);     my $n = [shift](http://perldoc.perl.org/functions/shift.html);     my $v = [shift](http://perldoc.perl.org/functions/shift.html);     my %acc = @\_;     $acc{$a}{$n} = $v;     if ( !exist\_in\_array($n,@cols) ){  [push](http://perldoc.perl.org/functions/push.html) (@cols,$n);     }   [return](http://perldoc.perl.org/functions/return.html) %acc;   } | A simple function to add a value to the hash reference. |
| **sub** exist\_in\_array {     my $value = [shift](http://perldoc.perl.org/functions/shift.html);     my @array = @\_;     my $ret = 0;     my %hash;     %hash = [map](http://perldoc.perl.org/functions/map.html) { $\_ => 1 } @array;     if ($hash{$value}){       $ret = 1;     }  [return](http://perldoc.perl.org/functions/return.html) $ret;   } | A simple function to check if a value exists in array. |
| **sub** to\_excel {     my %hash = @\_;     my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = [localtime](http://perldoc.perl.org/functions/localtime.html)([time](http://perldoc.perl.org/functions/time.html));     my $filename = strftime('%y-%m-%d\_%Hh%Mm%Ss',[localtime](http://perldoc.perl.org/functions/localtime.html)).'\_FollowMe.xls';     my $workbook  = Spreadsheet::WriteExcel->**new**($filename);     my $fl = 1;     my $x;     my $y = 1;     my $worksheet = $workbook->add\_worksheet();     my $temp = '';     my %columns;    [print](http://perldoc.perl.org/functions/print.html) "**\n\n** Starting account excel file creation ... **\n\n**";       for my $c ( [sort](http://perldoc.perl.org/functions/sort.html) [keys](http://perldoc.perl.org/functions/keys.html) %hash ) {       $x = 0;       foreach (@cols){         if ( $y == 1 ){ $worksheet->[write](http://perldoc.perl.org/functions/write.html)(0, $x, $\_); $columns{$\_} = [get\_column\_AA](#get_column_AAn8)($x+1);}           $temp = decode("utf8", $hash{$c}{$\_});           $temp = '' if ( !$temp || ![defined](http://perldoc.perl.org/functions/defined.html)($temp) );           $temp = test\_mode( $temp, $\_ ) if $test\_mode;           $worksheet->[write](http://perldoc.perl.org/functions/write.html)($y, $x, $temp);           $x++;       }       $y++;     }  [print](http://perldoc.perl.org/functions/print.html) "**\n** Please copy to upload configuration file, above [Columns], following lines :**\n\n**";     foreach (@cols){  [print](http://perldoc.perl.org/functions/print.html) $\_."=".$columns{$\_}."**\n**";     }  [print](http://perldoc.perl.org/functions/print.html) "**\n\n** Created new excel file : ".$dirname."/".$filename."**\n\n**";   } | Writing to the Excel file the entire content of the hash reference. |
| **sub** get\_column\_AA {     my $index = [shift](http://perldoc.perl.org/functions/shift.html);   my $col = int2latin($index);     $col = [uc](http://perldoc.perl.org/functions/uc.html) $col;  [return](http://perldoc.perl.org/functions/return.html) $col;   } | Returning column letter for a number. |
| **sub** get\_new\_i\_account{     my %acc = @\_;     my $req1 = "SELECT i\_account from Accounts where id = '";     my $req2 = "';";       foreach my $c ( [keys](http://perldoc.perl.org/functions/keys.html) %acc ) {       my $req = $req1.$acc{$c}{'AccountID'}.$req2;  [print](http://perldoc.perl.org/functions/print.html) $req."**\n**";       my $v = $dbh2->selectrow\_array($req, [undef](http://perldoc.perl.org/functions/undef.html));       if ($v && $v ne ''){         %acc = [add\_field\_and\_value](#add_field_and_valuen8)($c,'AccountID',$v,%acc);       } else {  [delete](http://perldoc.perl.org/functions/delete.html) $acc{$c};       }     }  [return](http://perldoc.perl.org/functions/return.html) %acc;   } | This function returns the new i\_account from new billing from the account id.  It is not used here, the import script use the ID instead. I let this function in case we need it later. |
| **sub** test\_mode {     # For formating a field if testing     my $par1 = [shift](http://perldoc.perl.org/functions/shift.html);     my $par2 = [shift](http://perldoc.perl.org/functions/shift.html);     # if ( $par2 eq 'CustomerName' ){       # $par1 = $par1.'['.`date +%y%m%d%H%M%S`.']';     # }  [return](http://perldoc.perl.org/functions/return.html) $par1;   }  \_\_END\_\_ | This function was used for generating unique values in order to not have duplicates values in the new billing.  Deactivated. |
| [[full version](http://switzernet.com/3/public/121016-import-customer-doc/syntax.php?file=files/pbs1/followme_download.pl.txt&lang=PERL&code_display=table)] |  |

## Import follow me

Import follow-me script hasn’t changed, we only need to place the configuration file and the new created excel file. Visit this link to view protected script file [[II](http://switzernet.com/3/public/121016-import-customer-doc/protected/syntax.php?file=files%2Fslave%2FimportFollowMe%2Fimport.pl.txt&lang=perl&code_display=table)] and this one [[III](http://switzernet.com/3/public/121016-import-customer-doc/protected/syntax.php?file=files/slave/importAccounts/importAccounts_SwitzernetXls.cfg.txt&lang=perl)] for the configuration file.

## Imports subscriptions

We use here the SOAP connexion for uploading customer sites, as portaone doesn’t provide script for subscription uploading.

|  |  |
| --- | --- |
| Code | Comments |
| #!/usr/bin/perl    **use** warnings;  **use** strict;  **use** DBI;  **use** Data::Dumper;  **use** Text::CSV;  **use** Switch;  **use** SOAP::Lite   # +trace=>'debug'   ; | Includes. |
| my $debug = 1;   my $start\_first\_next\_month = 1; | Debug option.  Option for starting subscription at the beginning of the next month (today if 0). |
| my $i\_subscriptions\_corr = {   # old => new      3 =>  4,     29 =>  1,      6 =>  5,      4 =>  6,      5 =>  7,      8 =>  2,     21 =>  8,     10 =>  3,     11 =>  9,     12 => 10,     13 => 11,     14 => 12,     18 => 13,     19 => 14,     20 => 15,      9 => 16,     15 => 17,     28 => 18,      7 => 19,     16 => 20,     17 => 21,     22 => 22,     23 => 23,     24 => 24,     25 => 25,     26 => 26,     27 => 27,      1 => 28,      2 => 29   }; | This hash makes the link between i\_subscription present in the old porta-billing and the new one. |
| my $db="porta-billing";   my $host="master.switzernet.com";   my $user = "xxxxxxxxxxxxx";  my $pass = "xxxxxxxxxxxxx";   my $dbh = DBI->[connect](http://perldoc.perl.org/functions/connect.html)( "dbi:mysql:dbname=$db;host=$host;", $user, $pass ) or [die](http://perldoc.perl.org/functions/die.html) "Connexion impossible à la base de données $db !";     my $db2="porta-billing";   my $host2="pbs1.switzernet.com";   my $user2 = "xxxxxxxxxxxxx";   my $pass2 = "xxxxxxxxxxxxx";   my $dbh2 = DBI->[connect](http://perldoc.perl.org/functions/connect.html)( "dbi:mysql:dbname=$db2;host=$host2;", $user2, $pass2 ) or [die](http://perldoc.perl.org/functions/die.html) "Connexion impossible à la base de données $db2 !"; | Mysql connection to the two old and new databases. |
| [open](http://perldoc.perl.org/functions/open.html) (LOGFILE, '>>log.txt');  my $date = `date +'%Y-%m-%d %T'`;  [chomp](http://perldoc.perl.org/functions/chomp.html)($date);  [print](http://perldoc.perl.org/functions/print.html) LOGFILE " -- LOG -- " . $date . " -- LOG -- **\n**";  [binmode](http://perldoc.perl.org/functions/binmode.html)(**STDOUT**, ':utf8');   my $proxy\_host = 'https://slave.switzernet.com'; # Porta-Billing Admin Server   my $proxy\_port = 'xxxxxxxxxxxxx';   my $login = 'xxxxxxxxxxxxx';   my $password = 'xxxxxxxxxxxxx';   my $uri\_base = 'http://portaone.com/Porta/SOAP';   my $proxy = "$proxy\_host:$proxy\_port/soap/";   my %uris = (     'Session' => "$uri\_base/Session",     'Account' => "$uri\_base/Account",     'Customer' => "$uri\_base/Customer",   );  [binmode](http://perldoc.perl.org/functions/binmode.html)(**STDOUT**, ':utf8');   my $proxy\_host = 'https://slave.switzernet.com'; # Porta-Billing Admin Server   my $proxy\_port = 'xxxxxxxxxxxxx';   my $login = 'xxxxxxxxxxxxx';   my $password = 'xxxxxxxxxxxxx';   my $uri\_base = 'http://portaone.com/Porta/SOAP';   my $proxy = "$proxy\_host:$proxy\_port/soap/";   my %uris = (     'Session' => "$uri\_base/Session",     'Account' => "$uri\_base/Account",     'Customer' => "$uri\_base/Customer",   );  **sub** fault\_handler {     my ($soap, $res) = @\_;  [die](http://perldoc.perl.org/functions/die.html) "SOAP Fault: $!, " . ([ref](http://perldoc.perl.org/functions/ref.html) $res ? $res->faultstring : $soap->transport->status);   }   my $session\_service = SOAP::Lite     ->uri($uris{'Session'})     ->proxy($proxy)     ->on\_fault(\&[fault\_handler](#fault_handlern9))   ;     my $customer\_service = SOAP::Lite     ->uri($uris{'Customer'})     ->proxy($proxy)     ->on\_fault(\&[fault\_handler](#fault_handlern9))   ;   my $account\_service = SOAP::Lite     ->uri($uris{'Account'})     ->proxy($proxy)     ->on\_fault(\&[fault\_handler](#fault_handlern9))   ;   # required to support dateTime type   $session\_service->serializer()   ->xmlschema('http://www.w3.org/2001/XMLSchema');   $customer\_service->serializer()     ->xmlschema('http://www.w3.org/2001/XMLSchema');   $account\_service->serializer()     ->xmlschema('http://www.w3.org/2001/XMLSchema');   my $LoginResponse = $session\_service->login($login, $password);   my $session\_id = $LoginResponse->result();  [print](http://perldoc.perl.org/functions/print.html) "Logged in with session $session\_id**\n**";   my $header = SOAP::Header->name('auth\_info')->value({ session\_id => $session\_id }); | Writing all events to the log file.  Then we establish connections to the SOAP interface of the new billing. |
| my $subscriptions = [get\_subscriptions](#get_subscriptionsn9)();  [print\_list](#print_listn9)($subscriptions);  [insert\_update\_subscr](#insert_update_subscrn9)($subscriptions); | Main routine which get the subscription and insert them in the new billing. |
| $dbh->disconnect();   $dbh2->disconnect(); | Disconnection of the MySQL databases. |
| [close](http://perldoc.perl.org/functions/close.html) (LOGFILE); | Closing log file. |
| **sub** get\_subscriptions {     my $req = 'SELECT cf.name, cfv.i\_customer as i\_customer, cfv.value as old\_i\_customer FROM Custom\_Field\_Values cfv INNER JOIN Custom\_Fields cf ON cfv.i\_custom\_field=cf.i\_custom\_field WHERE cf.name = **\'**old i\_customer**\'** AND cfv.value != **\'\'**;';     my $temp;     my $subs;     my $sth = $dbh->prepare($req);     $sth->execute();     my $h = $sth->fetchall\_hashref('i\_customer');     foreach my $k ( [keys](http://perldoc.perl.org/functions/keys.html)( %$h ) ) {       $req = "SELECT cs.i\_customer\_subscription, cs.i\_subscription, s.name, '' as discount\_rate, cs.start\_date, cs.activation\_date, cs.billed\_to, cs.finish\_date, cs.is\_finished, cs.i\_customer FROM Customer\_Subscriptions cs INNER JOIN Subscriptions s ON s.i\_subscription=cs.i\_subscription WHERE cs.i\_customer = '".$h->{$k}->{old\_i\_customer}."';";       my $sth2 = $dbh2->prepare($req);       $sth2->execute();  [print](http://perldoc.perl.org/functions/print.html) $req."**\n\n**" if ($debug);       $temp = $sth2->fetchall\_hashref('i\_customer\_subscription');       foreach my $tk ( [keys](http://perldoc.perl.org/functions/keys.html) %$temp ){         foreach my $tk2 ( [keys](http://perldoc.perl.org/functions/keys.html) ( %{$temp->{$tk}} ) ){           $temp->{$tk}->{$tk2} = '' if(!$temp->{$tk}->{$tk2});         }         $temp->{$tk}->{i\_subscription} = $i\_subscriptions\_corr->{$temp->{$tk}->{i\_subscription}};         $temp->{$tk}->{i\_customer} = $h->{$k}->{i\_customer};         ($temp->{$tk}->{todo}, $temp->{$tk}->{i\_customer\_subscription}) = subscription\_synch\_status($temp->{$tk});         $subs->{$tk} = $temp->{$tk};       }     }  [return](http://perldoc.perl.org/functions/return.html) $subs;   } | Getting the values of the follow me in the old billing and checking what to do. |
| **sub** update\_subscription\_new\_pb {     my $sub = [shift](http://perldoc.perl.org/functions/shift.html);       my $CustomerSubscriptionInfo =         {           name                     => $sub->{name},           discount\_rate            => $sub->{discount\_rate},           activation\_date          => $sub->{activation\_date},           finish\_date              => $sub->{finish\_date},           is\_finished              => $sub->{is\_finished},         }       ;    [print](http://perldoc.perl.org/functions/print.html) "ACT DATE ".$sub->{activation\_date};         my $UpdateCustomerSubscriptionRequest = {         i\_customer => $sub->{i\_customer},         i\_customer\_subscription => $sub->{i\_customer\_subscription},         subscription\_info => $CustomerSubscriptionInfo       };    [print](http://perldoc.perl.org/functions/print.html) " UPDATE **\n\n**" if ($debug);         my $AddUpdateCustomerSubscriptionResponse = $customer\_service->update\_subscription($header,$UpdateCustomerSubscriptionRequest)->result;       my $i\_customer\_subscription = $AddUpdateCustomerSubscriptionResponse->{i\_customer\_subscription};    [print](http://perldoc.perl.org/functions/print.html) "RETURN VALUE : ".$i\_customer\_subscription."**\n**" if ($debug);   } | This subroutine updates the subscription. It is not used currently. We prefer adding a new subscription instead of updating, which is more dangerous. |
| **sub** subscription\_synch\_status {     my $cust = [shift](http://perldoc.perl.org/functions/shift.html);     my $c = 0;     my $todo = ['ignore','update','insert'];     my $found = 0;     my $diff = 0;     my $h = {};     my $ics = '';     my $k = '';     my $req = "SELECT cs.i\_customer\_subscription ,s.name, '' as discount\_rate, cs.activation\_date, cs.finish\_date, cs.is\_finished, cs.i\_customer FROM Customer\_Subscriptions cs INNER JOIN Subscriptions s ON s.i\_subscription=cs.i\_subscription WHERE cs.i\_customer = '".$cust->{i\_customer}."' AND s.name = '".$cust->{name}."';";  [print](http://perldoc.perl.org/functions/print.html) $req."**\n\n**"  if ($debug);     my $sth = $dbh->prepare($req);     $sth->execute();     while ($h = $sth->fetchrow\_hashref) {       foreach $k ( [keys](http://perldoc.perl.org/functions/keys.html)( %$h ) ) {         $h->{$k} = '' if( !$h->{$k});  [print](http://perldoc.perl.org/functions/print.html) $k." : ".$h->{$k}."  ?   ".$cust->{$k}."**\n\n**"  if ($debug);                if ( $k eq 'i\_customer\_subscription' ){           $ics = $h->{$k};         } else {           $diff = 1;  [print](http://perldoc.perl.org/functions/print.html) "DIFF**\n**";         }       }       $found = 1;       if ( $h->{is\_finished} eq 'Y' ){           $diff = 0;       }     }     if ($found && !$diff){       $c = 0;     } elsif ( $found && $diff ){       $c = 1;     } else {       $c = 2;     }  [return](http://perldoc.perl.org/functions/return.html) ($todo->[$c], $ics);   } | This function check is the subscriptions of the customer passed in input are the same as in the old master.  It returns the the task to do for each one (insert, update, ignore). |
| **sub** insert\_subscription\_new\_pb {     my $sub = [shift](http://perldoc.perl.org/functions/shift.html);     my $i\_customer\_subscription = 0;     my $CustomerSubscriptionInfo =       {         i\_subscription  => $sub->{i\_subscription},         name            => $sub->{name},         discount\_rate   => $sub->{discount\_rate},         start\_date      => $sub->{start\_date},         int\_status      => 1,         discount\_rate   => $sub->{discount\_rate},         billed\_to       => $sub->{billed\_to},         finish\_date     => $sub->{finish\_date},         is\_finished     => $sub->{is\_finished},         i\_customer      => $sub->{i\_customer},       }     ;       if ($start\_first\_next\_month){       $CustomerSubscriptionInfo->{start\_date} = `date +%Y-%m-01 -d '+1 month'`;     }       my $AddCustomerSubscriptionRequest = {       i\_customer => $sub->{i\_customer},     subscription\_info => $CustomerSubscriptionInfo     };       foreach my $kcsi ( [keys](http://perldoc.perl.org/functions/keys.html) %$CustomerSubscriptionInfo ){  [print](http://perldoc.perl.org/functions/print.html) $kcsi . " => " . $CustomerSubscriptionInfo->{$kcsi} . "**\n**" ;     }    [print](http://perldoc.perl.org/functions/print.html) " INSERT **\n\n**" if ($debug);       my $AddUpdateCustomerSubscriptionResponse = $customer\_service->add\_subscription($header,$AddCustomerSubscriptionRequest)->result;     $i\_customer\_subscription = $AddUpdateCustomerSubscriptionResponse->{i\_customer\_subscription};    [print](http://perldoc.perl.org/functions/print.html) "RETURN VALUE : ".$i\_customer\_subscription."**\n**" if ($debug);   } | Inserting a new subscription in the new billing. The new subscription can be activated only in present or future. |
| **sub** insert\_update\_subscr {     my $subs = [shift](http://perldoc.perl.org/functions/shift.html);     foreach my $k ( [keys](http://perldoc.perl.org/functions/keys.html)( %$subs ) ) {       if ( $subs->{$k}->{todo} eq 'insert' ) {  [insert\_subscription\_new\_pb](#insert_subscription_new_pbn9)($subs->{$k});       } elsif ( $subs->{$k}->{todo} eq 'update' ){  [update\_subscription\_new\_pb](#update_subscription_new_n9)($subs->{$k});       } else {  [print](http://perldoc.perl.org/functions/print.html) LOGFILE " Ignoring subscription " . $subs->{$k}->{i\_subscription} . " OF CUSTOMER "  . $subs->{$k}->{i\_customer}."**\n**";  [print](http://perldoc.perl.org/functions/print.html) " INGNORING SUBSCRIPTION " if ($debug);       }     }   } | A simple function to define the procedure to follow depending of the ‘todo’ field we have added before. |
| **sub** print\_list {     my $h = [shift](http://perldoc.perl.org/functions/shift.html);     my $c = 0;     foreach my $k ( [keys](http://perldoc.perl.org/functions/keys.html)( %$h ) ) {  [print](http://perldoc.perl.org/functions/print.html) "**\n**> KEY ".$k.":**\n**";       foreach my $k2 ( [keys](http://perldoc.perl.org/functions/keys.html) ( %{$h->{$k}} ) ){  [print](http://perldoc.perl.org/functions/print.html) '- '.$k2.' : '.$h->{$k}->{$k2}."**\n**";       }       $c++;     }  [print](http://perldoc.perl.org/functions/print.html) "**\n** Total : ".$c." registers.**\n\n**"   } | Printing all data for debug. |
| **\_\_END\_\_** | The end. |
| [[full version](http://switzernet.com/3/public/121016-import-customer-doc/protected/syntax.php?file=files%2Fslave%2FimportSubscriptions%2Fimport.pl.txt&lang=perl&code_display=table)] |  |

# How-to use



**[UPDATE] use the new how-to document at:**   
<http://switzernet.com/3/public/130305-import-customers-how-to/>

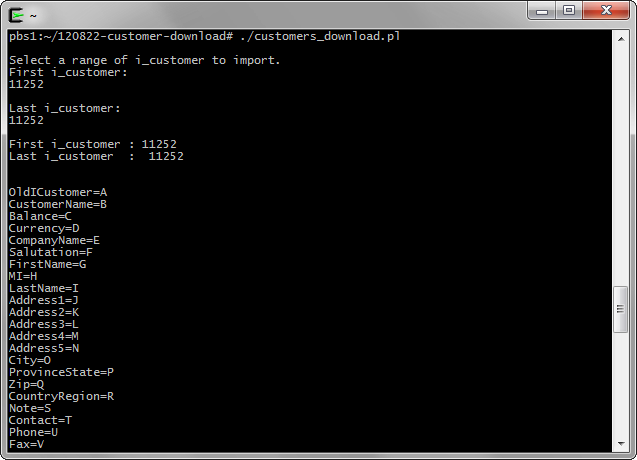
In this how-to, we will only take a sample account. The procedure is the same for large number of accounts.

## Import customers

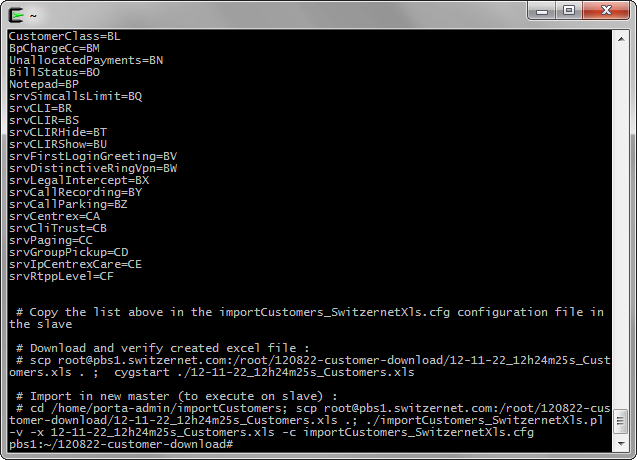
On pbs1 go to /root/120822-customer-download/ and execute the following commands:

|  |
| --- |
| pbs1:~/120822-customer-download# ./customers\_download.pl  Select a range of i\_customer to import.  First i\_customer:  11252  Last i\_customer:  [...] |

The new excel file will be created. As output, you will see the correspondences between column name and letters (A, B, C …) as the example bellow shows:

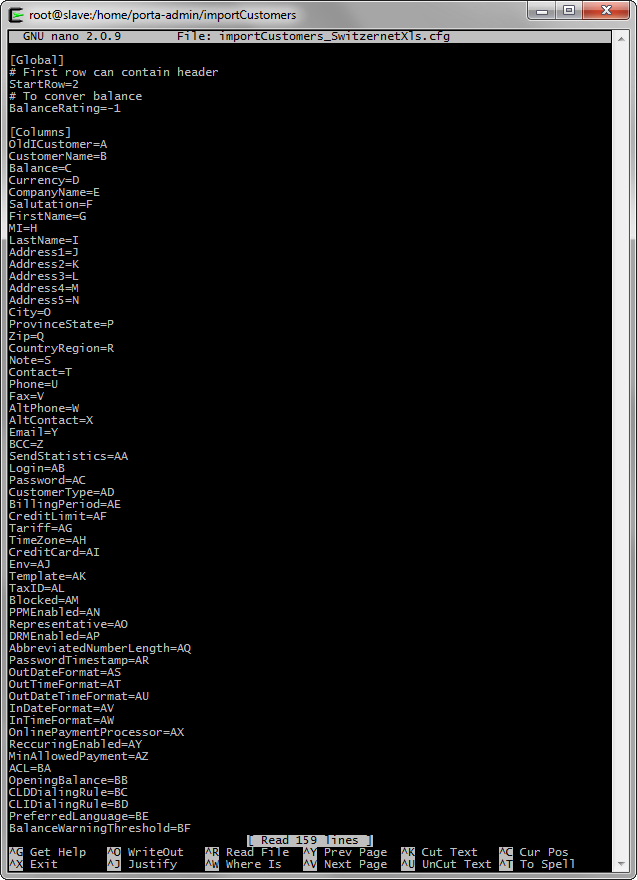


And next steps to execute:



Connect through SSH to the new slave server and go to /home/porta-admin/importCustomers. Open importCustomers\_SwitzernetXls.cfg and past the list of correspondences under [Columns].

|  |
| --- |
| Nicolas Bondier@NicolasBondier ~  $ ssh switz@slave.switzernet.com  Last login: Thu Nov 22 11:35:39 2012 from 212.147.8.99  [switz@slave ~]$ su -  Password:  [root@slave ~]# cd /home/porta-admin/importCustomers/  [root@slave importCustomers]# nano importCustomers\_SwitzernetXls.cfg |

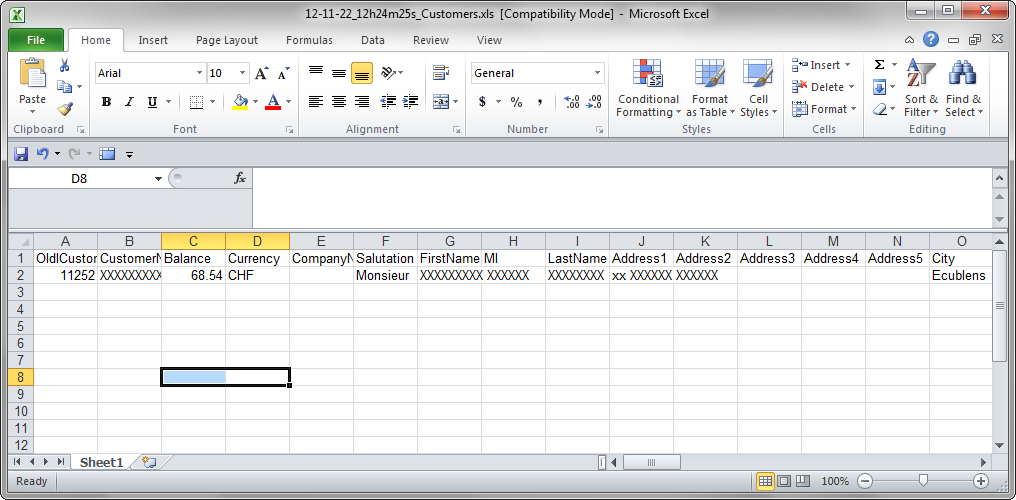


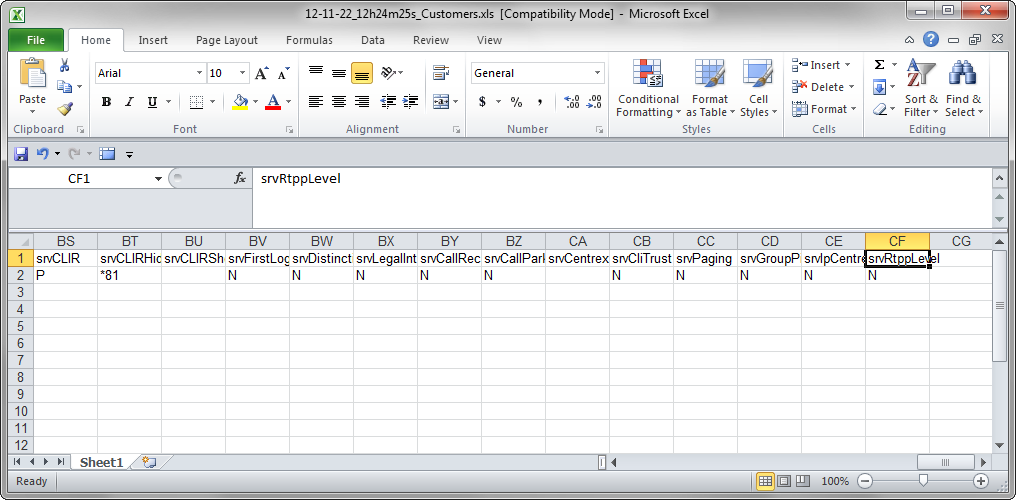
This is the only changes you have to make to importCustomers\_SwitzernetXls.cfg.

From Cygwin on your local computer, execute the command provided by the script:



This will locally download and open the new excel file. Check the values seem ok. The values must correspond to the column names, the first and last column must be the same as the first and last line of the list of importCustomers\_SwitzernetXls.cfg, all customers should be in the file, etc.

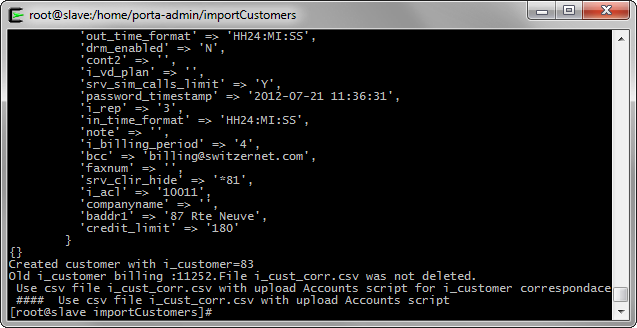




Once verified, go back to the slave in your working folder and execute the second command given as output of the downloading script in order to get the new excel file and uploading it.

|  |
| --- |
| [root@slave importCustomers]# cd /home/porta-admin/importCustomers;  scp root@pbs1.switzernet.com:/root/120822-customer-download/12-11-22\_12h24m25s\_Customers.xls .;  ./importCustomers\_SwitzernetXls.pl -v -x 12-11-22\_12h24m25s\_Customers.xls -c importCustomers\_SwitzernetXls.cfg |

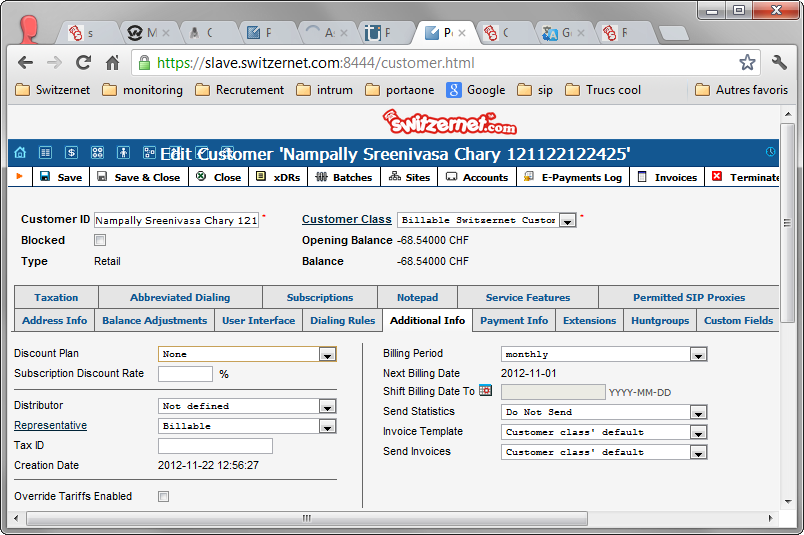
If everything has gone fine, the importCustomers\_SwitzernetXls.pl should have imported all customers of the 12-11-22\_12h24m25s\_Customers.xls and created a new file ‘i\_cust\_corr.csv’.



The ‘i\_cust\_corr.csv’ file is the link between the old i\_customer and new i\_customer in the two billings. It is essential for the next steps. Only one line is present in the sample file, as we only have one customer.



In the new porta-billig interface, you should now find the new customers you have created. In our example, the name of the customer has a suffix with a timestamp for testing.



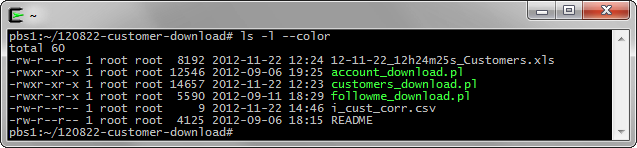
Verify the data from the web interface is the same as the old billing (with some exceptions in Service Features). If everything seems good, let’s upload the accounts for these customers.

## Import accounts

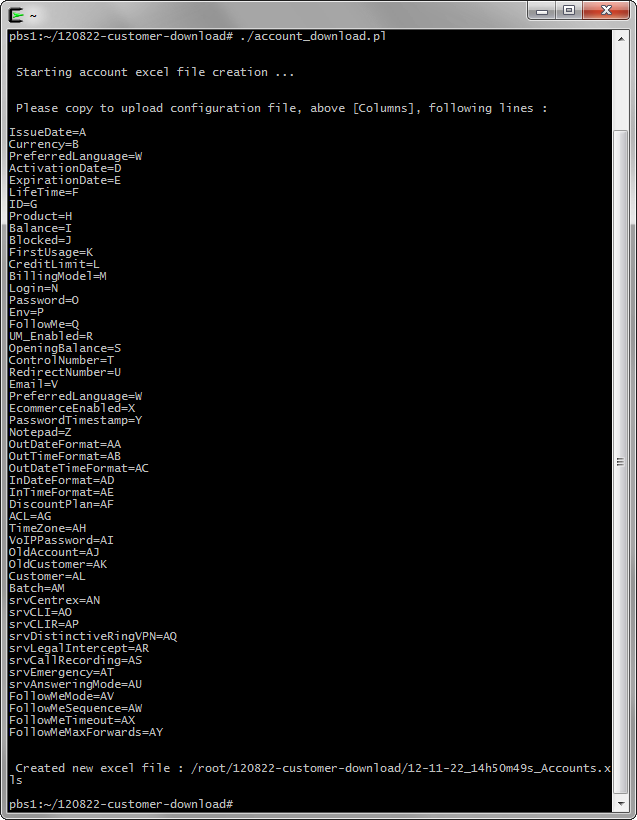
Go to your working folder on pbs1 and download the new created ‘i\_cust\_corr.csv’ at this place.

|  |
| --- |
| pbs1:~/120822-customer-download# scp switz@slave.switzernet.com:/home/porta-admin/importCustomers/i\_cust\_corr.csv .  switz@slave.switzernet.com's password:  i\_cust\_corr.csv 100% 9 0.0KB/s 00:00  pbs1:~/120822-customer-download# |

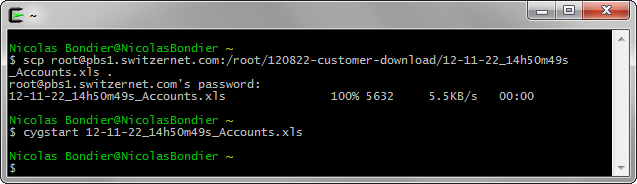
This is the list of files you should have in you working folder.

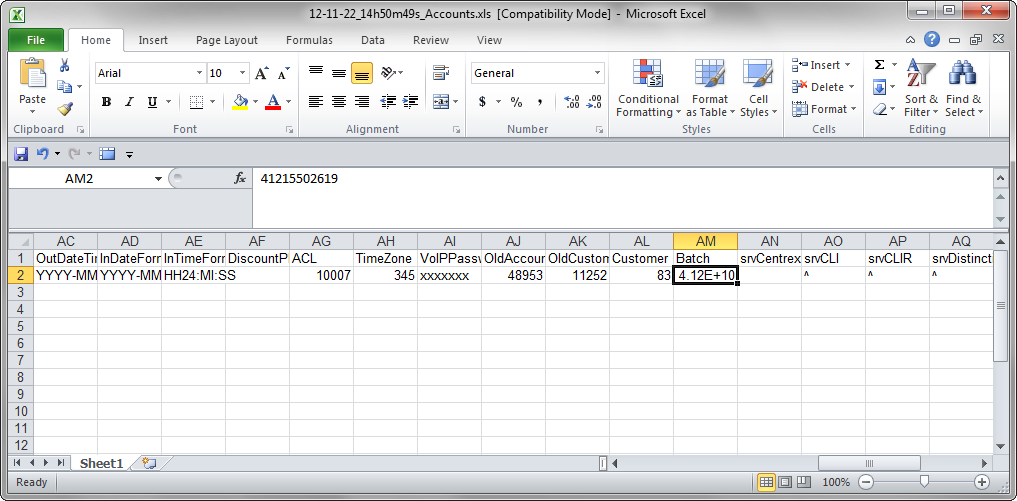
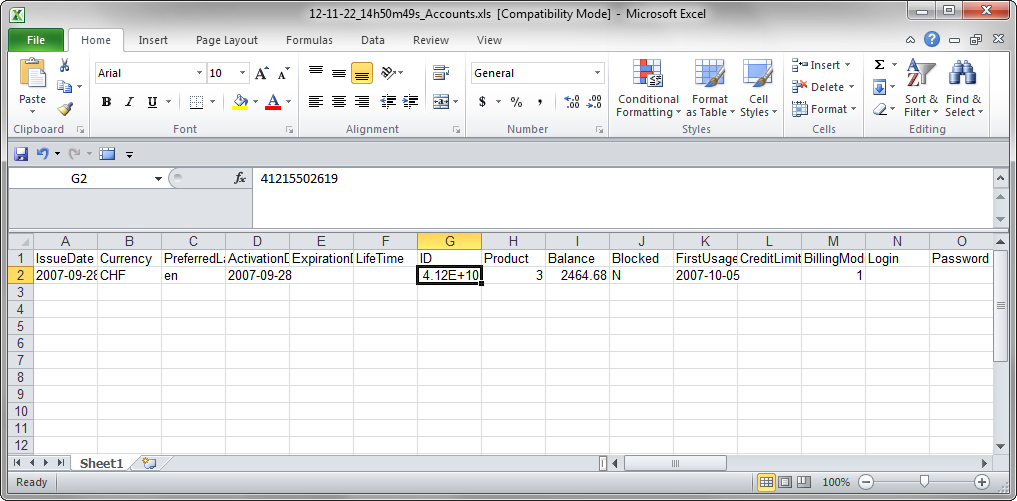


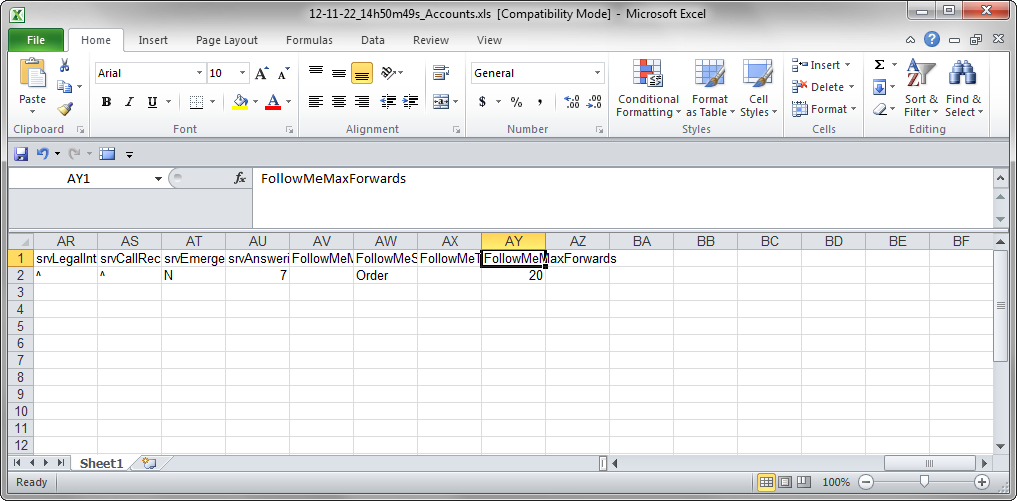
Once ‘i\_cust\_corr.csv’ copied, simply run the ‘./account\_download.pl’. It will read our new file, get all accounts data for the customers of ‘i\_cust\_corr.csv’ and create the new excel.



The file path is provided. Simply download to your desktop, open and verify if the excel is correct.

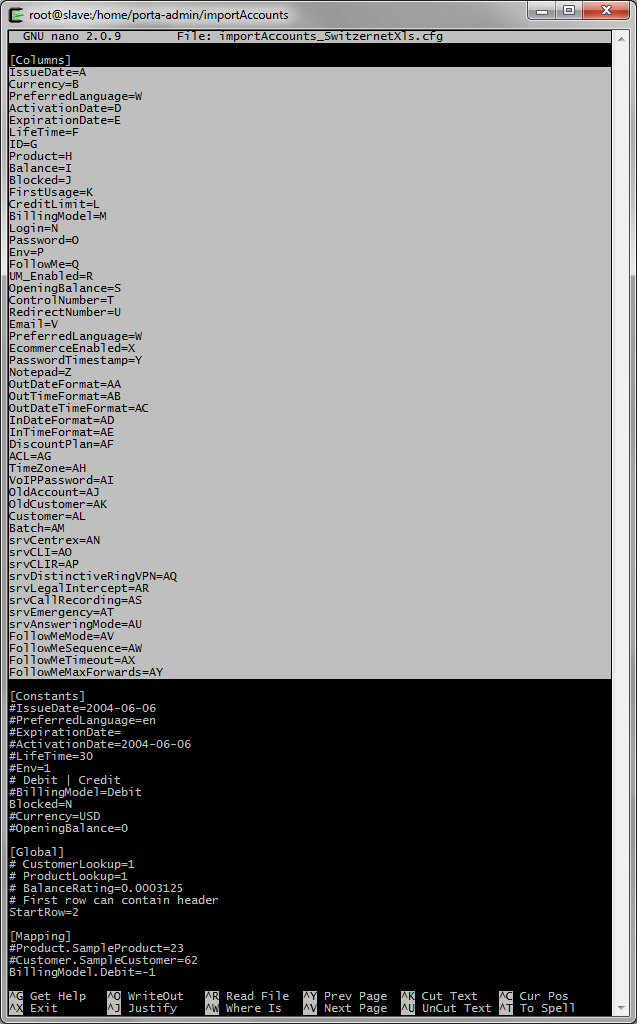




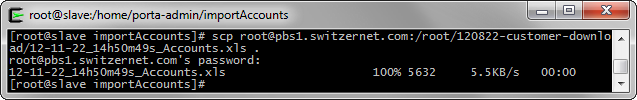


If the data seems correct, go to the slave and edit ‘importAccounts\_SwitzernetXls.cfg’. Replace the list under [Columns] with the list of column names provided by ‘./account\_download.pl’ script.

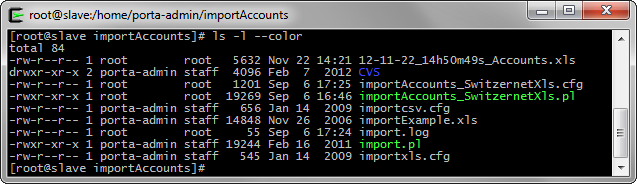
|  |
| --- |
| [root@slave importCustomers]# cd /home/porta-admin/importAccounts/  [root@slave importAccounts]# nano importAccounts\_SwitzernetXls.cfg |



Download now the accounts excel file in our working directory.



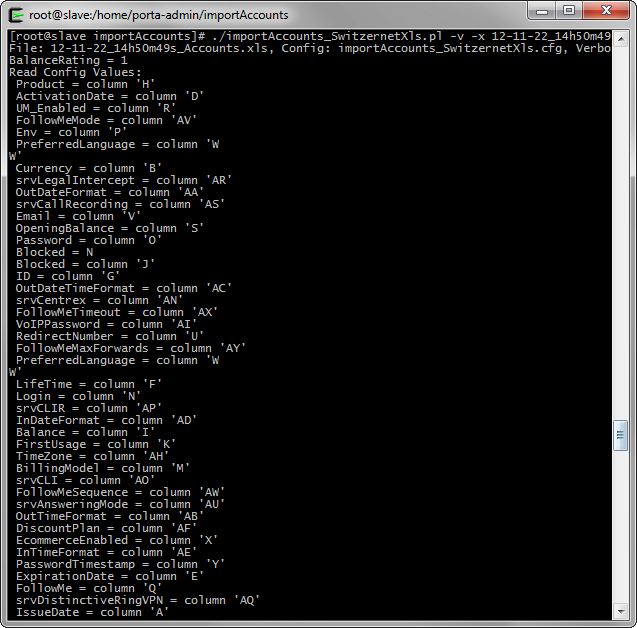
Here is a list of the files in the folder. We only use ‘importAccounts\_SwitzernetXls.cfg’, ‘importAccounts\_SwitzernetXls.pl’ and ‘12-11-22\_14h50m49s\_Accounts.xls’.

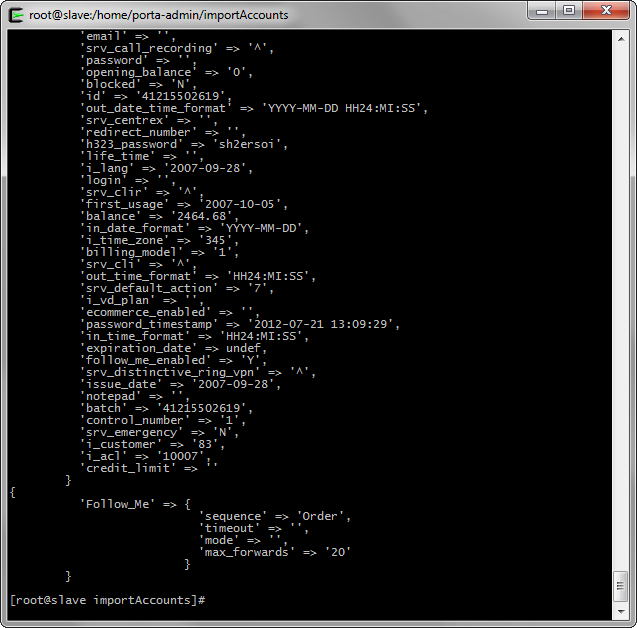


As you should have seen, the excel file contains all the data of accounts and the data for uploading, such as the new i\_customer.

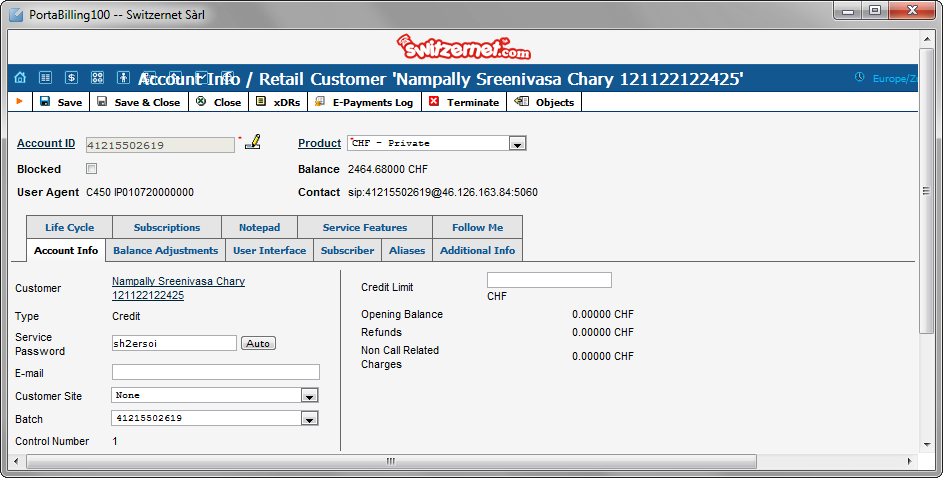
We only need to upload the new accounts. Run the following command with your new excel file :

|  |
| --- |
| ./importAccounts\_SwitzernetXls.pl -v -x 12-11-22\_14h50m49s\_Accounts.xls -c importAccounts\_SwitzernetXls.cfg |





If no error occurs while importing the data, directly go to the new web interface and check all the data. Only Follow Me and Subscription should be missing.



Note : On account creation on the new billing, if the account exist on both old and new billing, the dbas servers choose the new billing for authentication. The account has received a registration between the little interval I uploaded the account and the capture. This is why you can see the ‘Contact’ field is filled.

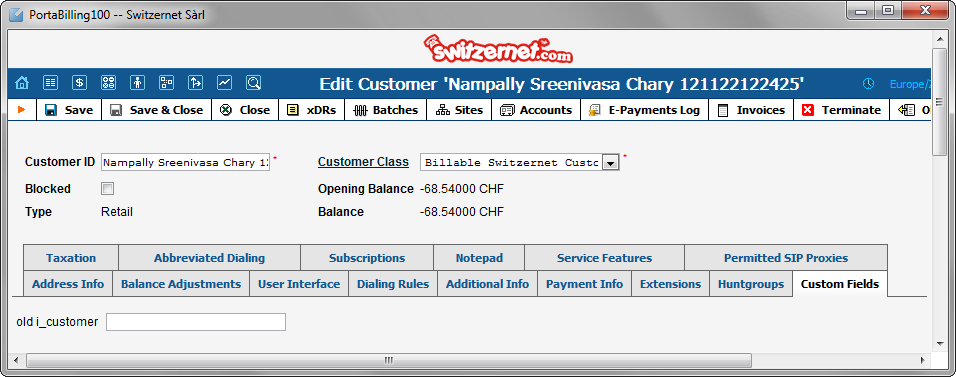
At this point the account is operational. We must continue the importation to provide other features like follow me, limit the maximum number of calls by account and add the subscriptions.

If you need to switch authorization and authentication from one billing to the other, use the following page:  
<http://switzernet.com/3/company/120820-acc-auth-to-billing/index.php>

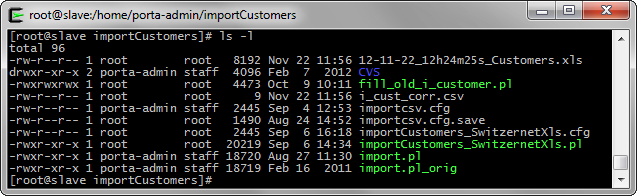
Note also that accounting is send to the 2 billing masters if the account is on both. This way, no data is lost.

## Import old i\_customer fields

We decided to implement a custom field in the porta-billing web interface. This field is ‘old i\_customer’. It will be the link between the old and new billing if we need it in the future. You can find it under any customer, under the ‘Custom Fields’ tab.

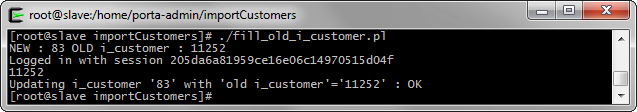


Go back to the ‘/home/porta-admin/importCustomers’ on the new slave. In this folder a script called ‘fill\_old\_i\_customer.pl’ is present.

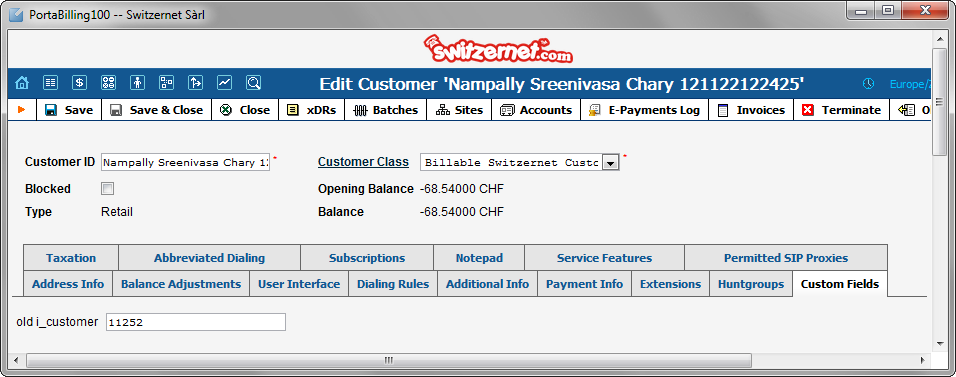


The script will use the ‘i\_cust\_corr.csv’ file and will upload the old i\_customer value to the custom field we have created. This is done through SOAP connection.

Just run the script:



The old i\_customer is now filled in the web interface:



## Import customer sites

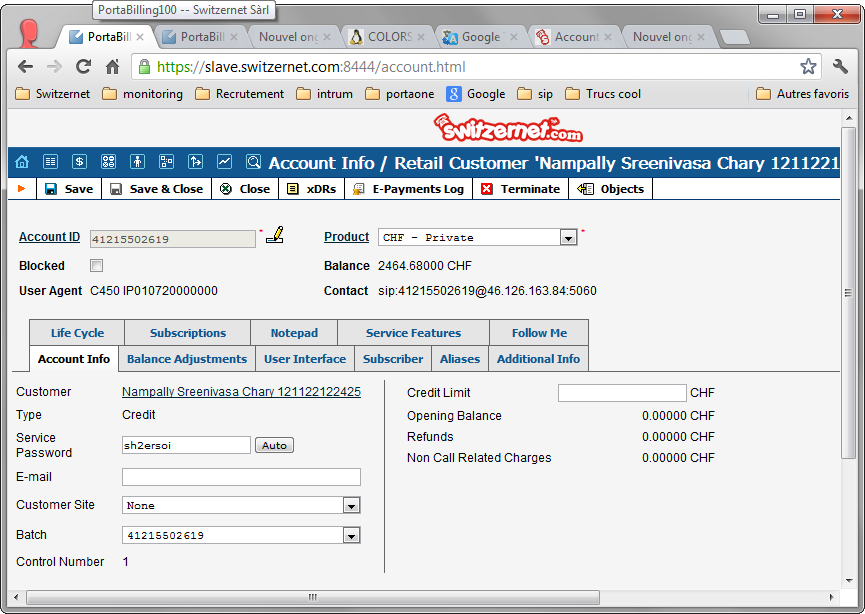
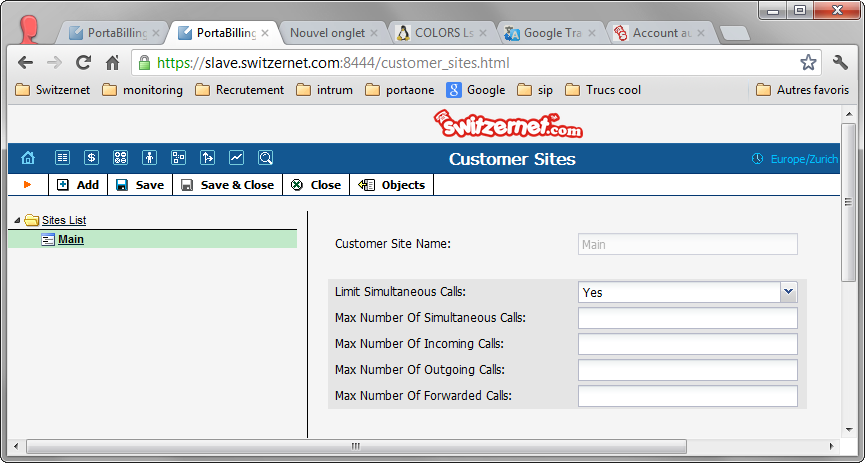
For limiting maximum simultaneous calls for each account, we decided to use the new Customers Sites feature as the limit for each customer as disappeared. This point is essential for fraud prevention and for meeting our product restrictions.

Change directory on the slave to ‘/home/porta-admin/importCustomerSites’. Only one script is present: ‘import.pl’.

It does not need any input file. This script checks every account on the new billing master. If the account has no assigned site, it creates one with default values for the product used.

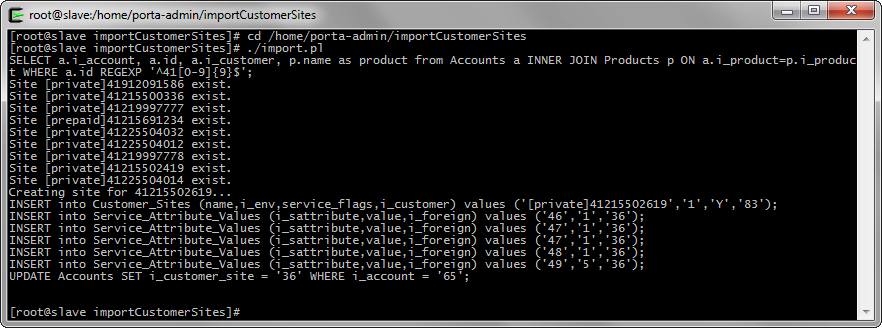
The advantage is that you can run this script every time you need to set a large quantity of customer’s sites to all accounts that do not have one, without looking for which accounts need to be updated with a new site. This script can be put in crontab if needed.

Here you can view the current state of our new customer’s sites:

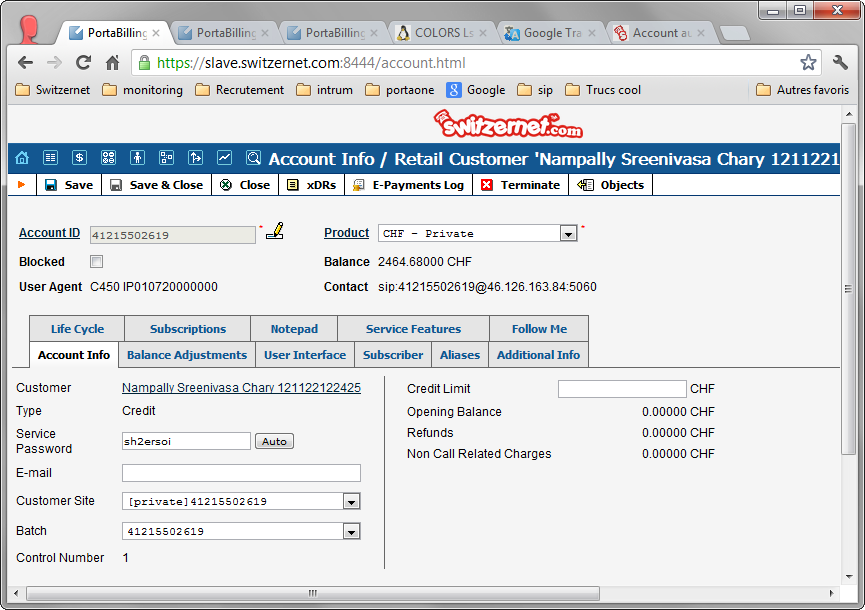
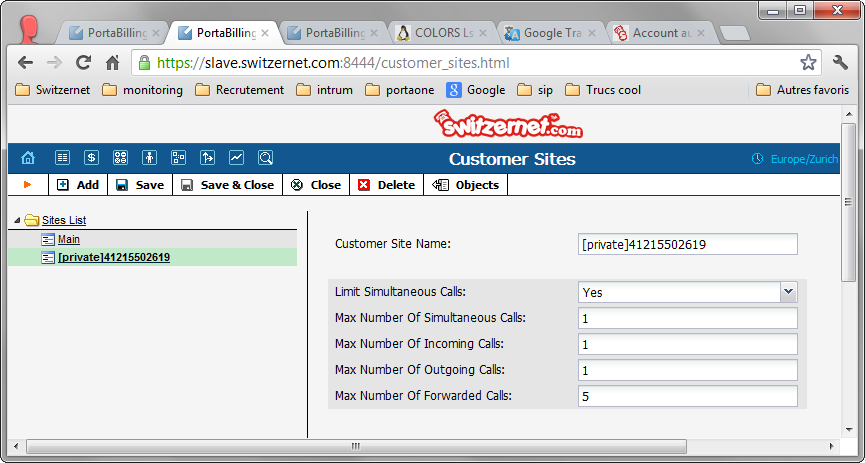


Launch the script ‘import.pl’:

|  |
| --- |
| [root@slave importCustomerSites]# cd /home/porta-admin/importCustomerSites  [root@slave importCustomerSites]# ./import.pl |



If everything is ok, go to the customers and account pages, you can view the changes:

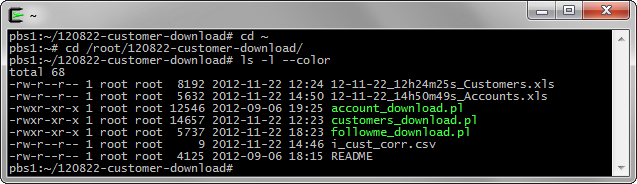


As you can view, the Customer has one site for each account with the type of site (private, business or prepaid). Note that sites for accounts with promotional product have the same name as for account with no promotional product.

## Import follow me

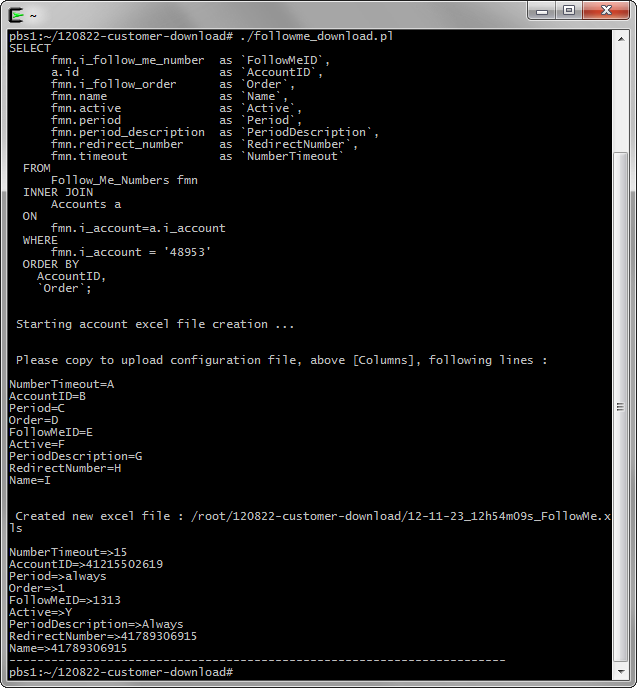
It is now time to import the follow-me. Go in the pbs1 work folder. The script to use this time is ‘followme\_download.pl’.

You still need to have the ‘i\_cust\_corr.csv’ in the folder, to know for which customer we need to get the data.



Run the ‘./followme\_download.pl’ script :

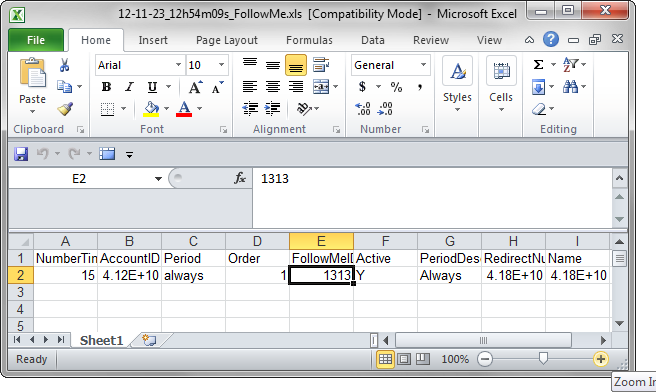
|  |
| --- |
| [root@slave importCustomerSites]# ./followme\_download.pl |



The script creates the Excel file of the customers’ Follow-Me and prints the new list of column for the configurations file.

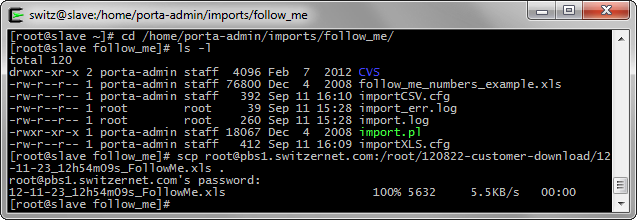
If no error occurs, open a local terminal, download and verify the new excel file:

|  |
| --- |
| Nicolas Bondier@NicolasBondier ~  $ scp root@pbs1.switzernet.com:/root/120822-customer-download/12-11-23\_12h54m09s\_FollowMe.xls .  root@pbs1.switzernet.com's password:  12-11-23\_12h54m09s\_FollowMe.xls 100% 5632 5.5KB/s 00:00  Nicolas Bondier@NicolasBondier ~  $ cygstart 12-11-23\_12h54m09s\_FollowMe.xls |

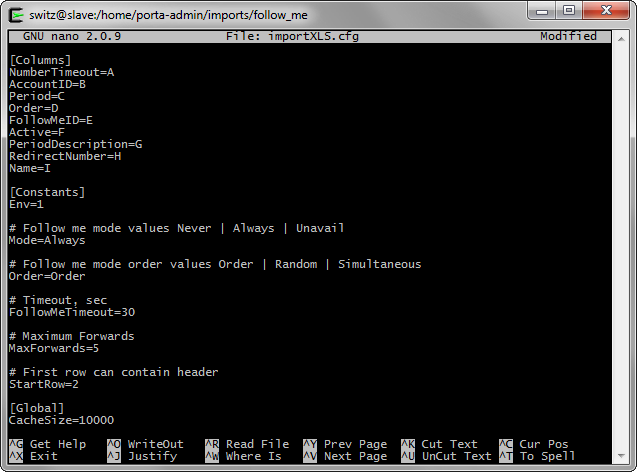


Once verified, go on slave and change directory to ‘/home/porta-admin/imports/follow\_me/’ and import your new excel file.

|  |
| --- |
| [root@slave ~]# cd /home/porta-admin/imports/follow\_me/  [root@slave follow\_me]# ls -l  total 120  drwxr-xr-x 2 porta-admin staff 4096 Feb 7 2012 **CVS**  -rw-r--r-- 1 porta-admin staff 76800 Dec 4 2008 follow\_me\_numbers\_example.xls  -rw-r--r-- 1 porta-admin staff 392 Sep 11 16:10 importCSV.cfg  -rw-r--r-- 1 root root 39 Sep 11 15:28 import\_err.log  -rw-r--r-- 1 root root 260 Sep 11 15:28 import.log  -rwxr-xr-x 1 porta-admin staff 18067 Dec 4 2008 **import.pl**  -rw-r--r-- 1 porta-admin staff 412 Sep 11 16:09 importXLS.cfg  [root@slave follow\_me]# scp root@pbs1.switzernet.com:/root/120822-customer-download/12-11-23\_12h54m09s\_FollowMe.xls .  root@pbs1.switzernet.com's password:  12-11-23\_12h54m09s\_FollowMe.xls 100% 5632 5.5KB/s 00:00  [root@slave follow\_me]# |

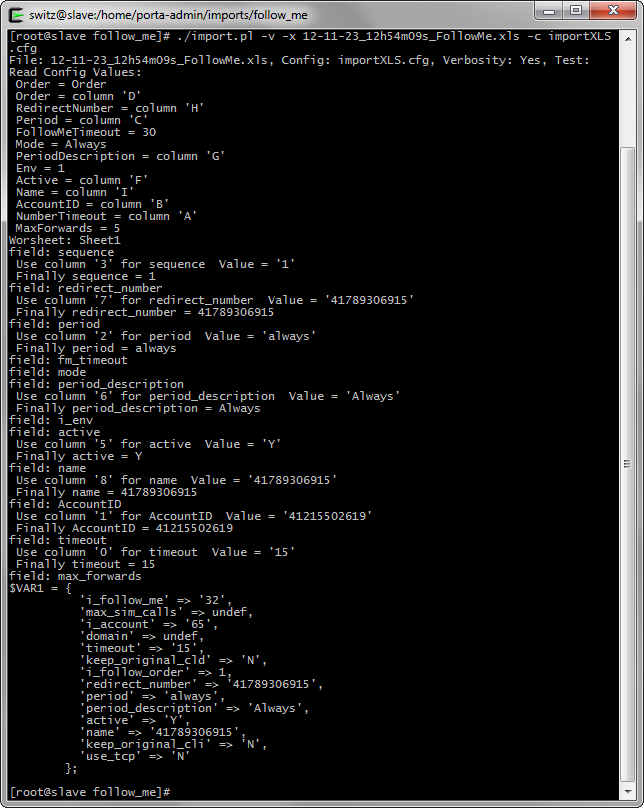


Open the ‘importXLS.cfg’ configuration file and past the column list under ‘[Columns]’:

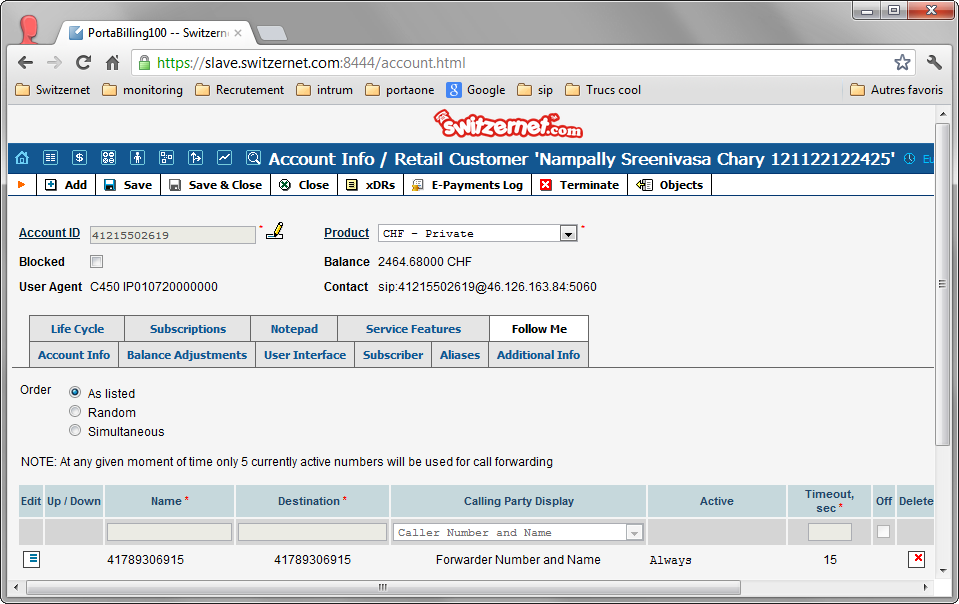


Verify all files are present and run the ‘import.pl’ script:

|  |
| --- |
| [root@slave follow\_me]# ./import.pl -v -x 12-11-23\_12h54m09s\_FollowMe.xls -c importXLS .cfg |



Inserting of the account’s Follow-Me is finished. We can see the new data in the new web interface:



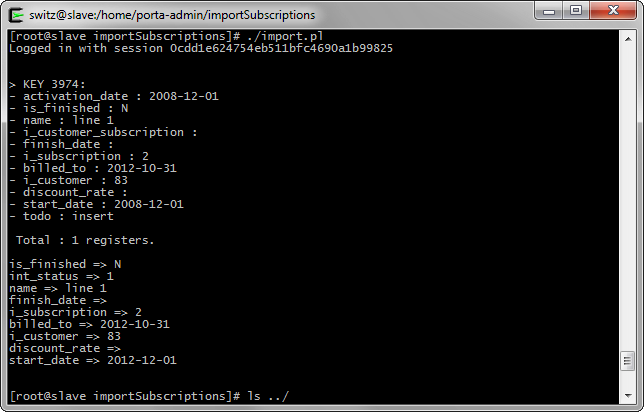
## Imports subscriptions

The last data to import are the subscription. For this part, we made our own scripts, using SOAP.

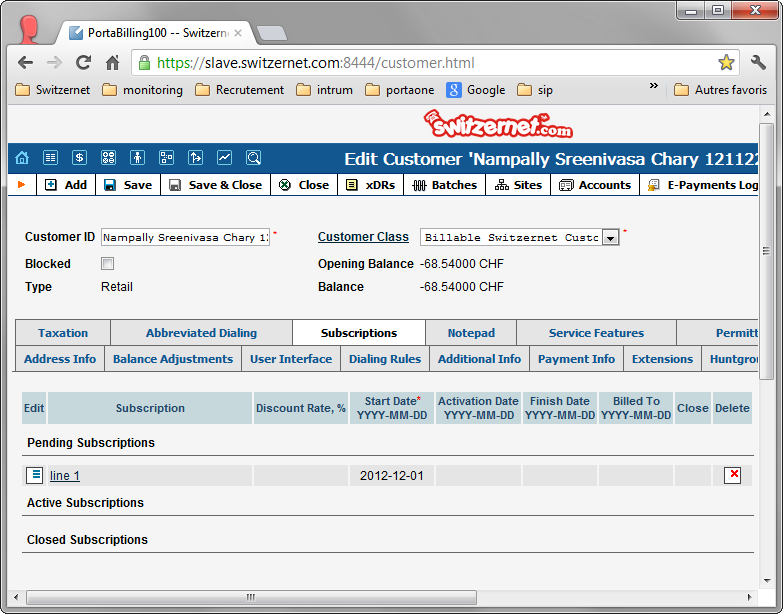
Please note that we cannot set the subscriptions in the past. We have to start it at the first next month.

The script checks the subscription for each customer who have the old i\_customer field filled. If the subscriptions differ, it synchronizes the subscriptions.

On slave server, change your directory to ‘/home/porta-admin/importSubscriptions/’. And run the ‘import.pl’ script:



And once the script has finished, you can view the new pending subscription:



Your customer importing is finished!

# References

The new version of this document:   
<http://switzernet.com/3/public/130305-import-customers-how-to/>

\* \* \*