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][html][docx]

* * *

Contents

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(rel2abs);`
`use File::Basename;`
`use Unicode::Map();`
`use Encode;`
`use List::Util qw(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 $dbh = DBI->connect( "dbi:mysql:dbname=$db;host=$host;", $user, $pass )`
`or die "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(@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(%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 "\nSelect a range of i_customer to import.\n";
  print "First i_customer:\n";
  my $first_c = <>;
  chomp($first_c);
  #print '\''.$first_c.'\'';
   while (  !isint($first c) ){
    print "Error ! Give a correct value.\n";
    print "First i_customer:\n";
    $first_c = <>;
    chomp($first_c);
  }
  print "\nLast i_customer:\n";
  my $last c = <>;
  chomp($last_c);

  while ( !$last c || !isint($last c) || $last c < $first c ){
    print "Error ! Give a correct value.\n";
    print "Last i_customer:\n";
    $last_c = <>;
    chomp($last_c);
  }

  sub isint{
    my $val = shift;
    return ($val =~ m/^\d+$/);
  }

  print "\nFirst i_customer : ".$first_c."\n";
  print "Last i_customer  :  ".$last_c."\n";
  return ($first_c,$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;
  my $stop = shift;
  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(@ret, $row[0]);
  }
  return @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,
```

This subroutine collects all required data from each customer and adds it in a hash.

In this hash, each key (ex: OldICustomer,

```
    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 DBI->errstr();
        $sth->execute() or die DBI->errstr();
        @cols = @{ $sth->{NAME} };
        $hash{$i cus[$n]} = $dbh->selectrow hashref($req);
        $n++;
  }
```

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.

```
  return %hash;
  }
  sub add_services {
    my %hash = @_;

    push (@cols, 'srvSimcallsLimit');
    for my $c ( sort keys %hash ) {
      $hash{$c}{'srvSimcallsLimit'} = get_srvSimcallsLimit($c);
    }

    push (@cols, 'srvCLI');
    for my $c ( sort keys %hash ) {
      $hash{$c}{'srvCLI'} = get_srvCLI($c);
    }

    push (@cols, 'srvCLIR');
    for my $c ( sort keys %hash ) {
      $hash{$c}{'srvCLIR'} = get_srvCLIR($c);
    }

    push (@cols, 'srvCLIRHide');
    for my $c ( sort keys %hash ) {
      $hash{$c}{'srvCLIRHide'} = get_srvCLIRHide($c);
    }

    push (@cols, 'srvCLIRShow');
    for my $c ( sort keys %hash ) {
      $hash{$c}{'srvCLIRShow'} = get_srvCLIRShow($c);
    }

    push (@cols, 'srvFirstLoginGreeting');
    for my $c ( sort keys %hash ) {
      $hash{$c}{'srvFirstLoginGreeting'} = get_srvFirstLoginGreeting($c);
    }

    push (@cols, 'srvDistinctiveRingVpn');
    for my $c ( sort keys %hash ) {
      $hash{$c}{'srvDistinctiveRingVpn'} = get_srvDistinctiveRingVpn($c);
    }

    push (@cols, 'srvLegalIntercept');
    for my $c ( sort keys %hash ) {
      $hash{$c}{'srvLegalIntercept'} = get_srvLegalIntercept($c);
    }

    push (@cols, 'srvCallRecording');
    for my $c ( sort keys %hash ) {
      $hash{$c}{'srvCallRecording'} = get_srvCallRecording($c);
    }

    push (@cols, 'srvCallParking');
    for my $c ( sort keys %hash ) {
      $hash{$c}{'srvCallParking'} = get_srvCallParking($c);
    }

    push (@cols, 'srvCentrex');
    for my $c ( sort keys %hash ) {
      $hash{$c}{'srvCentrex'} = get_srvCentrex($c);
    }

    push (@cols, 'srvCliTrust');
    for my $c ( sort keys %hash ) {
      $hash{$c}{'srvCliTrust'} = get_srvCliTrust($c);
    }

    push (@cols, 'srvPaging');
    for my $c ( sort keys %hash ) {
      $hash{$c}{'srvPaging'} = get_srvPaging($c);
    }

    push (@cols, 'srvGroupPickup');
    for my $c ( sort keys %hash ) {
      $hash{$c}{'srvGroupPickup'} = get_srvGroupPickup($c);
    }

    push (@cols, 'srvIpCentrexCare');
```

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.

<pre>for my $c (sort keys %hash) {
  $hash{$c}{'srvIpCentrexCare'} = get_srvIpCentrexCare($c);
}
 	
push (@cols, 'srvRtppLevel');
for my $c (sort keys %hash) {
  $hash{$c}{'srvRtppLevel'} = get_srvRtppLevel($c);
}
 	
%hash = set_defaults_values(%hash);
return %hash;
}</pre>	

<pre>sub get_srvSimcallsLimit {
  my $i_cus = shift;
  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 $ret;
}</pre>
	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.

<pre>sub get_srvCLI {
my $i_cus = shift;
  my $req = "select service_flags from Customers where i_customer = ".$i_cus." LIMIT 1;";
  my $v = $dbh->selectrow_array($req, undef);
  $v = substr($v, 2, 1);
  $v = 'A' if (!$v || $v ne 'Y');
  return $v;
}</pre>
	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.

<pre>sub get_srvCLIR {
  return 'P';
}</pre>
	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 '*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]
`sub get_srvCLIRShow { return ''; }`	Default is showing, we do not need a prefix for showing CLI.
`sub get_srvFirstLoginGreeting { return 'N'; }`	As we are uploading customer who are not new, I disabled this feature.
`sub get_srvDistinctiveRingVpn { return 'N'; }`	We do not support. This is for a distinctive ring when receiving call from an external network.
`sub get_srvLegalIntercept { return 'N'; }`	No legal intercept.
`sub get_srvCallRecording { return 'N'; }`	No recording service is implemented on Astrad servers as far.
`sub get_srvCallParking { return 'N'; }`	Not implemented.
`sub get_srvCentrex { my $i_cus = shift; 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); $v = '' if (!$v); return $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 'N'; }`	Correspond to the 'Accept Caller Identity' setting in service feature. This is not used by our Astrad

	servers.

`sub get_srvPaging {`
`  return 'N';`
`}`
	Paging service disabled.

`sub get_srvGroupPickup {`
`  return 'N';`
`}`
	Not implemented in Astrad servers.

`sub get_srvIpCentrexCare {`
`  return '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 'N';`
`}`
	Deactivation of RTP proxy.

`sub set_defaults_values {`
`  my %hash = @_;`
`  # Access list`
`  for my $c ( sort keys %hash ) {`
`  $hash{$c}{'ACL'} = '10011';`
`  }`
`  # Customer Class`
`  for my $c ( sort keys %hash ) {`
`  $hash{$c}{'CustomerClass'} = '2';`
`  }`
`  return %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(time);`
`  my $filename = strftime('%y-%m-%d_%Hh%Mm%Ss',localtime).'_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 keys %hash ) {`
`  $x = 0;`
`  foreach (@cols){`
`  if ( $y == 1 ){ $worksheet->write(0, $x, $_); $columns{$_} = get_column_AA($x+1);}`
`  $temp = decode("utf8", $hash{$c}{$_});`
`  $temp = '' if ( !$temp || !defined($temp) );`
`  #print $temp;`
`  $temp = test_mode( $temp, $_ ) if $test_mode;`
`  $worksheet->write($y, $x, $temp);`
`  $x++;`
`  }`
`  $y++;`
`  }`
 	
`  print "\n\n";`
`  foreach (@cols){`
`  print $_."=".$columns{$_}."\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

Code	Comments

<pre>   print "\n\n # Copy the list above in the importCustomers_SwitzernetXls.cfg
configuration file in the slave \n";
   print "\n # Download and verify created excel file :";
   print "\n # scp root\@pbs1.switzernet.com:".$dirname."/".$filename." .
; cygstart ./$filename \n";
   print "\n # Import in new master (to execute on slave) :";
   print "\n # cd /home/porta-admin/importCustomers; scp
root\@pbs1.switzernet.com:".$dirname."/".$filename." .;
./importCustomers_SwitzernetXls.pl -v -x ".$filename." -c
importCustomers_SwitzernetXls.cfg\n";
 	
   }</pre>
	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
	…

<pre> sub test_mode {
   my $par1 = shift;
   my $par2 = shift;
   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 $par1;
 }</pre>
	A simple function that some of values unique for testing.
	
	This is important to do not have duplicates values that can break the upload.

<pre> sub get_column_AA {
   my $index = shift;
   my $col = int2latin($index);
   $col = uc $col;
   return $col;
 }</pre>
	Return ‘A’ for 1, ‘B’ for 2, …

[full_version]

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].

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
<pre>my $old_i_customer = '';</pre>	Setting a global variable old_i_customer for further use.

<pre>my $i_cus_corr_file = 'i_cust_corr.csv';
if (unlink($i_cus_corr_file) == 0) {
    print "File ".$i_cus_corr_file." deleted successfully.";
} else {
    print "File ".$i_cus_corr_file." was not deleted.";
}</pre>
	Checking if the file of correspondences exists. If yes deleting it in order to create a

	new one.
old i customer => q{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 $object{$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, $subtables) = @_;
print STDERR "Created customer with i_customer=$customer->{__i_object}\n" if $op->{verbose};
print STDERR "Old i_customer billing :".$old_i_customer.".";
 	
open (BILL_CORR, '>>'.$i_cus_corr_file);
print BILL_CORR $customer->{__i_object}.",".$old_i_customer."\n";
close (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]

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(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 = "xxxxxxxxxxx"; my $user = "xxxxxxxxxxx"; my $pass = "xxxxxxxxxxx"; my $dbh = DBI->connect("dbi:mysql:dbname=$db;host=$host;", $user, $pass) or die "Connexion impossible à la base de données $db !"; my $i_customer_file = 'i_cust_corr.csv';	Connection to the database and initialization of variables.

```
my @cols;
my $init_col = 0;
```

```
# Creating hash of accounts
my %accounts_list = get_account_list($i_customer_file);
%accounts_list = verify_accounts(%accounts_list);

my %accounts = get_account_data(%accounts_list);

# Fixing id for external tables
%accounts = get_new_i_product(%accounts);
%accounts = get_new_i_access_level(%accounts);

# Adding all other fields
%accounts = get_batch(%accounts);
%accounts = get_srvCentrex(%accounts);
%accounts = get_srvCLI(%accounts);
%accounts = get_srvCLIR(%accounts);
%accounts = get_srvDistinctiveRingVPN(%accounts); %accounts =
get_srvLegalIntercept(%accounts);
%accounts = get_srvCallRecording(%accounts);
%accounts = get_srvEmergency(%accounts);
%accounts = get_srvAnsweringMode(%accounts);
%accounts = get_FollowMeMode(%accounts);
%accounts = get_FollowMeSequence(%accounts);
%accounts = get_FollowMeTimeout(%accounts);
%accounts = get_FollowMeMaxForwards(%accounts);


# Write to excel file
to_excel(%accounts) if $write_to_excel;

# printing the list
print_hash_ref(%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;
  my $csv = Text::CSV->new();
  my %corr;
  my $req = '';

  open (CSV, "<", $file) or die $!;
  while (<CSV>) {
      if ($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 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 "Failed to parse line: $err";
      }
  }
  close CSV;
  return %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,
```

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.

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 %i_acc){
my $req = $req1.$k."';";
my $sth = $dbh->prepare($req) or die DBI->errstr();
$sth->execute() or die DBI->errstr();
$hash{$k} = $dbh->selectrow_hashref($req);
if ($init_col == 0) {
@cols = @{ $sth->{NAME} }; $init col++;
}
%hash = add field and value($k,'Customer',$i acc{$k},%hash);
}
return %hash;
}	

sub print_hash_ref {
my %hash = @_;
for my $c (sort keys %hash) {
foreach (@cols){
$hash{$c}{$_} = '' if (!$hash{$c}{$_});
print $_ . "=>" . $hash{$c}{$_} . "\n";
}
print "----------
------\n";
}
}
	Printing the main account hash reference to view all values.

sub get_batch {
my %acc = @ ;
for my $c (sort keys %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);
%acc = add field and value ($c,'Batch',$v,%acc);
}
return %acc;
}
	We get account's batch.

sub get_srvCLI {	Accounts have

`my %acc = @_;`
`for my $c ( sort keys %acc ) {`
`my $req = "select service_flags from Accounts where i_account = ".$acc{$c}{'OldAccount'}." LIMIT 1;";`
`my $v = $dbh->selectrow_array($req, undef);`
`$v = substr($v, 1, 1);`
`$v = 'A' if (!$v || ($v ne 'Y' && $v ne '^'));`
`%acc = add field and value ($c,'srvCLI',$v,%acc);`
`}`
`return %acc;`
`}`
	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 keys %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);`
`$v = '' if (!$v);`
`%acc = add field and value ($c,'srvCentrex',$v,%acc);`
`}`
`return %acc; }`
	srvCentrex is the CLI to display if is different than account ID.

`sub get_srvCLIR {`
`my %acc = @_;`
`for my $c ( sort keys %acc ) {`
`%acc = add field and value ($c,'srvCLIR','^',%acc);`
`}`
`return %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 $c ( sort keys %acc ) {`
`%acc = add field and value ($c,'srvDistinctiveRingVPN','^',%acc);`
`}`
`return %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 $c ( sort keys %acc ) {`
`%acc = add field and value ($c,'srvLegalIntercept','^',%acc);`
`}`
`return %acc;`
`}`
	Astrad servers do not support this feature.
	We refer to customer level.

`sub get_srvCallRecording {`
`my %acc = @_;`
`for my $c ( sort keys %acc ) {`
`%acc = add field and value ($c,'srvCallRecording','^',%acc);`
`}`
`return %acc;`
`}`
	No call recording.
	We refer to customer level.

`sub get_srvEmergency {`
`my %acc = @_;`
`for my $c ( sort keys %acc ) {`
	Not activated.

`%acc = add_field_and_value ($c,'srvEmergency','N',%acc);`
`}`
`return %acc;`
`}`	

`sub get_srvAnsweringMode {`
`my %acc = @_;`
`for my $c ( sort keys %acc ) {`
`%acc = add_field_and_value ($c,'srvAnsweringMode','7',%acc);`
`# TODO :`
`# test with no voicemail : %acc = add_field_and_value ($c,'srvAnsweringMode','3',%acc);`
`}`
`return %acc; }`
	Voicemail answer.

`sub get_FollowMeMode {`
`my %acc = @_;`
`for my $c ( sort keys %acc ) {`
`%acc = add_field_and_value ($c,'FollowMeMode','',%acc);`
`}`
`return %acc;`
`}`
	The value of the follow me modes are still the same, always, never, … We just select.

`sub get_FollowMeSequence {`
`my %acc = @_;`
`for my $c ( sort keys %acc ) {`
`%acc = add_field_and_value ($c,'FollowMeSequence','Order',%acc);`
`}`
`return %acc;`
`}`
	Sequence value does not change, we keep the same.

`sub get_FollowMeTimeout {`
`my %acc = @_;`
`for my $c ( sort keys %acc ) {`
`my $req = "select timeout from Follow_Me where i_account = '".$acc{$c}{'OldAccount'}."' LIMIT 1;";`
`my $v = $dbh->selectrow_array($req, undef);`
`%acc = add_field_and_value ($c,'FollowMeTimeout',$v,%acc);`
`}`
`return %acc;`
`}`
	Timeout before going to the followme, in seconds.

`sub get_FollowMeMaxForwards {`
`my %acc = @_;`
`for my $c ( sort keys %acc ) {`
`%acc = add_field_and_value ($c,'FollowMeMaxForwards','20',%acc);`
`}`
`return %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 $c ( sort keys %acc ) {`
`%acc = add_field_and_value ($c,'Product',$prods->{$acc{$c}{'Product'}},%acc);`
`} return %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 $c ( sort keys %acc ) {`
`%acc = add_field_and_value ($c,'ACL','10007',%acc);`
`}`
`return %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;
  my $n = shift;
  my $v = shift;
  my %acc = @_;
  # print "| account : ".$a."|\n";
  # print "| name : ".$n."|\n";
  # print "| value : ".$v."|\n";
 	
  $acc{$a}{$n} = $v;
  if (!exist_in_array($n,@cols)){
  push (@cols,$n);
  }
  return %acc;
}
	Subroutine for adding values to the hash more quickly.

sub exist_in_array {
  my $value = shift;
  my @array = @ ;
  my $ret = 0;
  my %hash;
  %hash = map { $_ => 1 } @array;
  if ($hash{$value}){
  $ret = 1;
  }
  return $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(time);
  my $filename = strftime('%y-%m-%d_%Hh%Mm%Ss',localtime).'_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 "\n\n Starting account excel file creation ... \n\n";
 	
  for my $c (sort keys %hash) {
  $x = 0;
  foreach (@cols){
  if ($y == 1){ $worksheet->write(0, $x, $_); $columns{$_} = get_column_AA($x+1);}
  $temp = decode("utf8", $hash{$c}{$_});
  $temp = '' if (!$temp || !defined($temp));
  $temp = test_mode($temp, $_) if $test_mode;
  $worksheet->write($y, $x, $temp);
  $x++;
  }
  $y++;
  }
  print "\n Please copy to upload configuration file, above [Columns], following lines :\n\n";
  foreach (@cols){
  print $_."=".$columns{$_}."\n";
  }
  print "\n\n Created new excel file : ".$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;
  my $col = int2latin($index);
  $col = uc $col;
  return $col;
}
	Return ‘A’ for 1, ‘B’ for 2, …

sub test_mode {
  # For formating a field if testing
  my $par1 = shift;
  my $par2 = shift;
  # if ($par2 eq 'CustomerName'){
	Testing options. This function has also been used in the customer

`    # $par1 = $par1.'['.`date +%y%m%d%H%M%S`.']';`
`  # }`
`  return $par1;`
`}`
	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 $c ( sort keys %acc ) {`
`  if ($centrex_i_acc{$c}){`
`  delete $acc{$c} if (!$prefs{substr($centrex_i_acc{$c},0,7)});`
`  }`
`  }`
`  return %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]

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] and this one [III] 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 $customers_corr = get_customer_list($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="xxxxxxxxxxx";`
`my $host="xxxxxxxxxxx";`
`my $user = "xxxxxxxxxxx";`
`my $pass = "xxxxxxxxxxx";`
`my $dbh = DBI->connect( "dbi:mysql:dbname=$db;host=$host;", $user, $pass )`
`or die "Connexion impossible à la base de données $db !";`
	Setting the mysql connection to the new master server.

`binmode(STDOUT, ':utf8');`
`my $proxy_host = 'https://xxxxxxxxxxx'; # Porta-Billing Admin Server`
`my $proxy_port = 'xxxxxxxxxxx';`
`my $login = 'xxxxxxxxxxx';`
`my $password = 'xxxxxxxxxxx';`
`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 "SOAP Fault: $!, " . (ref $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');`
	Setting connection to the SOAP interface. Here we can access to Sessions, Customers, and Account.

`$account service->serializer()`
`  ->xmlschema('http://www.w3.org/2001/XMLSchema');`
`my $LoginResponse = $session_service->login($login, $password);`
`my $session_id = $LoginResponse->result();`
`print "Logged in with session $session_id\n";`
`my $header = SOAP::Header->name('auth info')->value({ session id => $session id });`	

`foreach ( @$customers_corr ){`
`  if (customer_exist_in_new_billing($_->{i_customer})){`
`  my $hash = {`
`  i_customer => $_->{i_customer},`
`  old_i_customer => $_->{old_i_customer}`
`  };`
`  update_custom_old_i_customer_field($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;`
 	
`  my $GetCustomerCustomFieldsValuesRequest = {`
`  i customer => $i custs->{i customer}`
`  };`
 	
`  my $GetCustomerCustomFieldsValuesResponse = $customer service->get custom fields values($header, $GetCustomerCustomFieldsValuesRequest)->result;`
`  if (! $GetCustomerCustomFieldsValuesResponse) {`
`  die "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 $UpdateCustomerCustomFieldsValuesResponse->{'custom_fields_values'}->[0]->{text_value}."\n";`
`  print "Updating i_customer \'".$i_custs->{i_customer}."\' with \'old i_customer\'=\'".$i_custs->{old_i_customer}."\' : OK\n"`
`  } else {`
`  print "Updating i_customer \'".$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;`
`  my $csv = Text::CSV->new();`
`  my $arr = [];`
`  my $i = 0;`
 	
`  open (CSV, "<", $file) or die $!;`
`  while (<CSV>) {`
`  if ($csv->parse($_)) {`
`  my @columns = $csv->fields();`
`  print "NEW : ".$columns[0]." OLD i_customer : ".$columns[1]."\n";`
`  $arr->[$i]->{i_customer} = $columns[0];`
`  $arr->[$i]->{old_i_customer} = $columns[1];`
	Parsing the excel file containing correspondences between old and new i_customer. (Format : new_i_customer1, old_i_customer1 new_i_customer2,

`        $i++;`
`      } else {`
`        my $err = $csv->error_input;`
`        print "Failed to parse line: $err";`
`      }`
`  }`
`  close CSV;`
`  return $arr;`
`}`
	old_i_customer2
	… , …
	)

`sub customer_exist_in_new_billing {`
`  my $i_cus = shift;`
`  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 $ret;`
`}`
	Checking if the i_customer exist in new billing.

`if ($@) {`
`  print "An error occured: $@\n";`
`}`
`# logging out is required`
`$session_service->logout($session_id);`
`exit 0;`
	Loging out of SOAP.

[full version]

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 $account_data = get_accounts();`
`$account_data = set_limits($account_data);`
`main_site_upload($account_data);`
`print_list($account_data);`
`$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="xxxxxxxxxxxxxxxx";`
`my $host="xxxxxxxxxxxxxxxx";`
`my $user = "xxxxxxxxxxxxxxxx";`
`my $pass = "xxxxxxxxxxxxxxxx";`
`my $dbh = DBI->connect( "dbi:mysql:dbname=$db;host=$host;", $user, $pass ) or die "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`	Select i_account, numer and i_customer from

REGEXP \'^41[0-9]{9}$\';';
  my $sth = $dbh->prepare($req);
  $sth->execute();
  my $h = $sth->fetchall_hashref('i_account');
  print $req."\n";
  if ($test mode){
  for (keys %$h) {
  delete $h->{$ };
  }
  }
  return $h;
}
	Swiss accounts (Do not take Verizon Inbound for ex) and returning a hash with the data.

sub set_limits {
  my $h = shift;
  foreach my $k (keys(%$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 $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;
  my $cur_site = '';
  foreach my $k (keys(%$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 ($cur_site && 0){ # FOR UPDATE. BUT DISABLED BECAUSE Service_Attribute_Values MUST BE UPDATED TOO. OR IT WILL BREAK EVERYTHING
  update customer site name("[".$default_prefixe_name->{$h->{$k}->{'product'}}."]".$h->{$k}->{'id'}, $cur site);
  } else {
  $cur_site = add new customer site("[".$default_prefixe_name->{$h->{$k}->{'product'}}."]".$h->{$k}->{'id'}, $h->{$k}->{'i_customer'});
  }
  $h->{$k}->{'i_customer_site'} = $cur_site;
  insert service attribute values($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($h->{$k}->{'i_account'},$cur_site);
  print "\n";
  } elsif (! $cur name) {
  print "Creating site for $h->{$k}->{'id'}...\n";
  $cur_site = add new customer site("[".$default_prefixe_name->{$h->{$k}->{'product'}}."]".$h->{$k}->{'id'}, $h->{$k}->{'i_customer'});
  $h->{$k}->{'i_customer_site'} = $cur_site;
  insert service attribute values($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($h->{$k}->{'i_account'},$cur_site);
  print "\n";
  } else {
  print "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;
  my $ics = '';
  my $req = 'SELECT i_customer_site FROM Accounts WHERE id = \''.$id.'\'';
  $ics = $dbh->selectrow_array($req);
  return $ics;
}
	Getting the current customer site selected on an account.

sub add_new_customer_site {
  my $name = shift;
  my $i_customer = shift;
	Creation of the customer site, with the

`my $req = '';`
`$req = 'INSERT into Customer Sites (name,i env,service flags,i customer) values (\''.$name.'\',\'1\',\'Y\',\''.$i_customer.'\');';`
`print $req."\n";`
`$dbh->do($req);`
`my $new i cust site = $dbh->last insert id(undef, undef, qw(Customer Sites i customer site));`
`return $new i cust site;`
`}`
	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;`
`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 $req."\n";`
`$req = 'INSERT into Service_Attribute_Values (i_sattribute,value,i_foreign) values (\'47\',\''.$params[1].'\',\''.$i_cus_site.'\');';`
`print $req."\n";`
`$sth = $dbh->prepare($req);`
`$sth->execute();`
`print $req."\n";`
`$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 $req."\n";`
`$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 $req."\n";`
`return 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 $name = shift;`
`print "\n NAME : ".$name."\n";`
`my $i site = shift;`
`my $req = 'UPDATE Customer_Sites SET name = \''.$name.'\' WHERE i_customer_site = \''.$i_site.'\';';`
`my $sth = $dbh->prepare($req);`
`$sth->execute();`
`print $req."\n" if ($verbose);`
`}`
	Update a customer site name if the name doesn't correspond to the template.

`sub update_account_customer_site {`
`my $i_acc = shift;`
`my $i_site = shift;`
`my $req = 'UPDATE Accounts SET i_customer_site = \''.$i_site.'\' WHERE i_account = \''.$i_acc.'\';';`
`my $sth = $dbh->prepare($req);`
`$sth->execute();`
`print $req."\n" if ($verbose);`
`}`
	Update the account's to set the new i_customer_site to it.

`sub print_list {`
`my $h = shift;`
`my $c = 0;`
`foreach my $k ( keys( %$h ) ) {`
`print "\n> I_ACCOUNT ".$k.":\n";`
`foreach my $k2 ( keys ( %{$h->{$k}} ) ){`
`print '- '.$k2.' : '.$h->{$k}->{$k2}."\n";`
`}`
`$c++;`
`}`
`print "\n Total : ".$c." Accounts.\n\n"`
	Printing the list of accounts.

```
}
__END__
```

[full version]

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(rel2abs);`
`use File::Basename;`
`use Text::CSV;`
`use Encode;`
`use Number::Latin;`
`use List::Util qw(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 = "xxxxxxxxxxx";`
`my $user = "xxxxxxxxxxx";`
`my $pass = "xxxxxxxxxxx";`
`my $dbh = DBI->connect( "dbi:mysql:dbname=$db;host=$host;", $user, $pass ) or die "Connexion impossible à la base de données $db !";`
 	
`my $dbh2 = DBI->connect( "dbi:mysql:dbname=porta-billing;host=xxxxxxxxxxxx;port=xxxxxxxxxxx", "xxxxxxxxxxx", "xxxxxxxxxxx" ) or die "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($i_customer_file);`
 	
`my %follow_mes = get_follow_me_data(%accounts_list);`
 	
`#%follow_mes = get_new_i_account(%follow_mes);`
 	
`# Write to excel file`
`to_excel(%follow_mes) if $write_to_excel;`
 	
`# printing the list`
`print_hash_ref(%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;`
`  my $csv = Text::CSV->new();`
`  my %corr;`
`  my $req = '';`
 	
`  open (CSV, "<", $file) or die $!;`
`  while (<CSV>) {`
`  if ($csv->parse($_)) {`
`  my @columns = $csv->fields();`
`  #print "Searching for i_account : ".$columns[1]."\n";`
`  $req = "SELECT i_account, id FROM Accounts WHERE i_customer =`
	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.

".$columns[1].";";
my $sth = $dbh->prepare($req) or die DBI->errstr();
$sth->execute();
while (my @results = $sth->fetchrow_array()) {
print "Let's insert i_account ".$results[0]." for account
".$results[1]."\n";
$corr{ $results[0] } = $results[1];
}
} else {
my $err = $csv->error_input;
print "Failed to parse line: $err";
}
}
close CSV;
return %corr;
}
	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 %i_acc){
my $req = $req1.$k.$req2;
print $req."\n";
my $sth = $dbh->prepare($req) or die DBI->errstr();
$sth->execute();
while(my $ref = $sth->fetchrow_hashref) {
if ($init_col == 0) {
@cols = keys %$ref;
$init_col++;
}
foreach my $t (sort keys %{$ref}){
$hash{$key}{$t} = $ref->{$t};
}
$key++;
}
}
return %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 keys %hash) {
foreach (@cols){
$hash{$c}{$_} = '' if (!$hash{$c}{$_});
print $_ . "=>" . $hash{$c}{$_} . "\n";
}
print "----------
------\n";
}
}
	Printing the hash for testing.

sub add_field_and_value {
my $a = shift;
my $n = shift;
my $v = shift;
	A simple function to add a value to the hash reference.

`  my %acc = @_;`
`  $acc{$a}{$n} = $v;`
`  if ( !exist_in_array($n,@cols) ){`
`  push (@cols,$n);`
`  } return %acc;`
`}`	

`sub exist_in_array {`
`  my $value = shift;`
`  my @array = @ ;`
`  my $ret = 0;`
`  my %hash;`
`  %hash = map { $_ => 1 } @array;`
`  if ($hash{$value}){`
`  $ret = 1;`
`  }`
`  return $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(time);`
`  my $filename = strftime('%y-%m-%d_%Hh%Mm%Ss',localtime).'_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 "\n\n Starting account excel file creation ... \n\n";`
 	
`  for my $c ( sort keys %hash ) {`
`  $x = 0;`
`  foreach (@cols){`
`  if ( $y == 1 ){ $worksheet->write(0, $x, $_); $columns{$_} = get_column_AA($x+1);}`
`  $temp = decode("utf8", $hash{$c}{$_});`
`  $temp = '' if ( !$temp || !defined($temp) );`
`  $temp = test_mode( $temp, $_ ) if $test_mode;`
`  $worksheet->write($y, $x, $temp);`
`  $x++;`
`  }`
`  $y++;`
`  }`
`  print "\n Please copy to upload configuration file, above [Columns], following lines :\n\n";`
`  foreach (@cols){`
`  print $_."=".$columns{$_}."\n";`
`  }`
`  print "\n\n Created new excel file : ".$dirname."/".$filename."\n\n";`
`}`
	Writing to the Excel file the entire content of the hash reference.

`sub get_column_AA {`
`  my $index = shift;  my $col = int2latin($index);`
`  $col = uc $col;`
`  return $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 %acc ) {`
`  my $req = $req1.$acc{$c}{'AccountID'}.$req2;`
`  print $req."\n";`
`  my $v = $dbh2->selectrow_array($req, undef);`
`  if ($v && $v ne ''){`
`  %acc = add_field_and_value($c,'AccountID',$v,%acc);`
`  } else {`
`  delete $acc{$c};`
`  }`
`  }`
`  return %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`	This function was used

`  my $par1 = shift;`
`  my $par2 = shift;`
`  # if ( $par2 eq 'CustomerName' ){`
`  # $par1 = $par1.'['.`date +%y%m%d%H%M%S`.']';`
`  # }`
`  return $par1;`
`}`
`__END__`
	for generating unique values in order to not have duplicates values in the new billing.
	
	Deactivated.

[full version]

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] and this one [III] 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,`
	This hash makes the link between i_subscription present in the old porta-billing and the new one.

```
    1 => 28,
    2 => 29
 };
```

```
 my $db="porta-billing";
 my $host="master.switzernet.com";
 my $user = "xxxxxxxxxxxx";
 my $pass = "xxxxxxxxxxxx";
 my $dbh = DBI->connect( "dbi:mysql:dbname=$db;host=$host;", $user, $pass )
or die "Connexion impossible à la base de données $db !";

 my $db2="porta-billing";
 my $host2="pbs1.switzernet.com";
 my $user2 = "xxxxxxxxxxxx";
 my $pass2 = "xxxxxxxxxxxx";
 my $dbh2 = DBI->connect( "dbi:mysql:dbname=$db2;host=$host2;", $user2,
$pass2 ) or die "Connexion impossible à la base de données $db2 !";
```

Mysql connection to the two old and new databases.

```
open (LOGFILE, '>>log.txt');
my $date = `date +'%Y-%m-%d %T'`;
chomp($date);
print LOGFILE " -- LOG -- " . $date . " -- LOG -- \n";
binmode(STDOUT, ':utf8');
my $proxy host = 'https://slave.switzernet.com'; # 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",
);
binmode(STDOUT, ':utf8');
my $proxy host = 'https://slave.switzernet.com'; # 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 {
  my ($soap, $res) = @_;
  die "SOAP Fault: $!, " . (ref $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 "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 $subscriptions = get subscriptions();
print list($subscriptions);
insert update subscr($subscriptions);
```

Main routine which get the subscription and insert them in the new billing.

```
$dbh->disconnect();
$dbh2->disconnect();
```

Disconnection of the MySQL databases.

close (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 $k (keys(%$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 $req."\n\n" if ($debug);
$temp = $sth2->fetchall_hashref('i_customer_subscription');
foreach my $tk (keys %$temp){
foreach my $tk2 (keys (%{$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 $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;
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 "ACT DATE ".$sub->{activation_date};
 	
my $UpdateCustomerSubscriptionRequest = {
i_customer => $sub->{i_customer},
i_customer_subscription => $sub->{i_customer_subscription},
subscription_info => $CustomerSubscriptionInfo
};
 	
print " UPDATE \n\n" if ($debug);
 	
my $AddUpdateCustomerSubscriptionResponse = $customer_service->update_subscription($header,$UpdateCustomerSubscriptionRequest)->result;
my $i_customer_subscription = $AddUpdateCustomerSubscriptionResponse->{i_customer_subscription};
 	
print "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;
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
	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

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 $req."\n\n" if ($debug);
my $sth = $dbh->prepare($req);
$sth->execute();
while ($h = $sth->fetchrow hashref) {
foreach $k (keys(%$h)) {
$h->{$k} = '' if(!$h->{$k});
print $k." : ".$h->{$k}." ? ".$cust->{$k}."\n\n" if ($debug);
if ($k eq 'i_customer_subscription'){
$ics = $h->{$k};
} else {
$diff = 1;
print "DIFF\n";
}
}
$found = 1;
if ($h->{is finished} eq 'Y'){
$diff = 0;
}
}
if ($found && !$diff){
$c = 0;
} elsif ($found && $diff){
$c = 1;
} else {
$c = 2;
}
return ($todo->[$c], $ics);
}
	to do for each one (insert, update, ignore).

sub insert_subscription_new_pb {
my $sub = shift;
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 $kcsi (keys %$CustomerSubscriptionInfo){
print $kcsi . " => " . $CustomerSubscriptionInfo->{$kcsi} . "\n" ;
}
 	
print " INSERT \n\n" if ($debug);
 	
my $AddUpdateCustomerSubscriptionResponse = $customer_service->add_subscription($header,$AddCustomerSubscriptionRequest)->result;
$i_customer_subscription = $AddUpdateCustomerSubscriptionResponse->{i_customer_subscription};
 	
print "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;
foreach my $k (keys(%$subs)) {
	A simple function to define the procedure

`if ( $subs->{$k}->{todo} eq 'insert' ) {`
`  insert subscription new pb($subs->{$k});`
`} elsif ( $subs->{$k}->{todo} eq 'update' ){`
`  update subscription new pb($subs->{$k});`
`} else {`
`  print LOGFILE " Ignoring subscription " . $subs->{$k}->{i_subscription} . " OF CUSTOMER " . $subs->{$k}->{i_customer}."\n";`
`  print " INGNORING SUBSCRIPTION " if ($debug);`
`}`
`}`
`}`
	to follow depending of the 'todo' field we have added before.

`sub print_list {`
`  my $h = shift;`
`  my $c = 0;`
`  foreach my $k ( keys( %$h ) ) {`
`  print "\n> KEY ".$k.":\n";`
`  foreach my $k2 ( keys ( %{$h->{$k}} ) ){`
`  print '- '.$k2.' : '.$h->{$k}->{$k2}."\n";`
`  }`
`  $c++;`
`  }`
`  print "\n Total : ".$c." registers.\n\n"`
`}`
	Printing all data for debug.

`__END__`	The end.

[full version]

How-to use

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

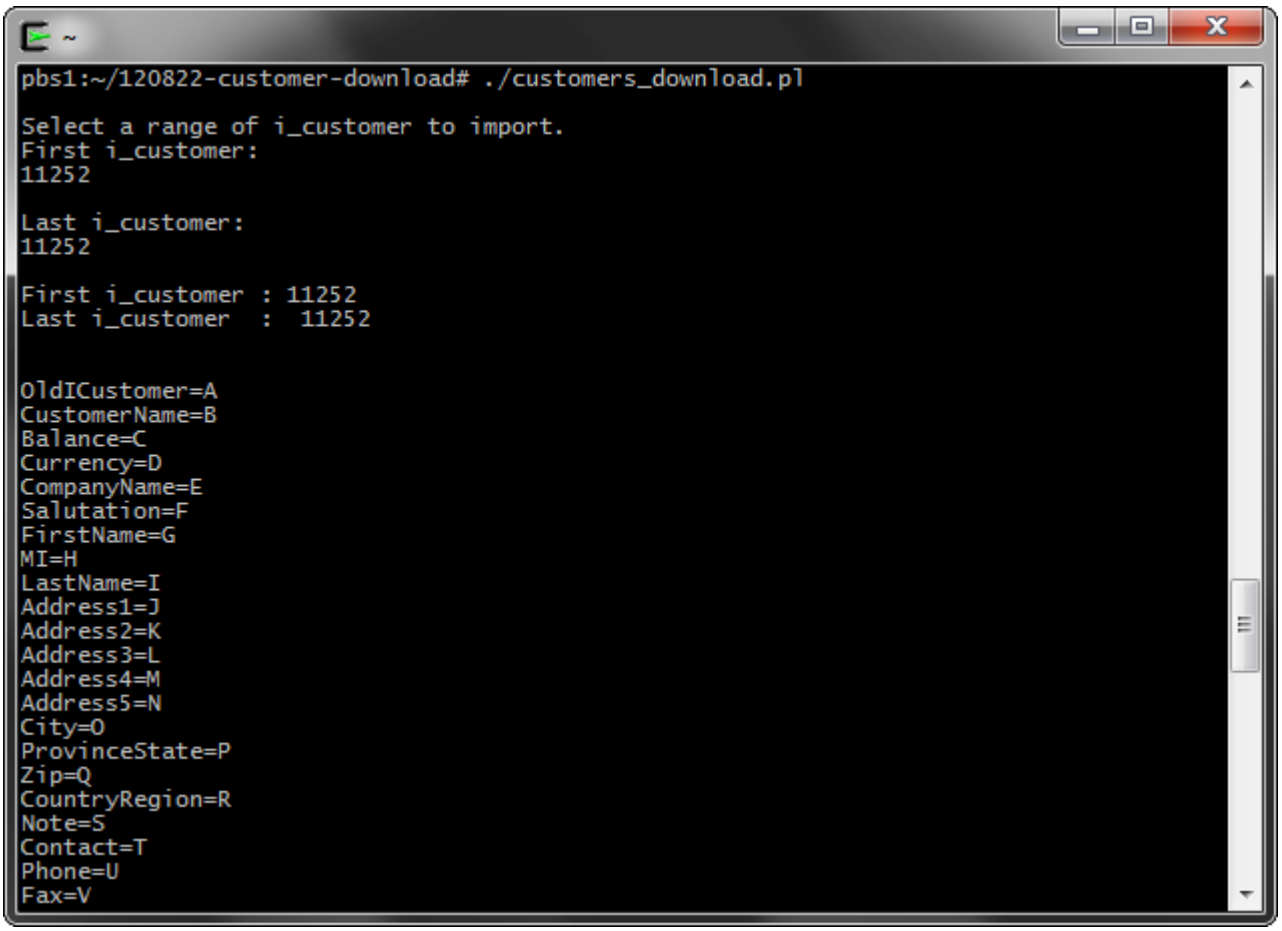
In this how-to, we will only take a sample account. The procedure is the same for large number of accounts.

Import customers

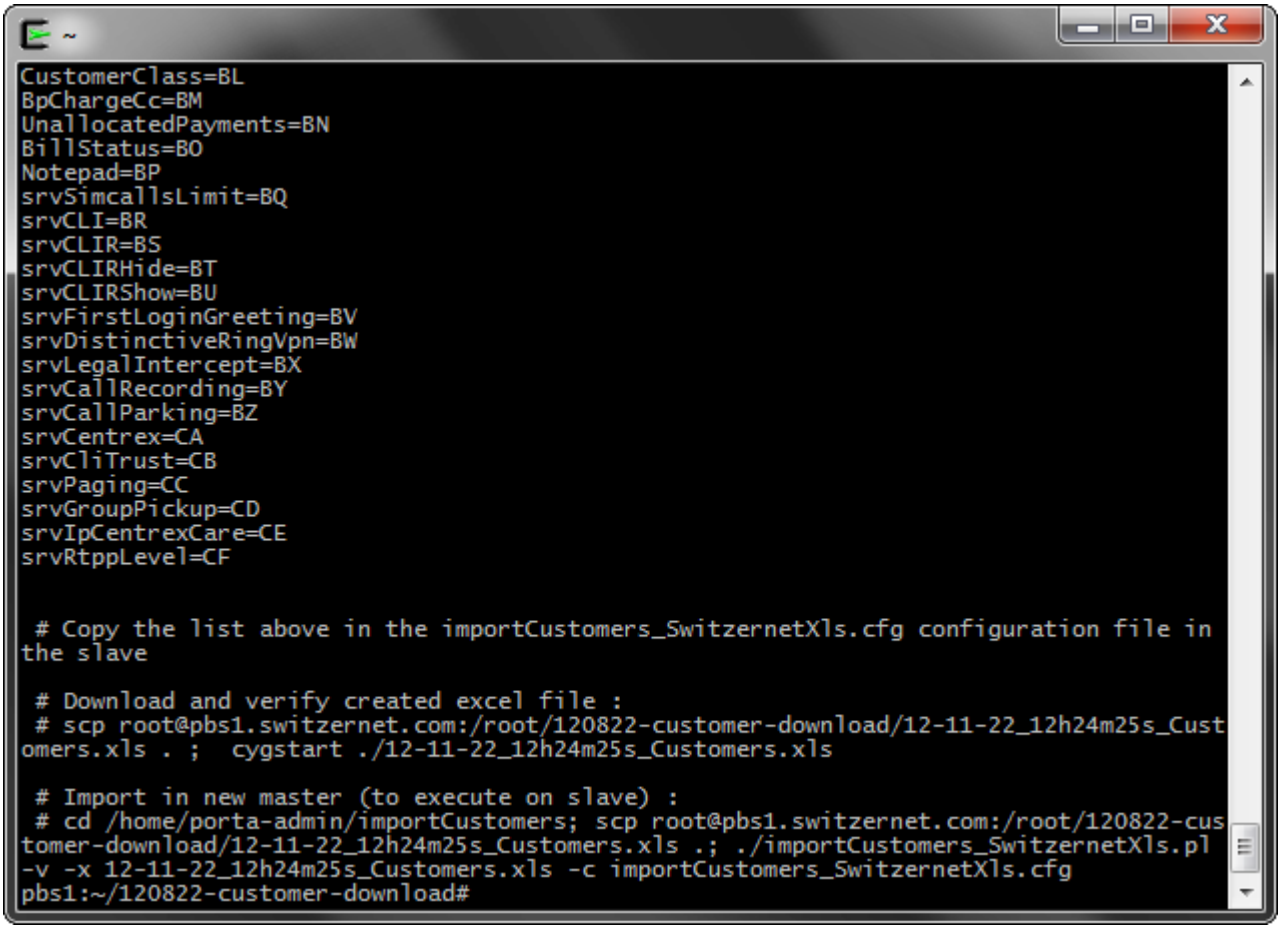
On pbs1 go to /root/120822-customer-download/ and execute the following commands:

```
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:

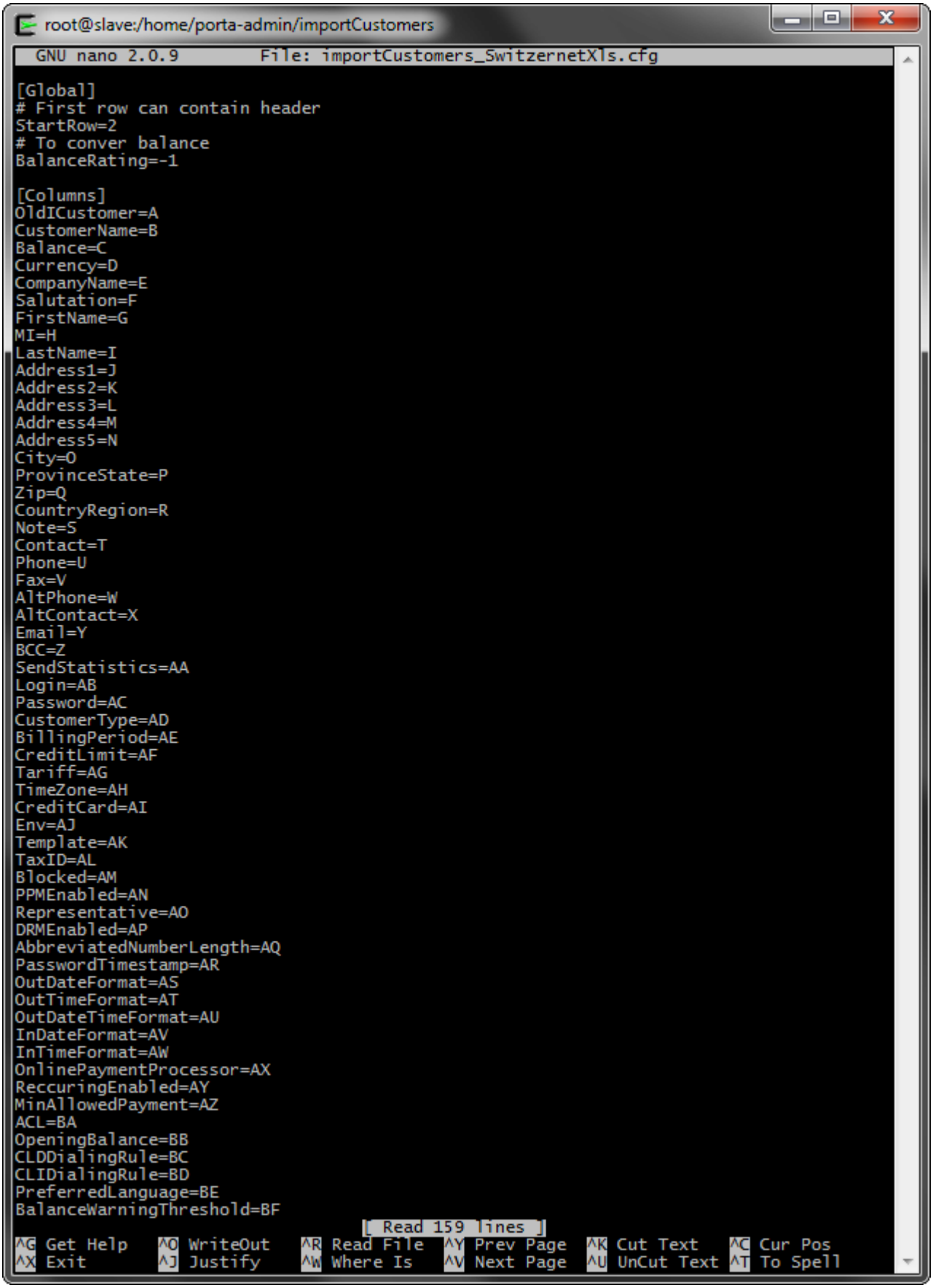


And next steps to execute:



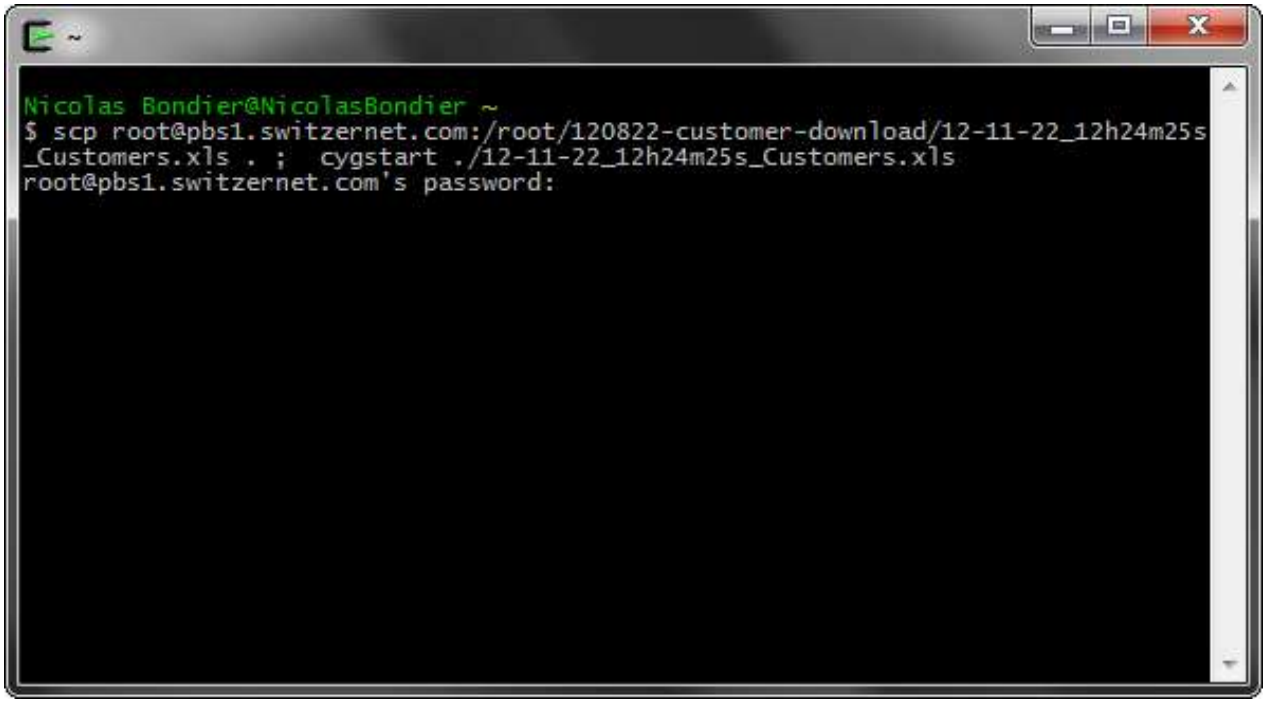
Connect through SSH to the new slave server and go to */home/porta-admin/importCustomers*. Open *importCustomers_SwitzernetXls.cfg* and past the list of correspondences under *[Columns]*.

```
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
```

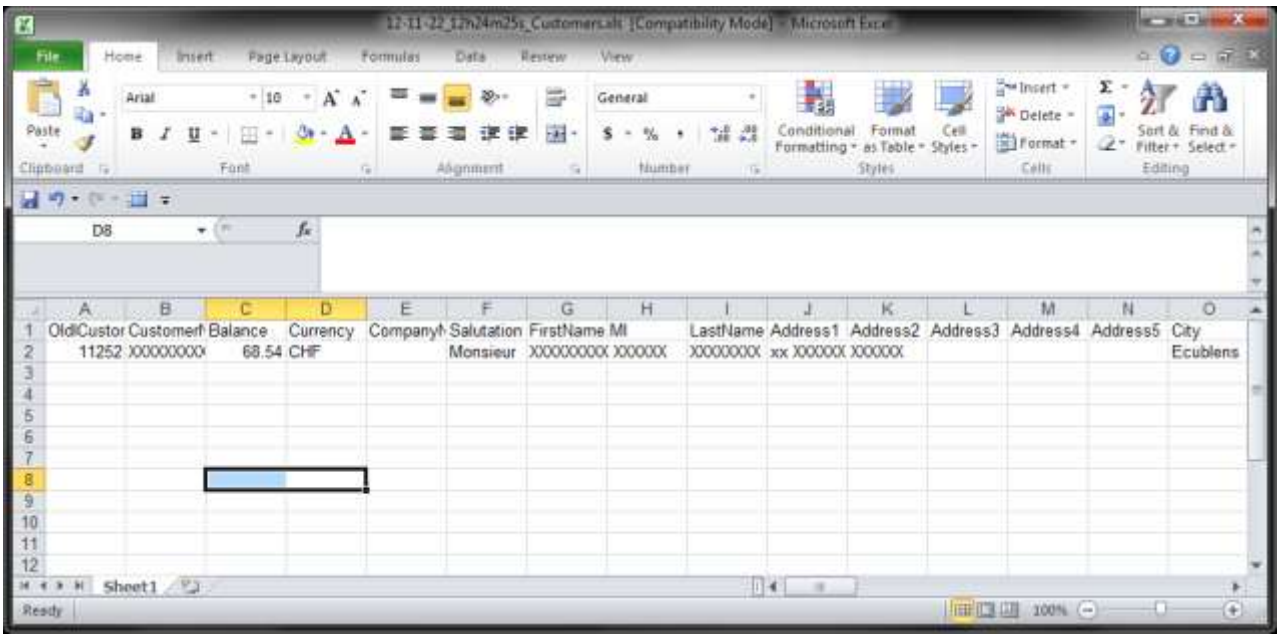


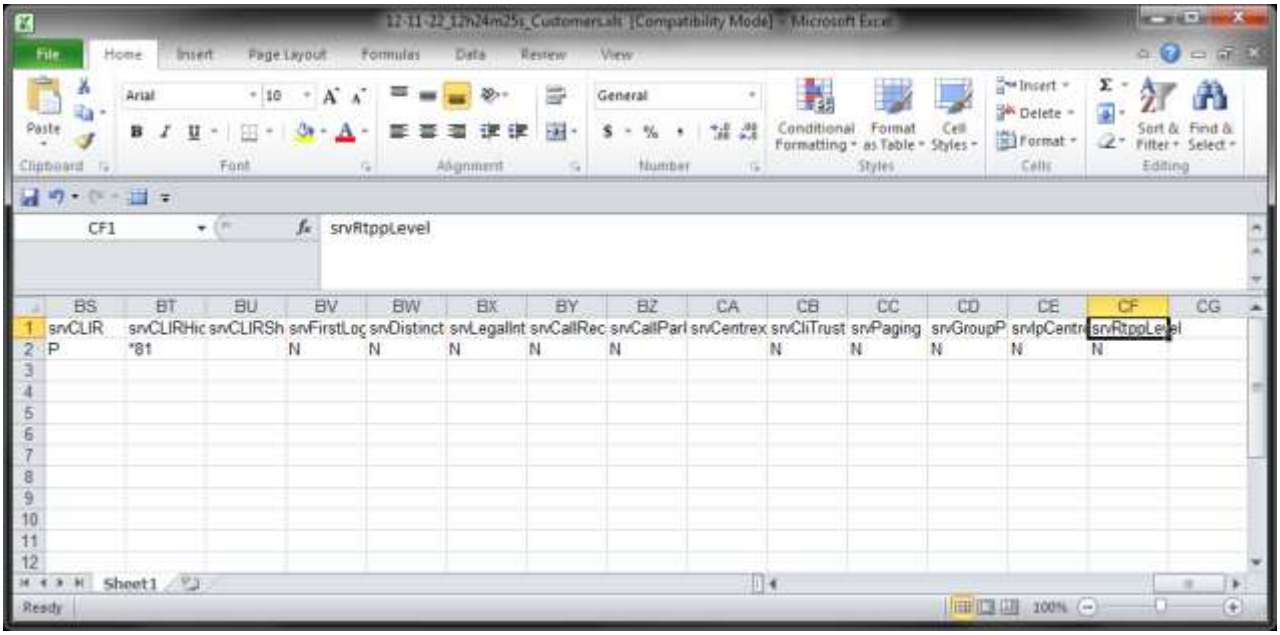
This is the only changes you have to make to *importCustomers_SwitzernetXls.cfg*.

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.

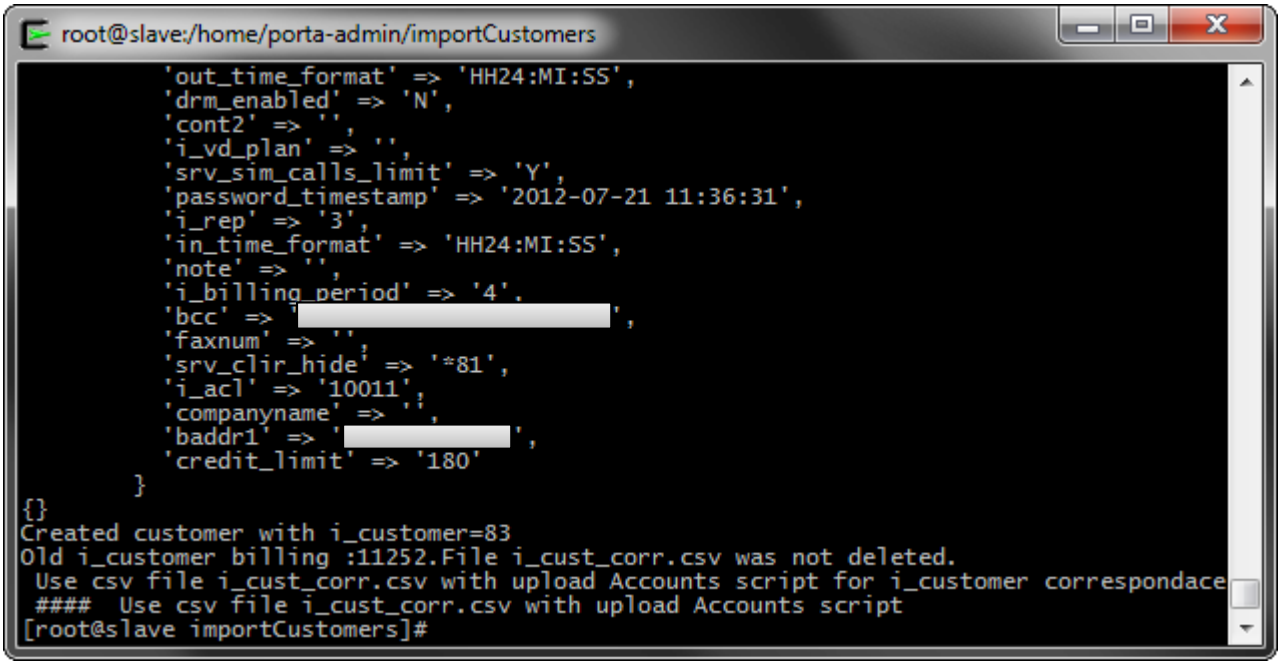




Once verified, go back to the slave in your working folder and execute the second command given as output of the downloading script in order to get the new excel file and uploading it.

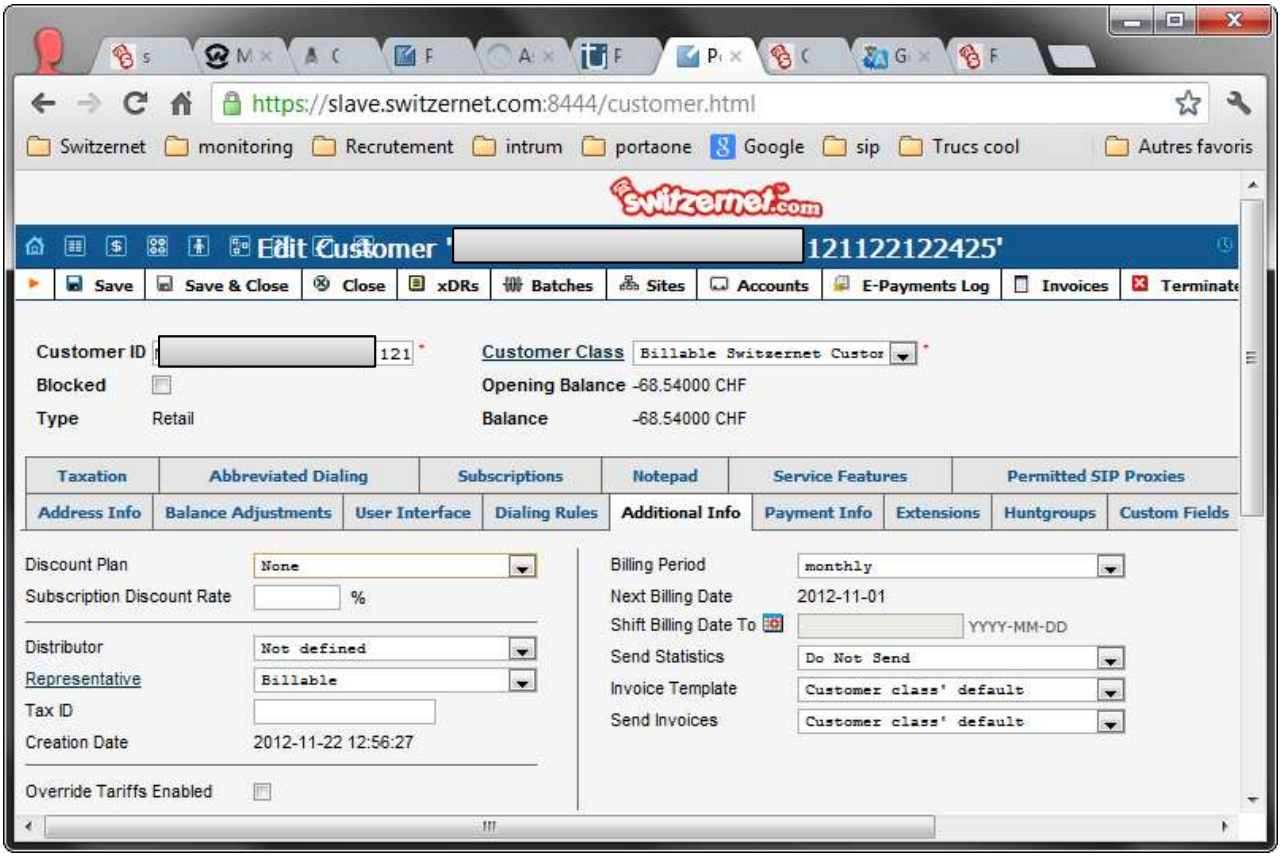
```
[root@slave importCustomers]# cd /home/porta-admin/importCustomers;
scp root@pbs1.switzernet.com:/root/120822-customer-download/12-11-
22_12h24m25s_Customers.xls .;
./importCustomers_SwitzernetXls.pl -v -x 12-11-22_12h24m25s_Customers.xls -c
importCustomers_SwitzernetXls.cfg
```

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.



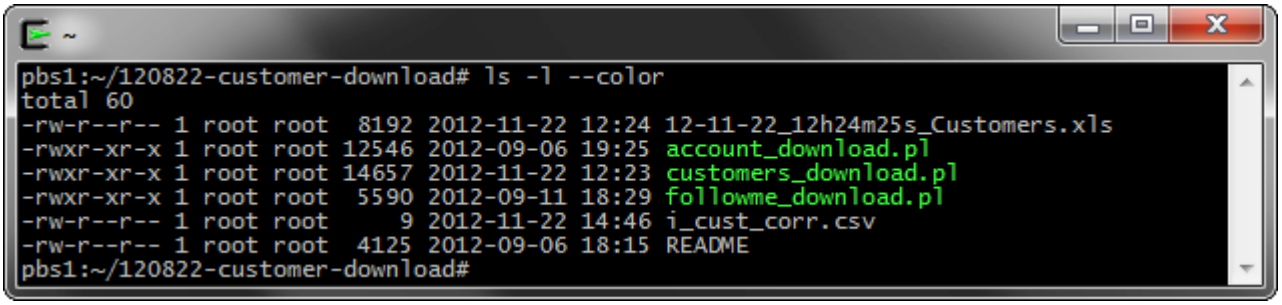
Verify the data from the web interface is the same as the old billing (with some exceptions in Service Features). If everything seems good, let's upload the accounts for these customers.

Import accounts

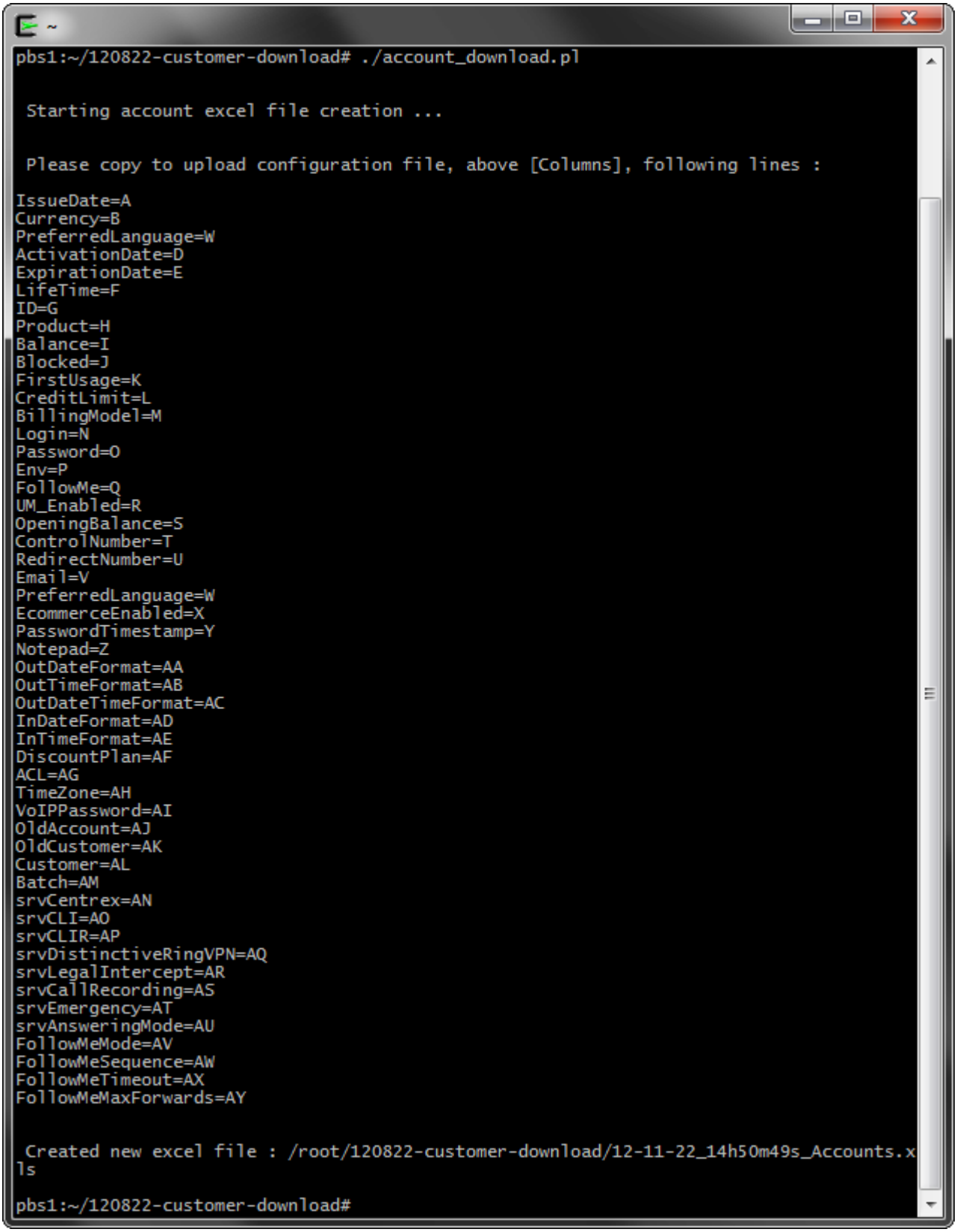
Go to your working folder on pbs1 and download the new created '*i_cust_corr.csv*' at this place.

```
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#
```

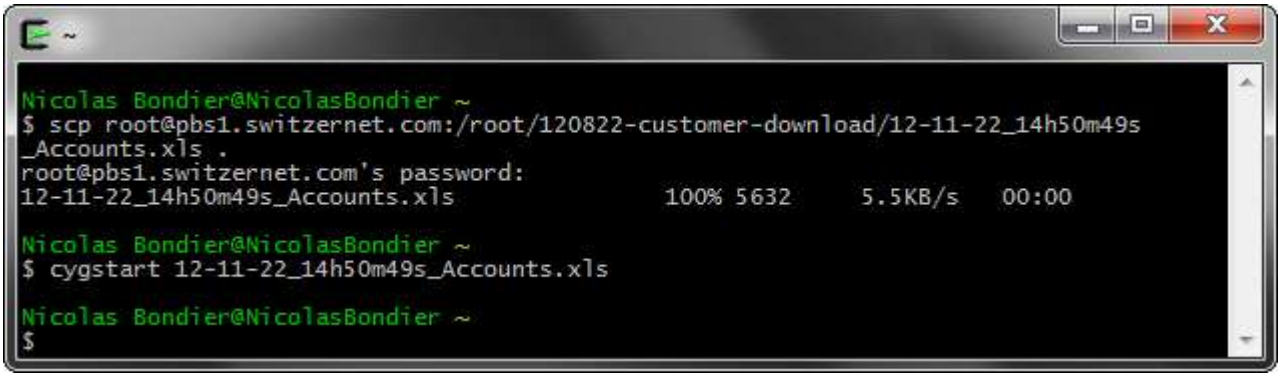
This is the list of files you should have in you working folder.

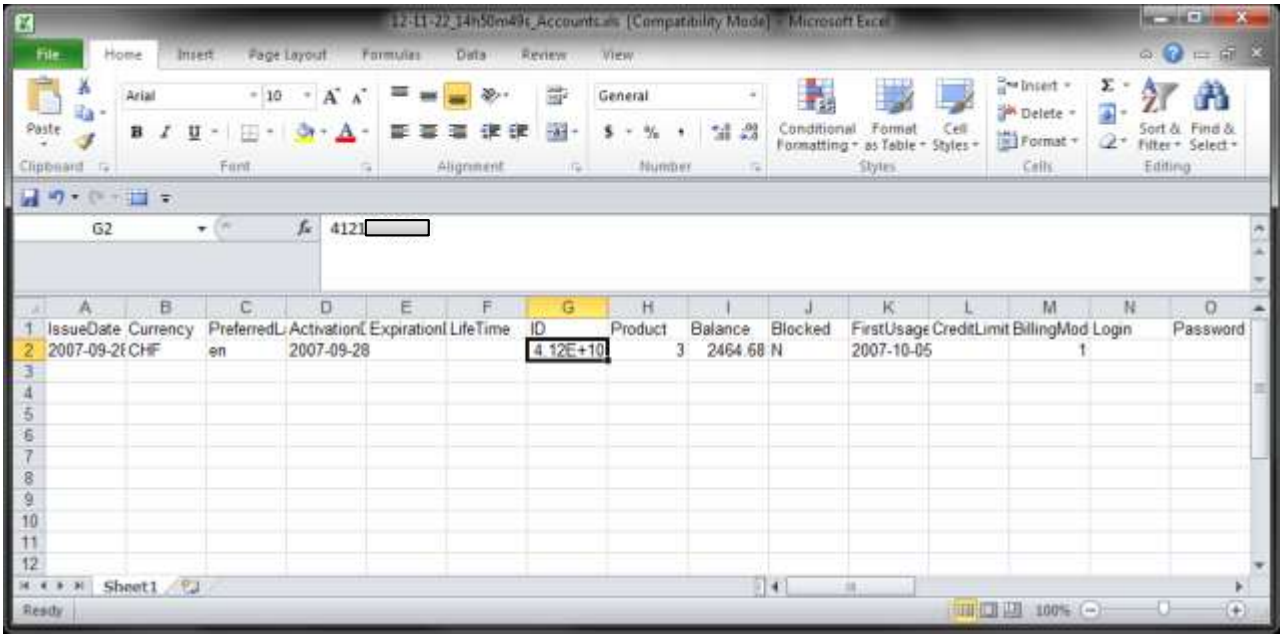


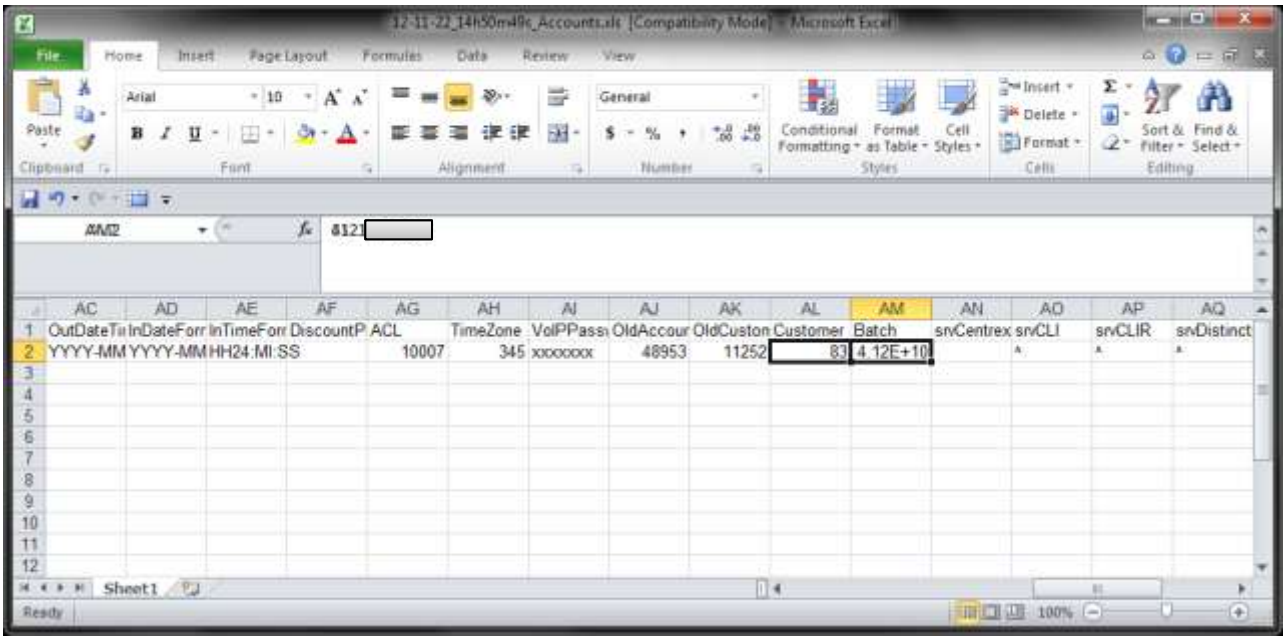
Once '*i_cust_corr.csv*' copied, simply run the '*./account_download.pl*'. It will read our new file, get all accounts data for the customers of '*i_cust_corr.csv*' and create the new excel.

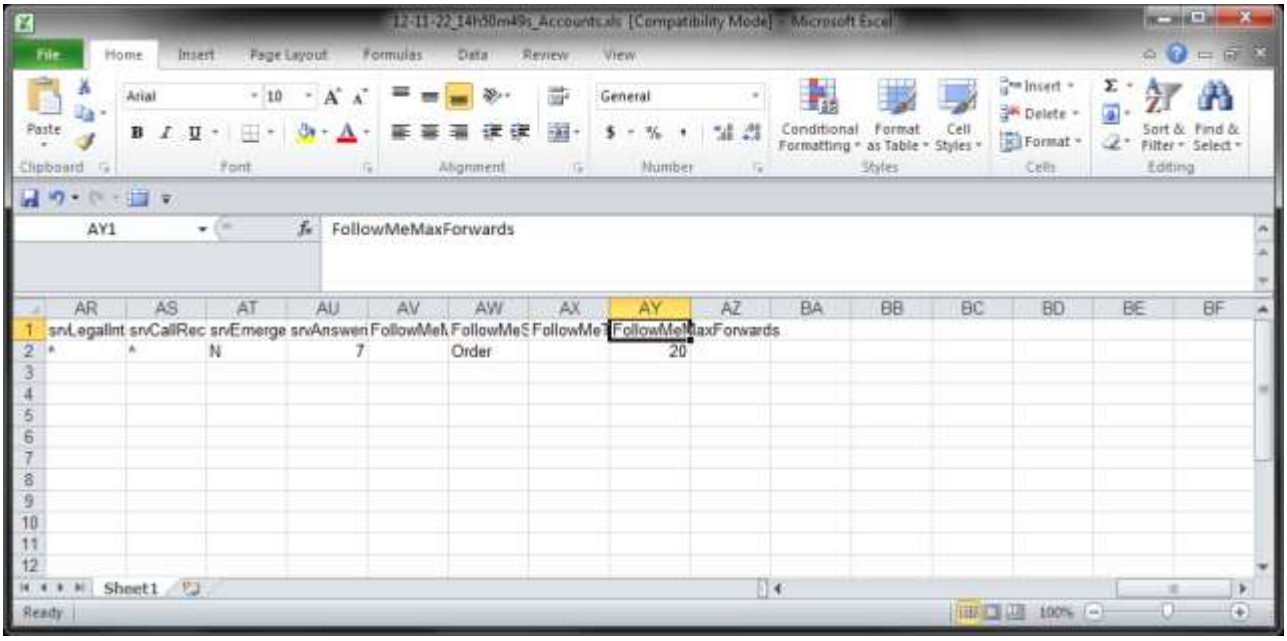


The file path is provided. Simply download to your desktop, open and verify if the excel is correct.


Nicolas Bondier@NicolasBondier ~
$ scp root@pbs1.switzernet.com:/root/120822-customer-download/12-11-22_14h50m49s
_Accounts.xls .
root@pbs1.switzernet.com's password:
12-11-22_14h50m49s_Accounts.xls 100% 5632 5.5KB/s 00:00
Nicolas Bondier@NicolasBondier ~
$ cygstart 12-11-22_14h50m49s_Accounts.xls
Nicolas Bondier@NicolasBondier ~
$

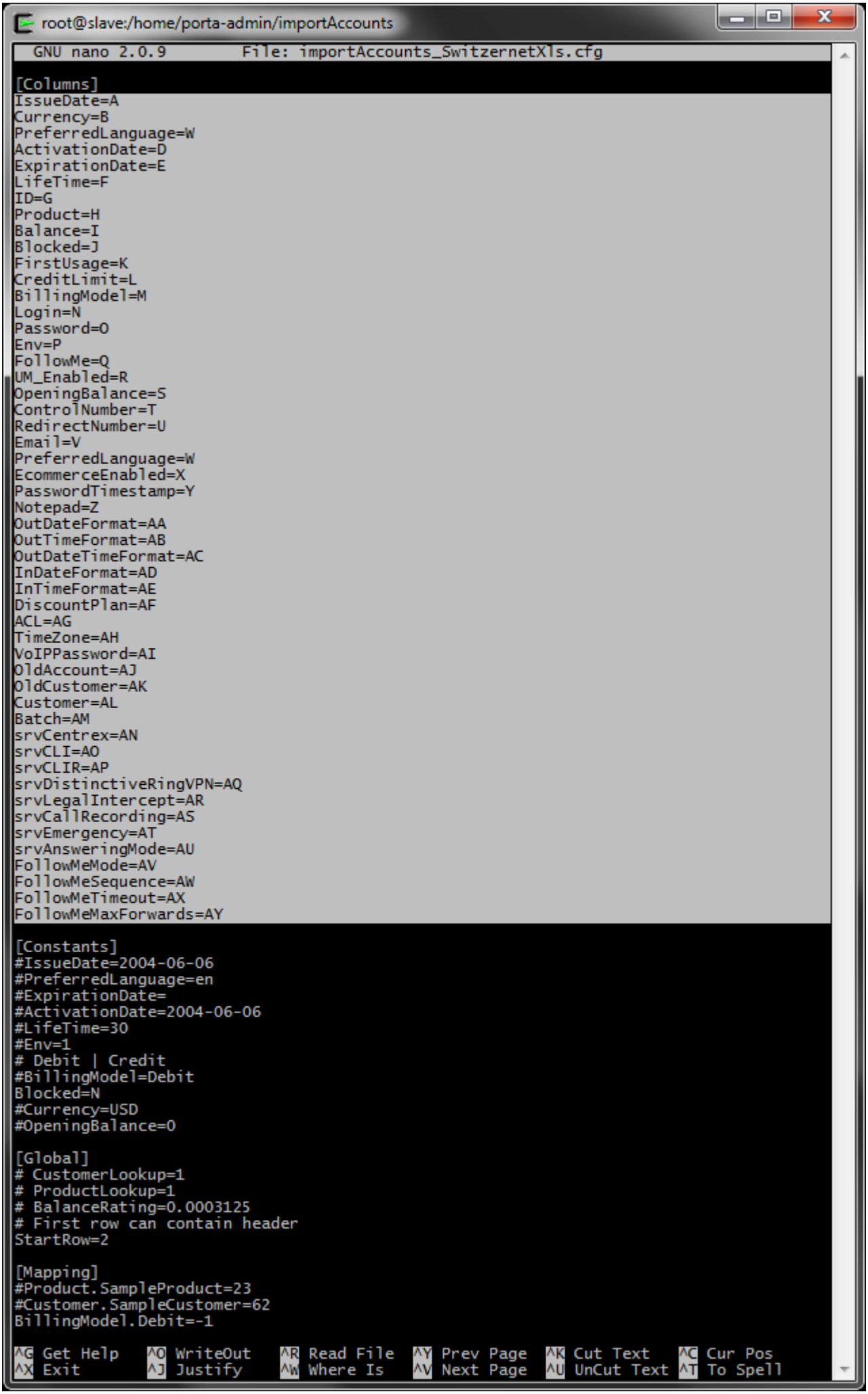

IssueDate Currency PreferredL ActivationL ExpirationL LifeTime ID Product Balance Blocked FirstUsage CreditLimit BillingMod Login Password
2007-09-2 CHF en 2007-09-28 4.12E+10 3 2464.68 N 2007-10-05 1


OutDateTi InDateForr InTimeFor DiscountP ACL TimeZone VoIPPass OldAccour OldCustor Customer Batch srvCentrex srvCLI srvCLIR srvDistinct
YYYY-MM YYYY-MM HH24:MI:SS 10007 345 xxxxxxx 48953 11252 83 4.12E+10



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
```


root@slave:/home/porta-admin/importAccounts
GNU nano 2.0.9 File: importAccounts_SwitzernetXls.cfg
[Columns]
IssueDate=A
Currency=B
PreferredLanguage=W
ActivationDate=D
ExpirationDate=E
LifeTime=F
ID=G
Product=H
Balance=I
Blocked=J
FirstUsage=K
CreditLimit=L
BillingModel=M
Login=N
Password=O
Env=P
FollowMe=Q
UM_Enabled=R
OpeningBalance=S
ControlNumber=T
RedirectNumber=U
Email=V
PreferredLanguage=W
EcommerceEnabled=X
PasswordTimestamp=Y
Notepad=Z
OutDateFormat=AA
OutTimeFormat=AB
OutDateTimeFormat=AC
InDateFormat=AD
InTimeFormat=AE
DiscountPlan=AF
ACL=AG
TimeZone=AH
VoIPPassword=AI
OldAccount=AJ
OldCustomer=AK
Customer=AL
Batch=AM
srvCentrex=AN
srvCLI=AO
srvCLIR=AP
srvDistinctiveRingVPN=AQ
srvLegalIntercept=AR
srvCallRecording=AS
srvEmergency=AT
srvAnsweringMode=AU
FollowMeMode=AV
FollowMeSequence=AW
FollowMeTimeout=AX
FollowMeMaxForwards=AY
[Constants]
#IssueDate=2004-06-06
#PreferredLanguage=en
#ExpirationDate=
#ActivationDate=2004-06-06
#LifeTime=30
#Env=1
Debit | Credit
#BillingModel=Debit
Blocked=N
#Currency=USD
#OpeningBalance=0
[Global]
# CustomerLookup=1
# ProductLookup=1
# BalanceRating=0.0003125
# First row can contain header
StartRow=2
[Mapping]
#Product.SampleProduct=23
#Customer.SampleCustomer=62
BillingModel.Debit=-1
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Download now the accounts excel file in our working directory.

```
root@slave:/home/porta-admin/importAccounts
[root@slave importAccounts]# scp root@pbs1.switzernet.com:/root/120822-customer-downlo
ad/12-11-22_14h50m49s_Accounts.xls .
root@pbs1.switzernet.com's password:
12-11-22_14h50m49s_Accounts.xls                    100% 5632     5.5KB/s   00:00
[root@slave importAccounts]#
```

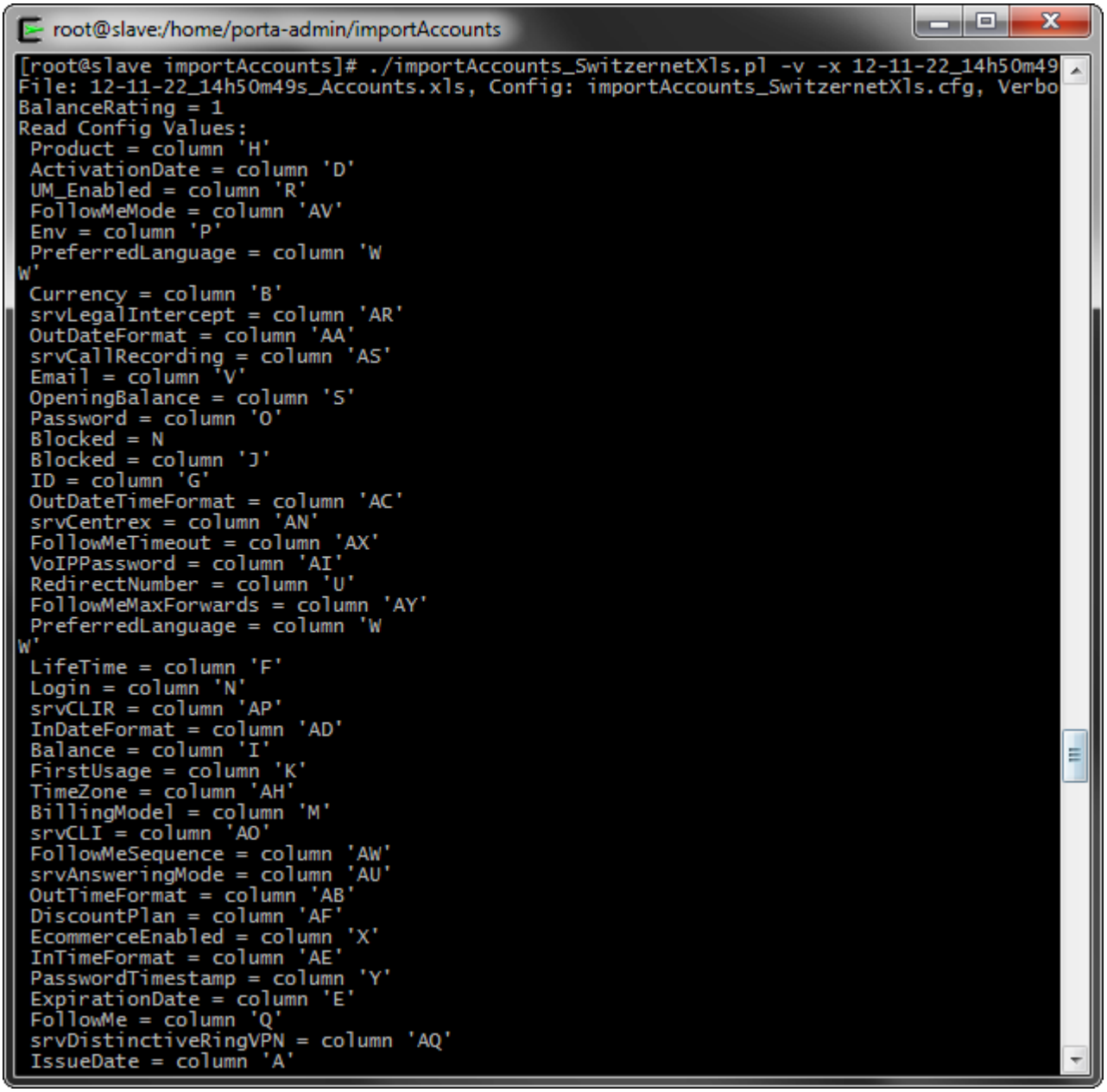
Here is a list of the files in the folder. We only use '*importAccounts_SwitzernetXls.cfg*', '*importAccounts_SwitzernetXls.pl*' and '*12-11-22_14h50m49s_Accounts.xls*'.

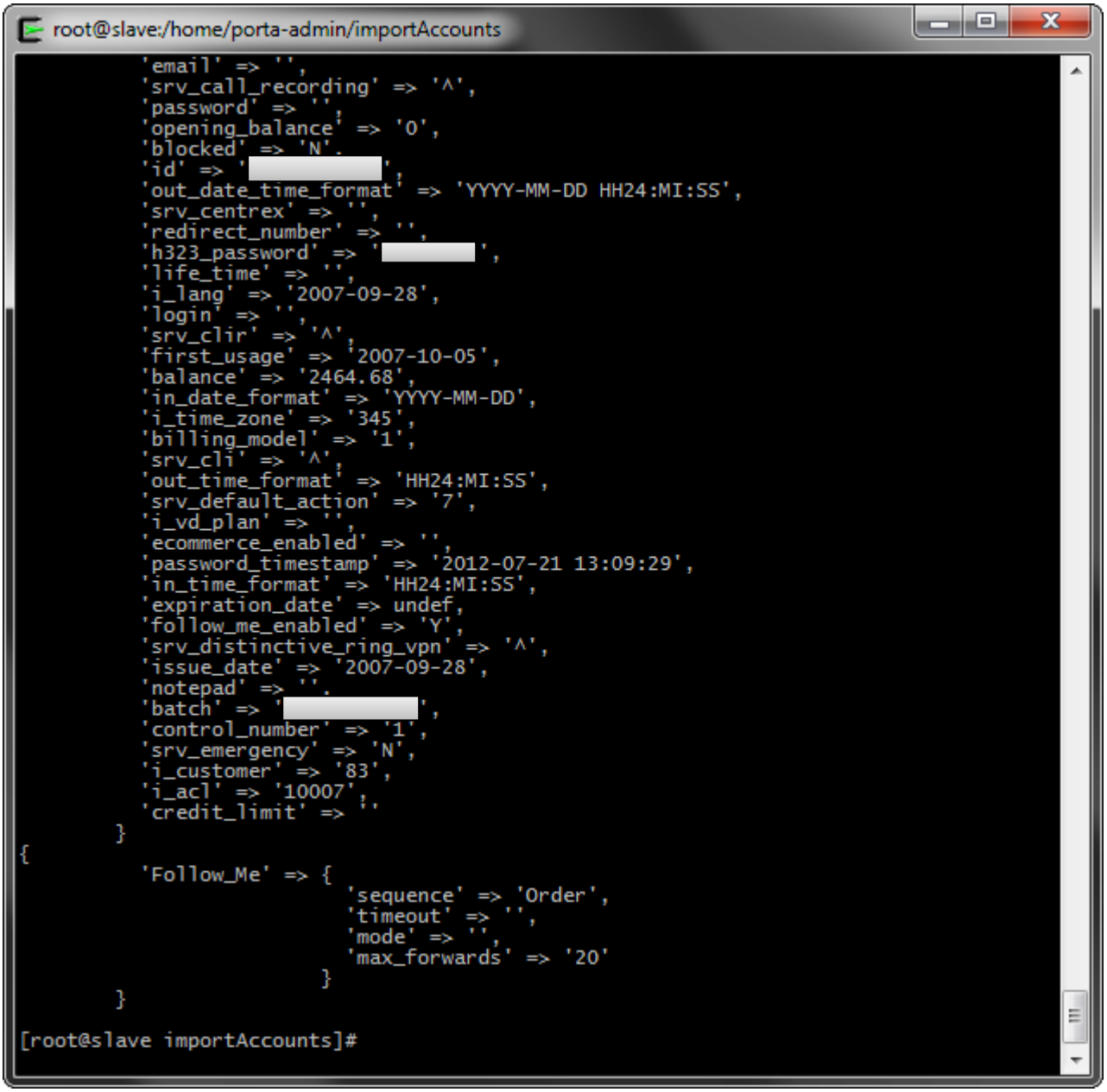
```
root@slave:/home/porta-admin/importAccounts
[root@slave importAccounts]# ls -l --color
total 84
-rw-r--r-- 1 root        root   5632 Nov 22 14:21 12-11-22_14h50m49s_Accounts.xls
drwxr-xr-x 2 porta-admin staff  4096 Feb  7  2012 CVS
-rw-r--r-- 1 root        root   1201 Sep  6 17:25 importAccounts_SwitzernetXls.cfg
-rwxr-xr-x 1 root        root  19269 Sep  6 16:46 importAccounts_SwitzernetXls.pl
-rw-r--r-- 1 porta-admin staff   656 Jan 14  2009 importcsv.cfg
-rw-r--r-- 1 porta-admin staff 14848 Nov 26  2006 importExample.xls
-rw-r--r-- 1 root        root     55 Sep  6 17:24 import.log
-rwxr-xr-x 1 porta-admin staff 19244 Feb 16  2011 import.pl
-rw-r--r-- 1 porta-admin staff   545 Jan 14  2009 importxls.cfg
[root@slave importAccounts]#
```

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.

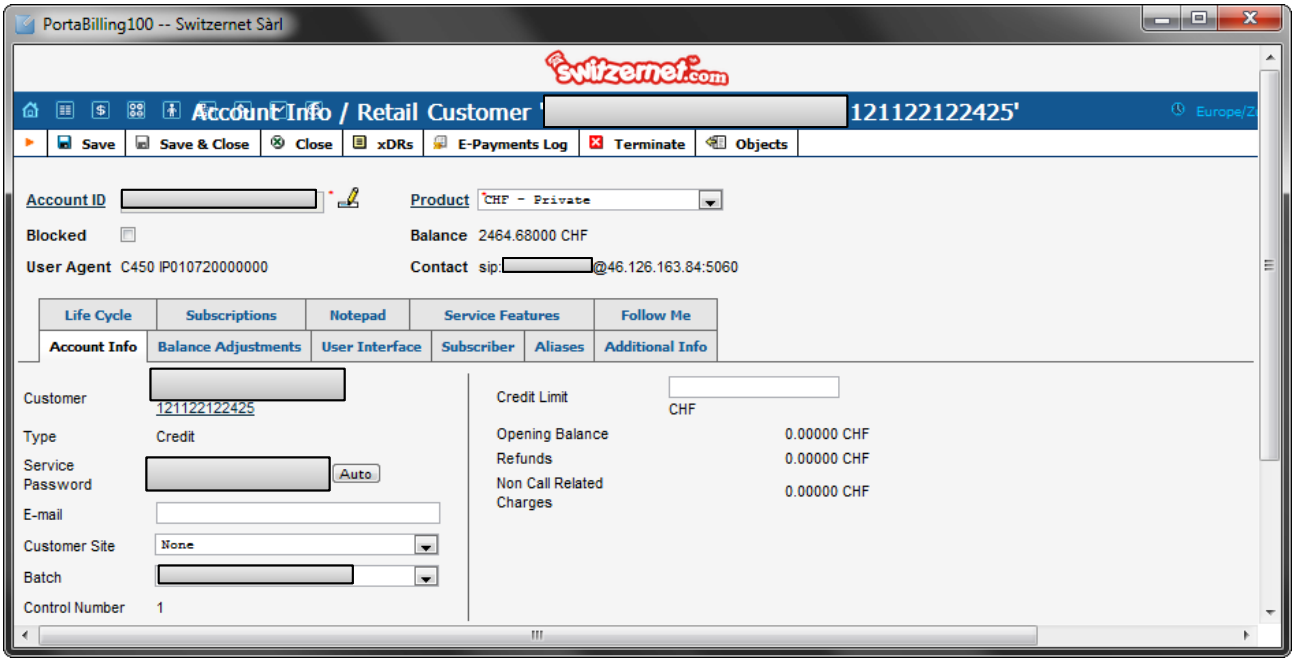
We only need to upload the new accounts. Run the following command with your new excel file :

```
./importAccounts_SwitzernetXls.pl -v -x 12-11-22_14h50m49s_Accounts.xls -c
importAccounts_SwitzernetXls.cfg
```


root@slave:/home/porta-admin/importAccounts
[root@slave importAccounts]# ./importAccounts_SwitzernetXls.pl -v -x 12-11-22_14h50m49
File: 12-11-22_14h50m49s_Accounts.xls, Config: importAccounts_SwitzernetXls.cfg, Verbo
BalanceRating = 1
Read Config Values:
Product = column 'H'
ActivationDate = column 'D'
UM_Enabled = column 'R'
FollowMeMode = column 'AV'
Env = column 'P'
PreferredLanguage = column 'W
W'
Currency = column 'B'
srvLegalIntercept = column 'AR'
OutDateFormat = column 'AA'
srvCallRecording = column 'AS'
Email = column 'V'
OpeningBalance = column 'S'
Password = column 'O'
Blocked = N
Blocked = column 'J'
ID = column 'G'
OutDateTimeFormat = column 'AC'
srvCentrex = column 'AN'
FollowMeTimeout = column 'AX'
VoIPPassword = column 'AI'
RedirectNumber = column 'U'
FollowMeMaxForwards = column 'AY'
PreferredLanguage = column 'W
W'
LifeTime = column 'F'
Login = column 'N'
srvCLIR = column 'AP'
InDateFormat = column 'AD'
Balance = column 'I'
FirstUsage = column 'K'
TimeZone = column 'AH'
BillingModel = column 'M'
srvCLI = column 'AO'
FollowMeSequence = column 'AW'
srvAnsweringMode = column 'AU'
OutTimeFormat = column 'AB'
DiscountPlan = column 'AF'
EcommerceEnabled = column 'X'
InTimeFormat = column 'AE'
PasswordTimestamp = column 'Y'
ExpirationDate = column 'E'
FollowMe = column 'Q'
srvDistinctiveRingVPN = column 'AQ'
IssueDate = column 'A'



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.

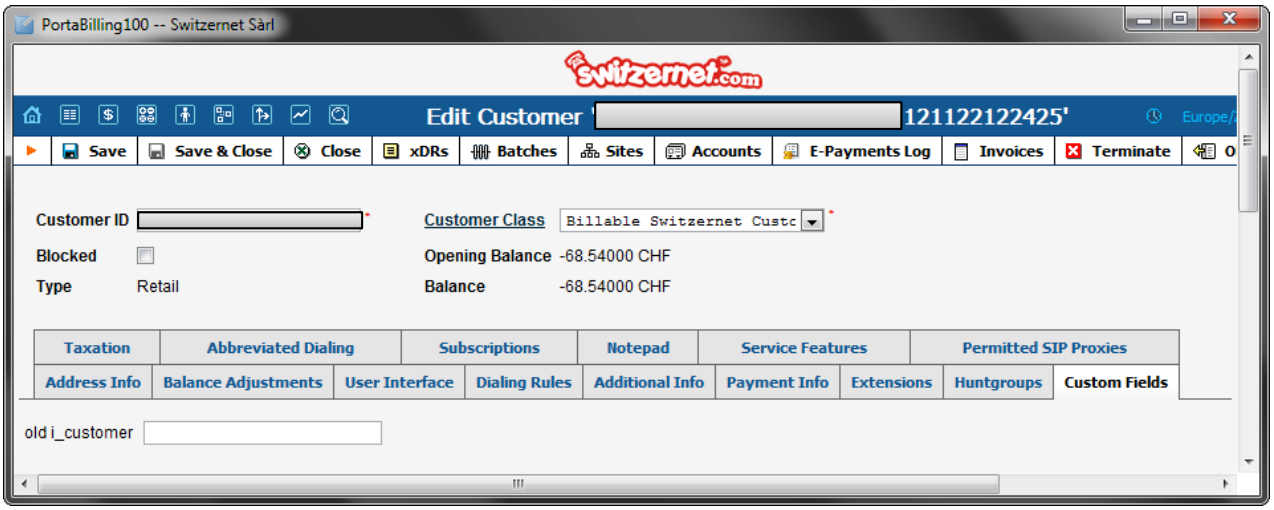
At this point the account is operational. We must continue the importation to provide other features like follow me, limit the maximum number of calls by account and add the subscriptions.

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

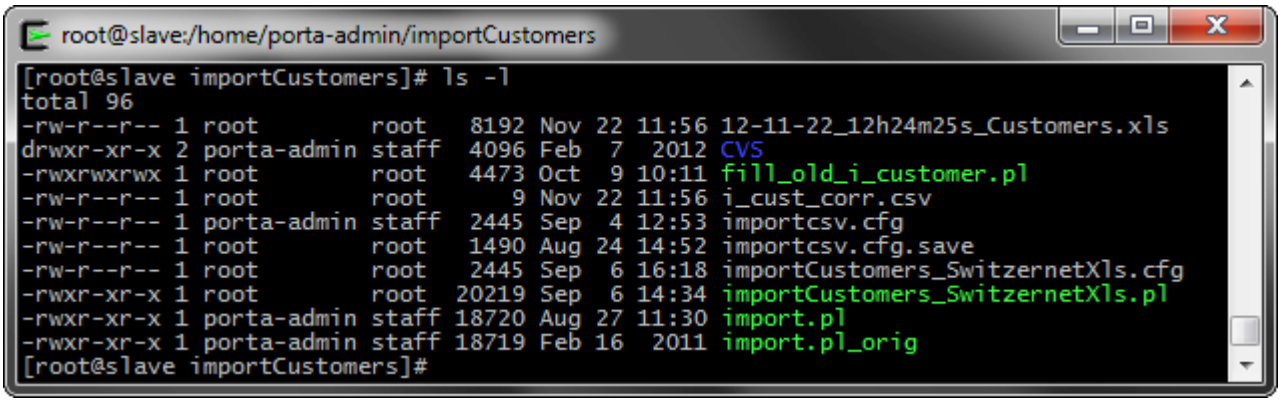
Note also that accounting is send to the 2 billing masters if the account is on both. This way, no data is lost.

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.

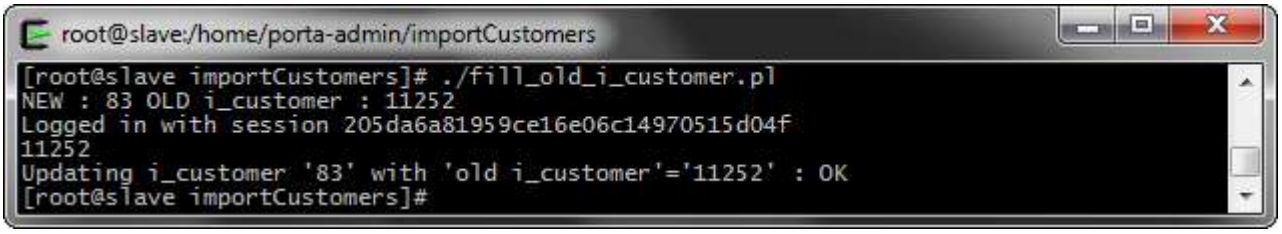


Go back to the '/home/porta-admin/importCustomers' on the new slave. In this folder a script called 'fill_old_i_customer.pl' is present.

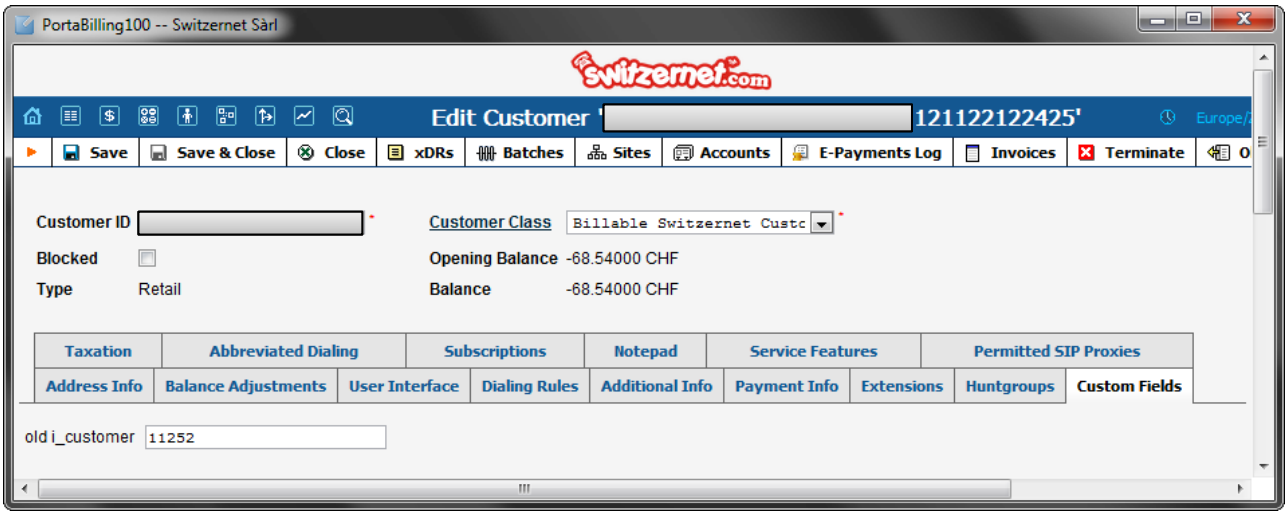


The script will use the '*i_cust_corr.csv*' file and will upload the old *i_customer* value to the custom field we have created. This is done through SOAP connection.

Just run the script:



The old i_customer is now filled in the web interface:



Import customer sites

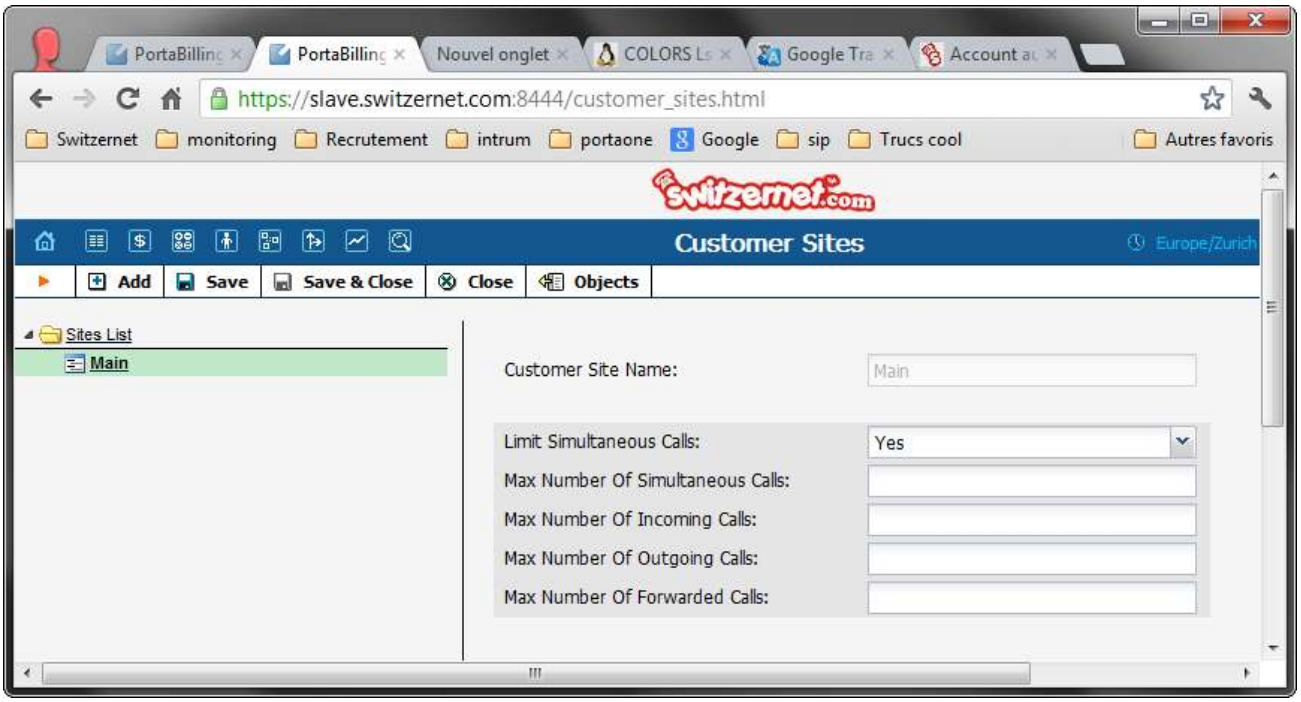
For limiting maximum simultaneous calls for each account, we decided to use the new Customers Sites feature as the limit for each customer as disappeared. This point is essential for fraud prevention and for meeting our product restrictions.

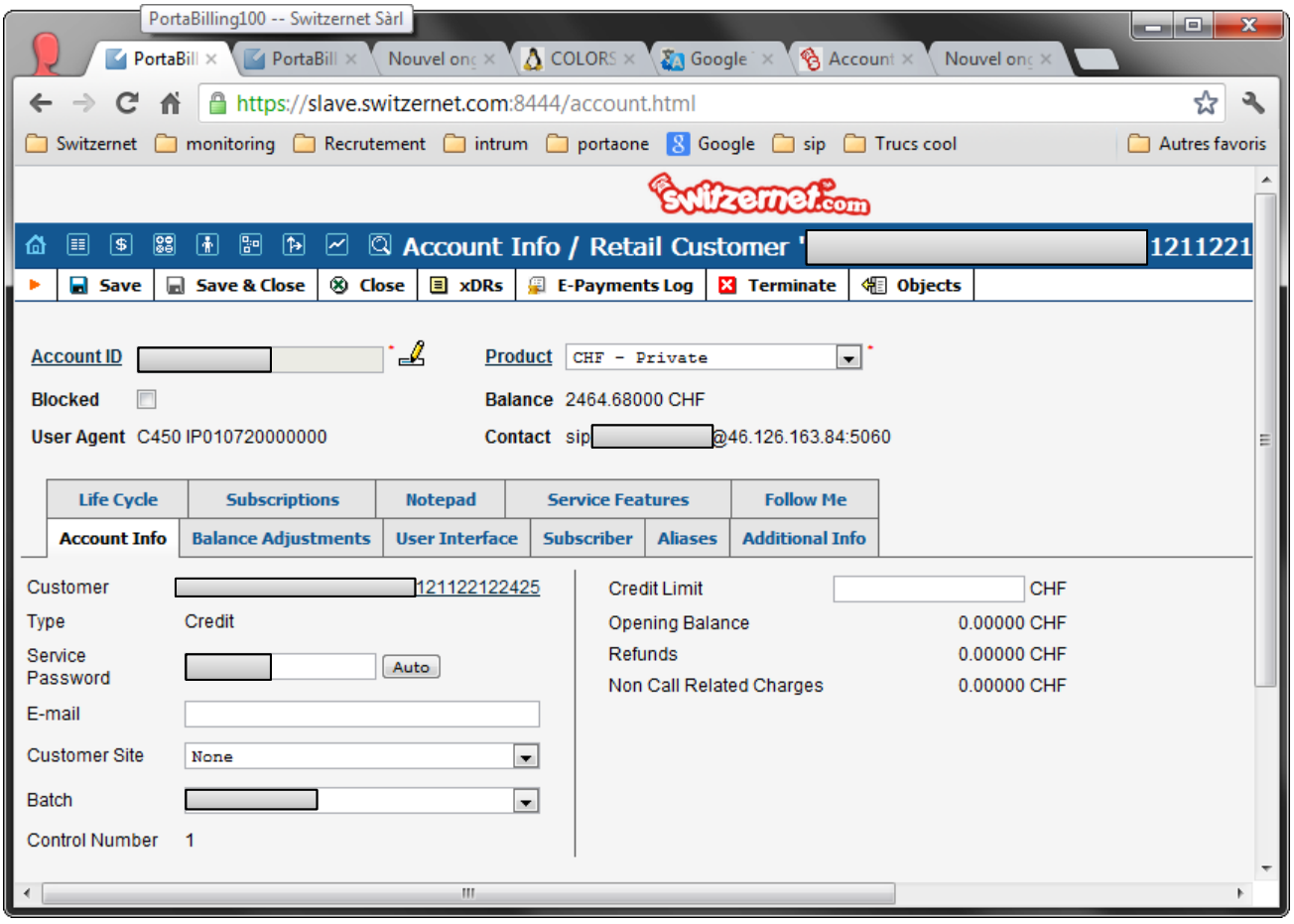
Change directory on the slave to '*/home/porta-admin/importCustomerSites*'. Only one script is present: '*import.pl*'.

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.

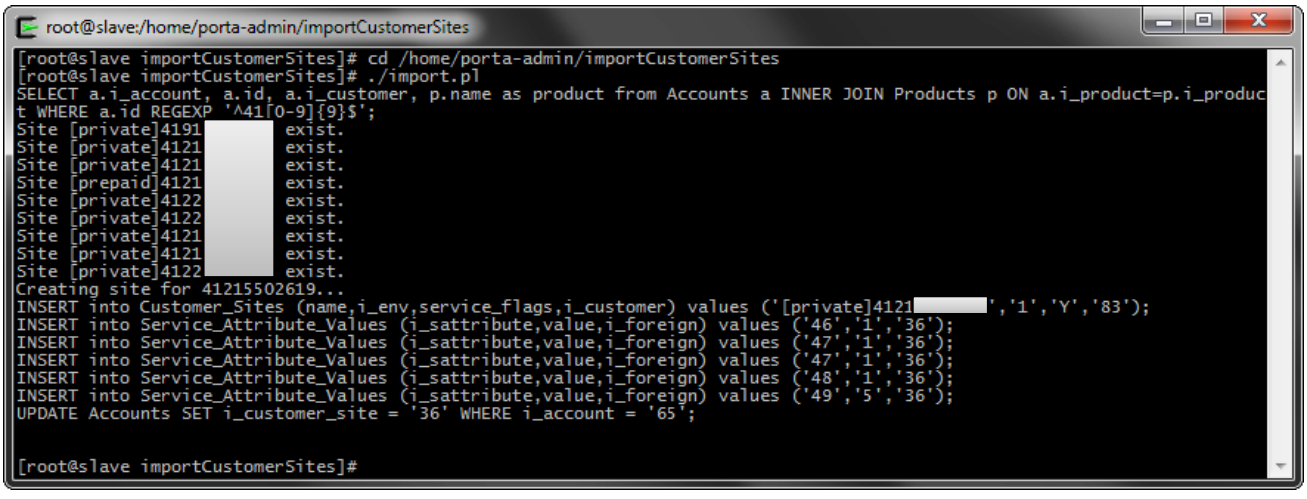
Here you can view the current state of our new customer's sites:



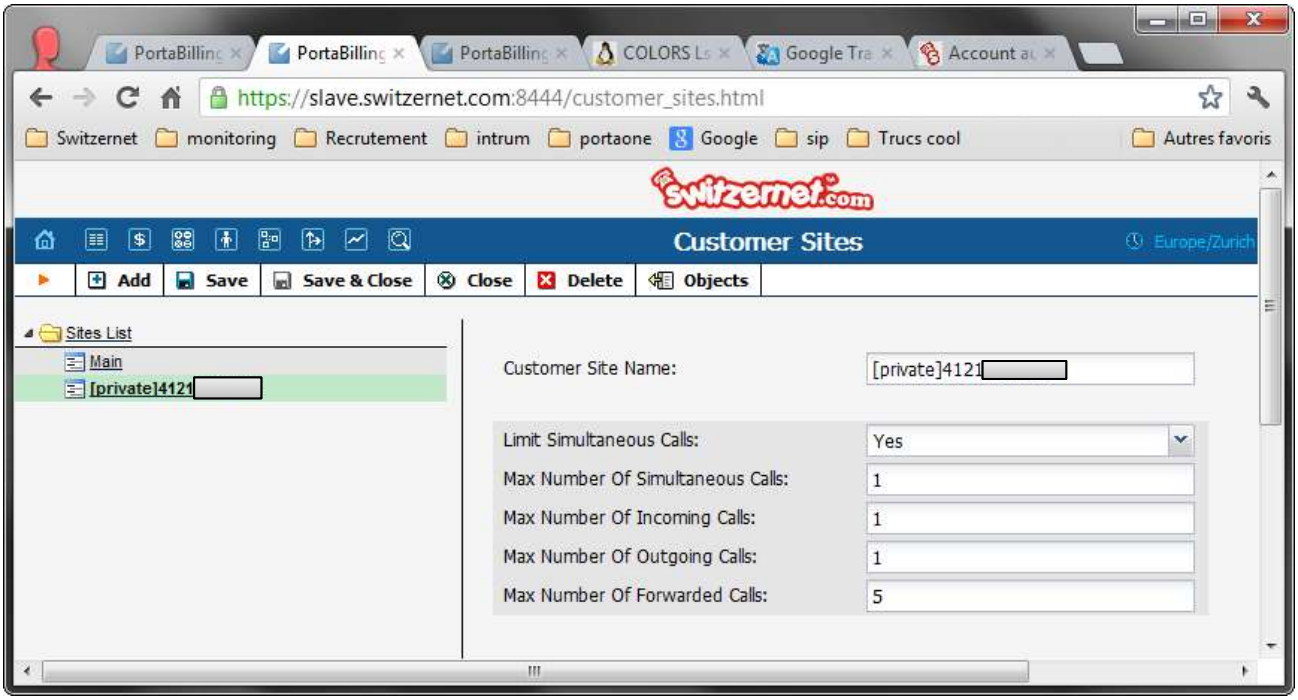


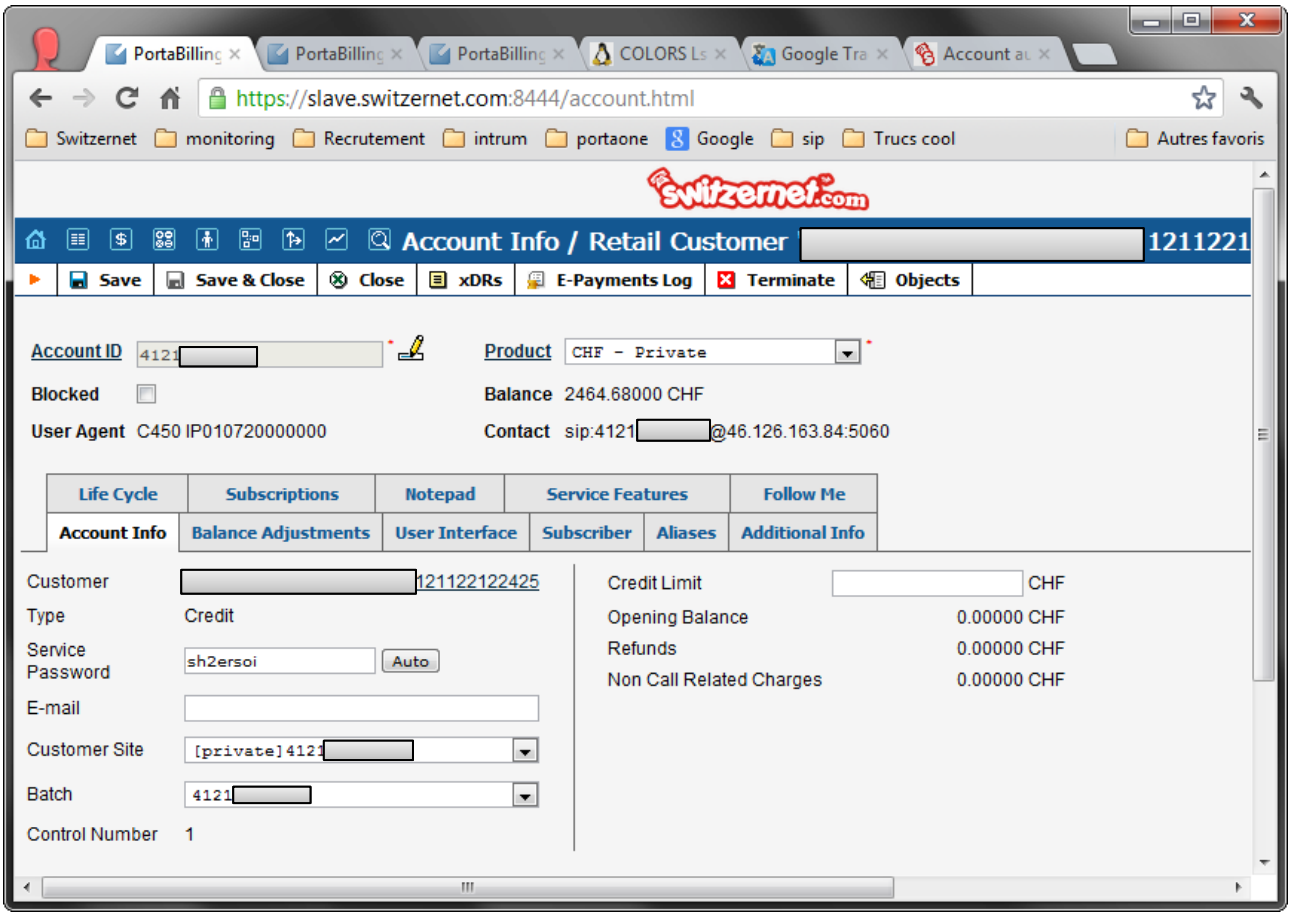
Launch the script '*import.pl*':

```
[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:



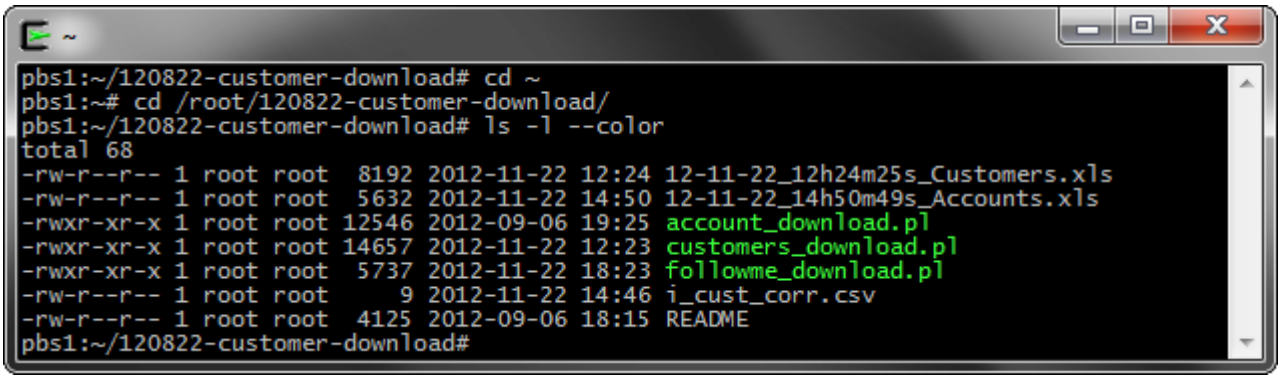


As you can view, the Customer has one site for each account with the type of site (private, business or prepaid). Note that sites for accounts with promotional product have the same name as for account with no promotional product.

Import follow me

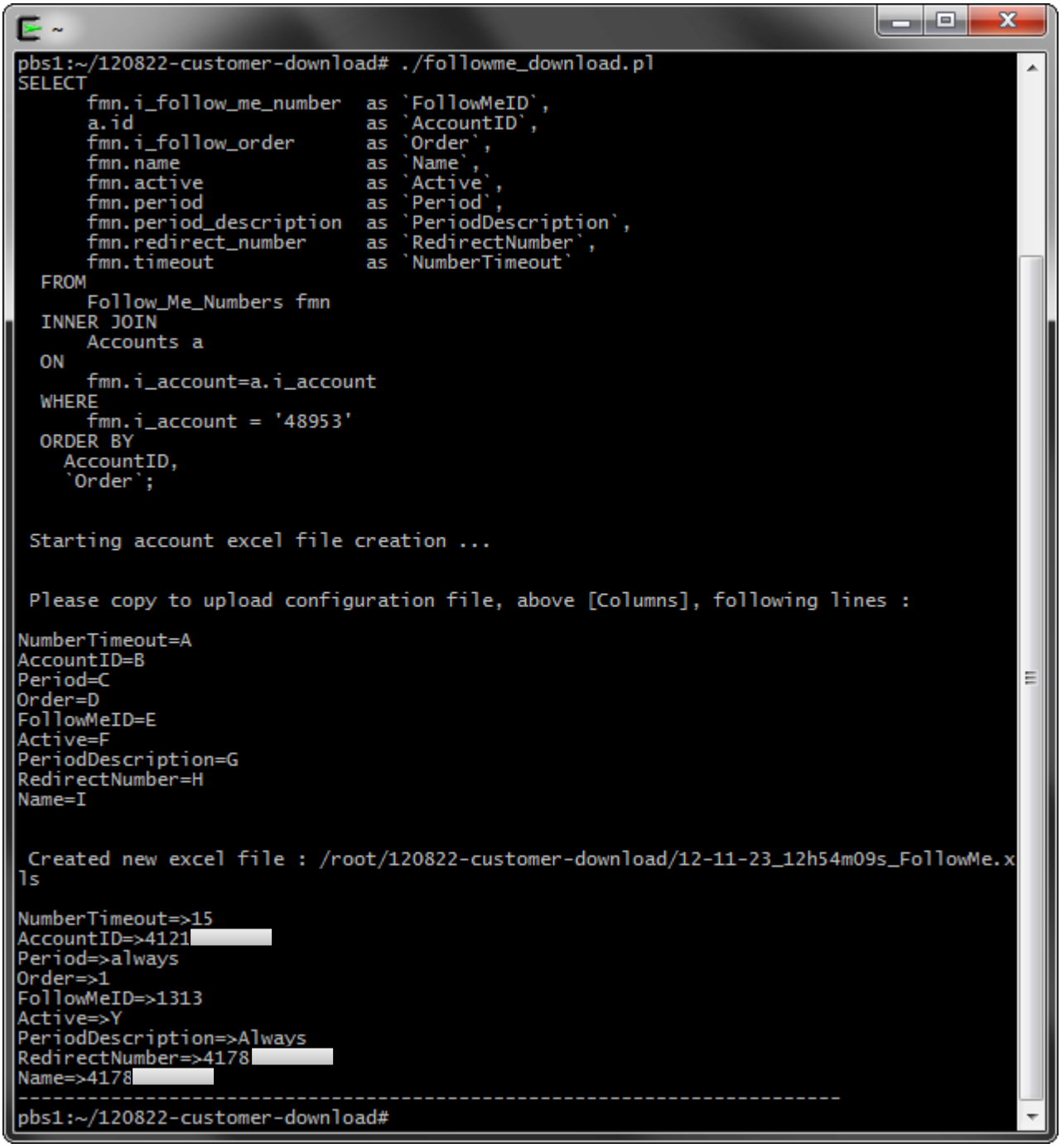
It is now time to import the follow-me. Go in the pbs1 work folder. The script to use this time is 'followme_download.pl'.

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 :

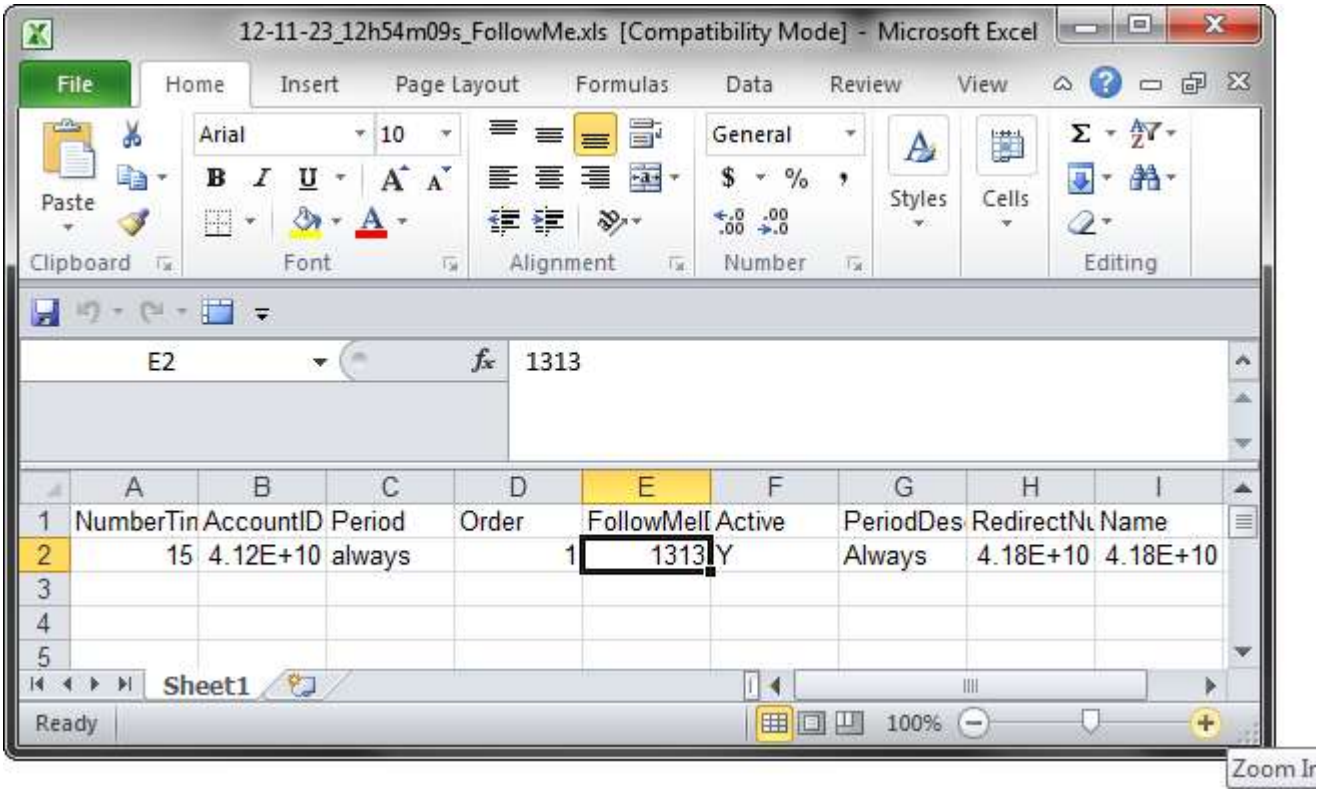
```
[root@slave importCustomerSites]# ./followme_download.pl
```



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

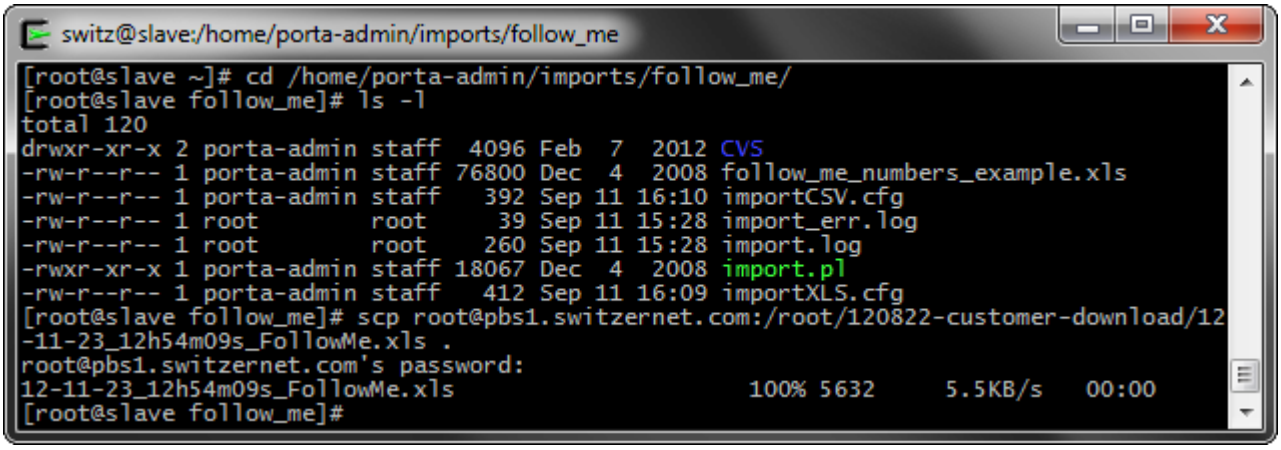
If no error occurs, open a local terminal, download and verify the new excel file:

```
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
```

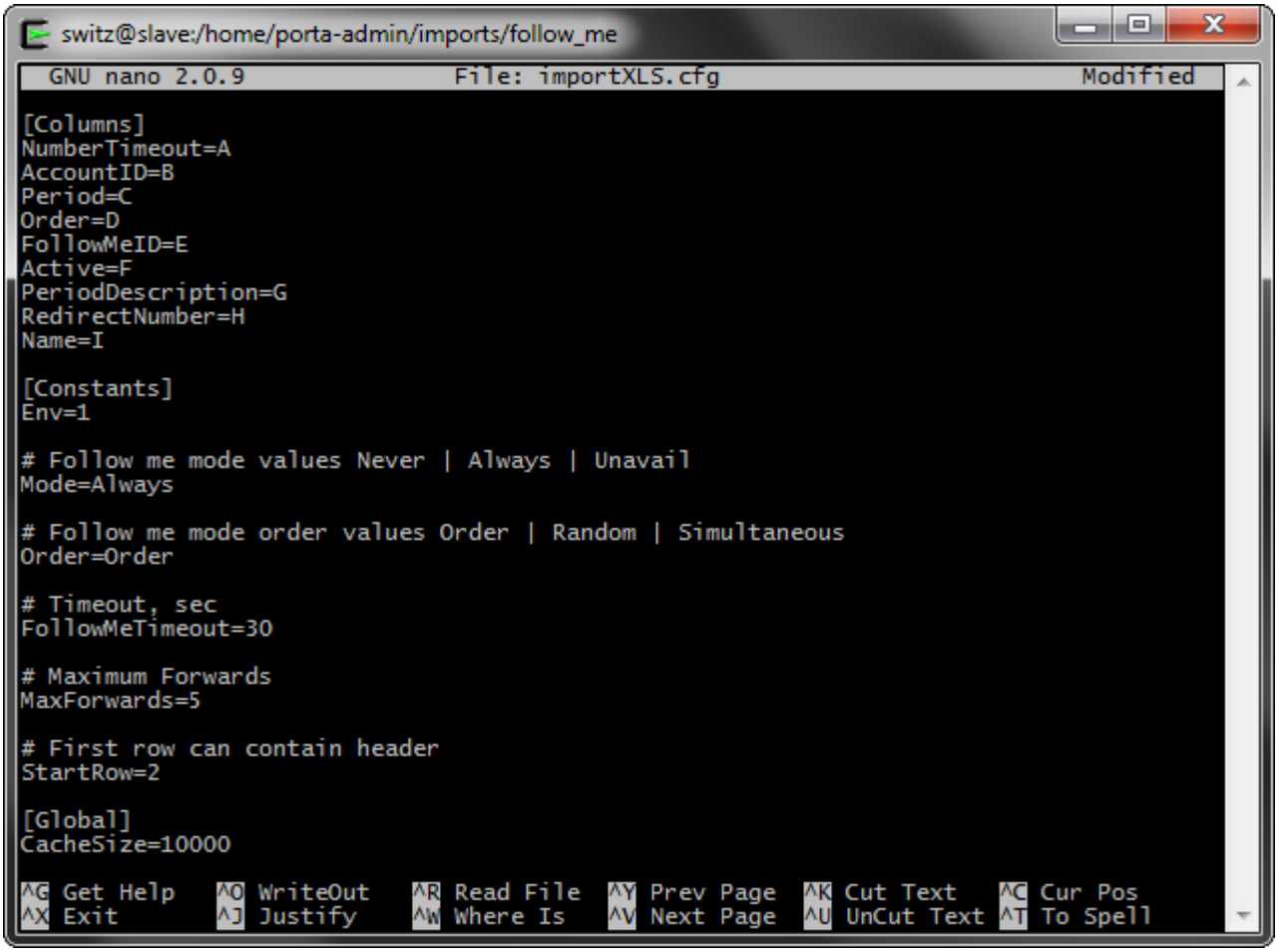


Once verified, go on slave and change directory to '*/home/porta-admin/imports/follow_me/*' and import your new excel file.

```
[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]#
```

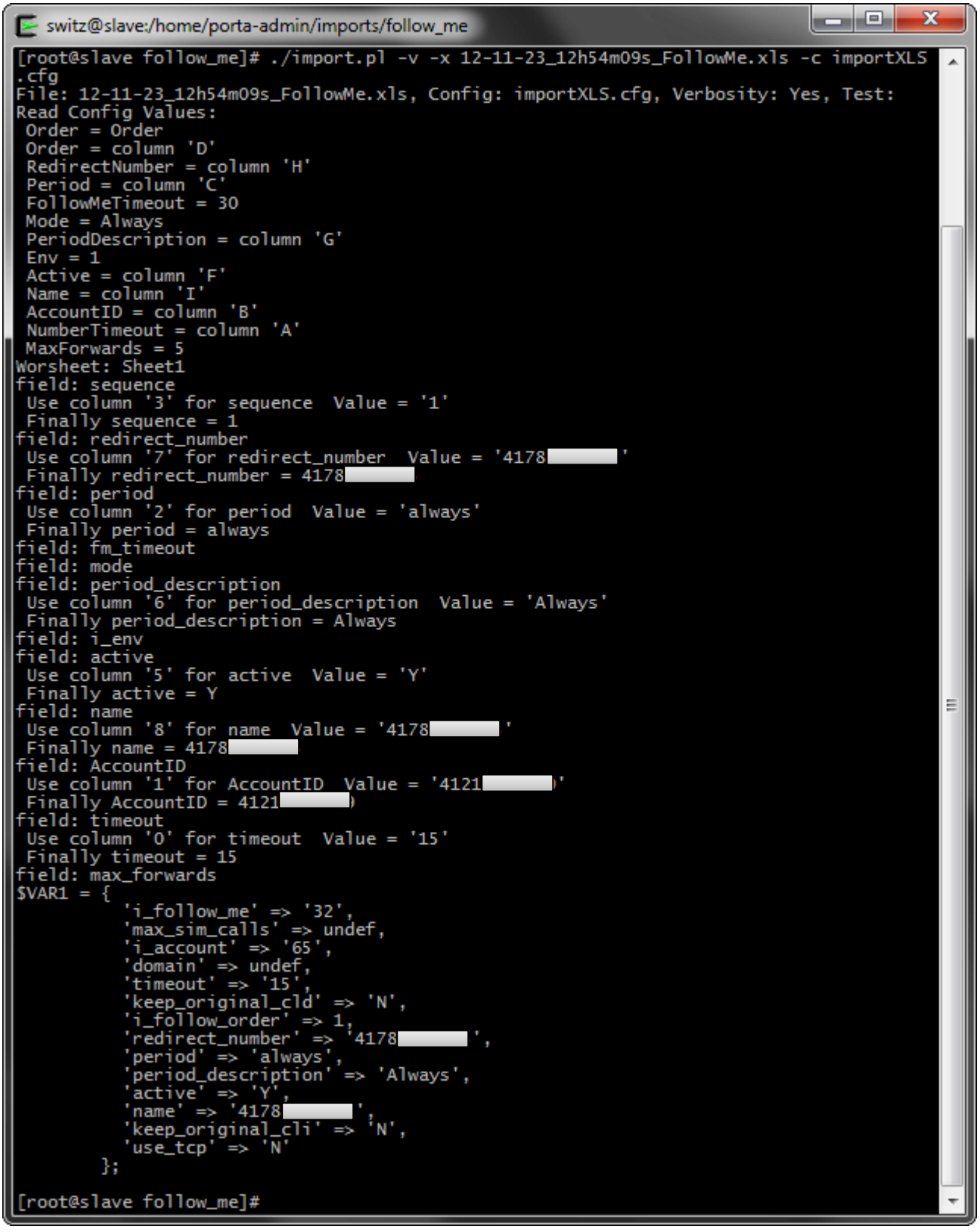


Open the '*importXLS.cfg*' configuration file and past the column list under '*[Columns]*':

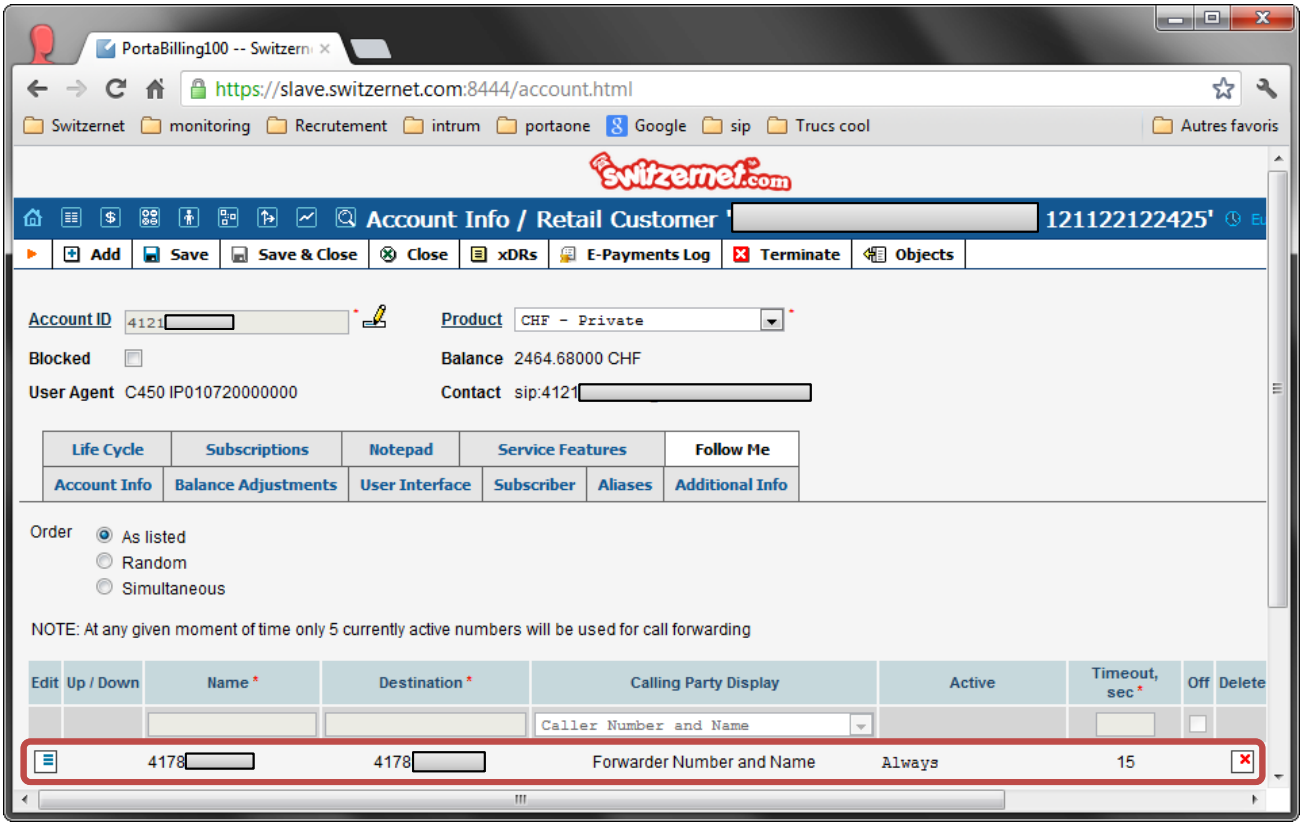


Verify all files are present and run the '*import.pl*' script:

```
[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:



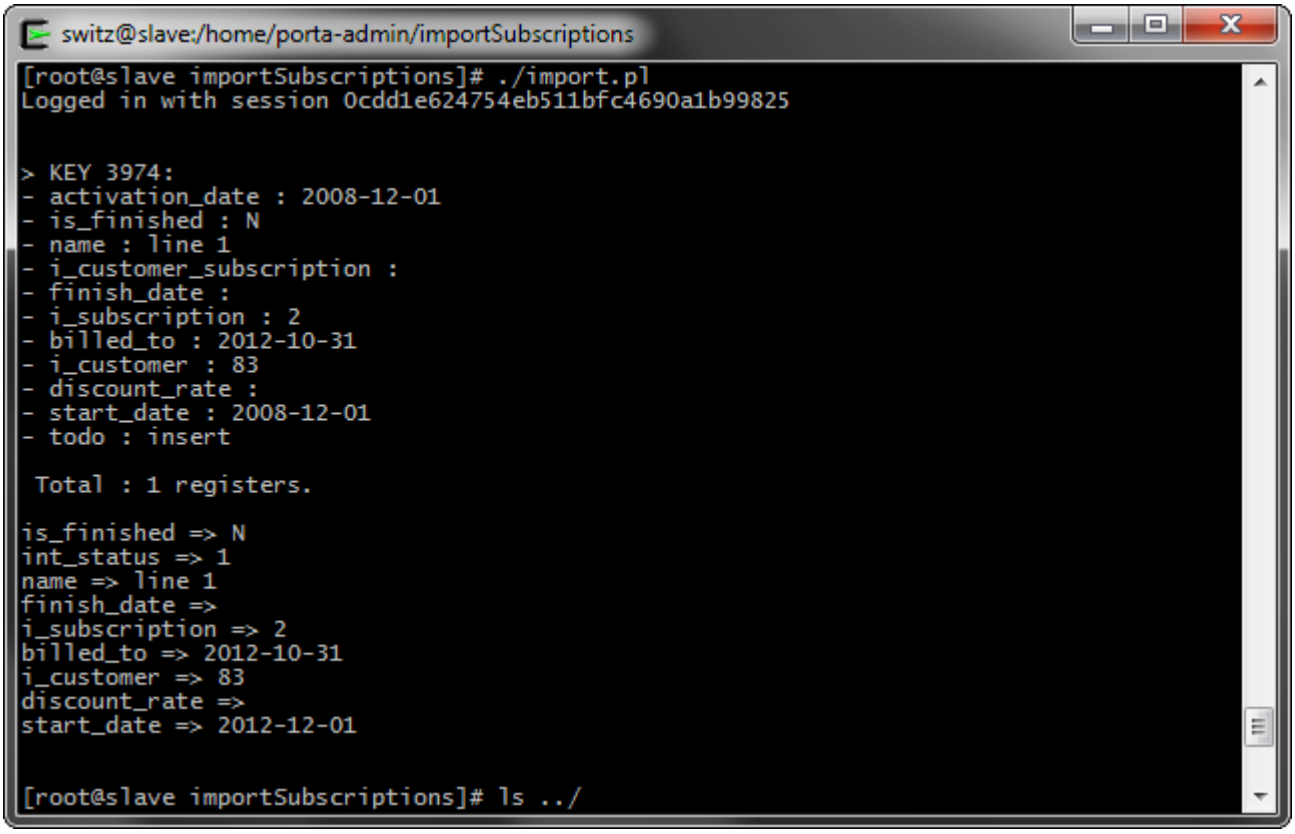
Imports subscriptions

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

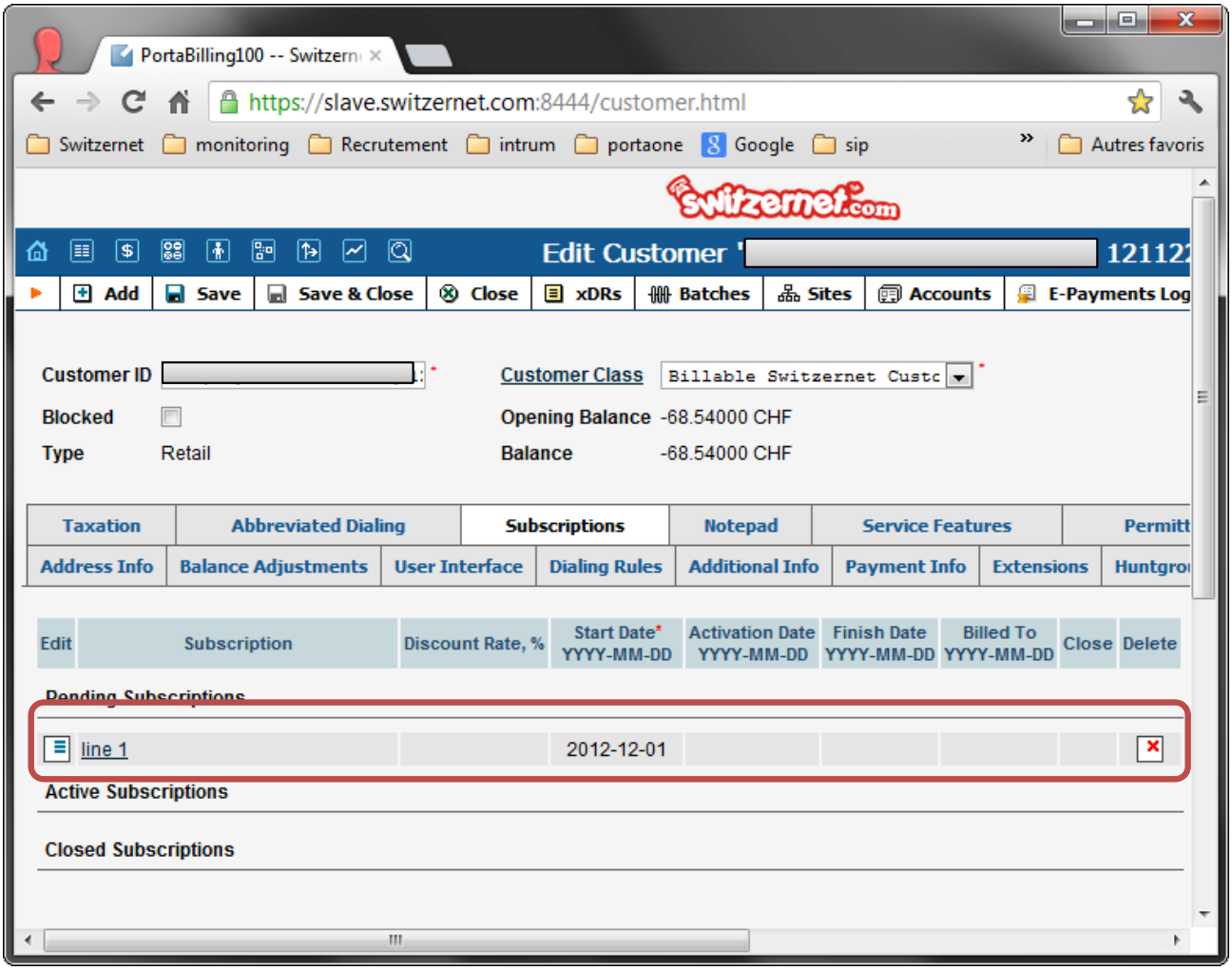
Please note that we cannot set the subscriptions in the past. We have to start it at the first next month.

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/

* * *