# Accman Analysis

# Scripts dealing with Department Data

Richard Gundersen

Mike Minns

### 2nd March 2012

[Accman Analysis 1](#_Toc318795859)

[Scripts dealing with Department Data 1](#_Toc318795860)

[Introduction 6](#_Toc318795861)

[Sources of Departmental Data 7](#_Toc318795862)

[Overview: SAP Import Process 8](#_Toc318795863)

[Banner\_import.bat 9](#_Toc318795864)

[Sap\_import.bat 10](#_Toc318795865)

[sap\_import.pl 11](#_Toc318795866)

[Introduction 11](#_Toc318795867)

[Transactions Generated 11](#_Toc318795868)

[Process 11](#_Toc318795869)

[Banner\_import.bat 13](#_Toc318795870)

[Introduction 13](#_Toc318795871)

[Process 13](#_Toc318795872)

[Staffautostate.pl 15](#_Toc318795873)

[Introduction 15](#_Toc318795874)

[Process 15](#_Toc318795875)

[Banner4import.pl 17](#_Toc318795876)

[Introduction 17](#_Toc318795877)

[In Pictures 17](#_Toc318795878)

[Examples 17](#_Toc318795879)

[Transactions Generated 18](#_Toc318795880)

[Process 18](#_Toc318795881)

[Assign.pl 21](#_Toc318795882)

[Introduction 21](#_Toc318795883)

[Purpose 21](#_Toc318795884)

[Transactions Generated 21](#_Toc318795885)

[Process 21](#_Toc318795886)

[allocateLib.pl 25](#_Toc318795887)

[Introduction 25](#_Toc318795888)

[In Pictures 25](#_Toc318795889)

[Examples 25](#_Toc318795890)

[Depends On 25](#_Toc318795891)

[Methods 25](#_Toc318795892)

[Banner4person.pl 28](#_Toc318795893)

[Introduction 28](#_Toc318795894)

[In Pictures 28](#_Toc318795895)

[Examples 28](#_Toc318795896)

[Depends On 29](#_Toc318795897)

[Process 29](#_Toc318795898)

[Banner4Talis.pl 31](#_Toc318795899)

[Introduction 31](#_Toc318795900)

[In Pictures 31](#_Toc318795901)

[Examples 31](#_Toc318795902)

[Depends On 31](#_Toc318795903)

[Process 31](#_Toc318795904)

[deDuplicateWeekly.pl 33](#_Toc318795905)

[Introduction 33](#_Toc318795906)

[In Pictures 33](#_Toc318795907)

[Examples 33](#_Toc318795908)

[Depends On 33](#_Toc318795909)

[Process 33](#_Toc318795910)

[DisableAccount.pl 35](#_Toc318795911)

[Introduction 35](#_Toc318795912)

[In Pictures 35](#_Toc318795913)

[Examples 35](#_Toc318795914)

[Depends On 35](#_Toc318795915)

[Methods 35](#_Toc318795916)

[EnableAccount.pl 37](#_Toc318795917)

[Introduction 37](#_Toc318795918)

[In Pictures 37](#_Toc318795919)

[Examples 37](#_Toc318795920)

[Depends On 37](#_Toc318795921)

[Methods 37](#_Toc318795922)

[EnableDisable.pl 39](#_Toc318795923)

[Introduction 39](#_Toc318795924)

[In Pictures 39](#_Toc318795925)

[Examples 39](#_Toc318795926)

[Depends On 39](#_Toc318795927)

[Methods 39](#_Toc318795928)

[restoreUFAA.pl 41](#_Toc318795929)

[Introduction 41](#_Toc318795930)

[In Pictures 41](#_Toc318795931)

[Examples 41](#_Toc318795932)

[Depends On 41](#_Toc318795933)

[Process 41](#_Toc318795934)

[StudentAutoState.pl 43](#_Toc318795935)

[Introduction 43](#_Toc318795936)

[In Pictures 43](#_Toc318795937)

[Examples 43](#_Toc318795938)

[Depends On 43](#_Toc318795939)

[Process 43](#_Toc318795940)

[StudentPerson.pl 46](#_Toc318795941)

[Introduction 46](#_Toc318795942)

[In Pictures 46](#_Toc318795943)

[Examples 46](#_Toc318795944)

[Depends On 46](#_Toc318795945)

[Process 46](#_Toc318795946)

[Methods 47](#_Toc318795947)

[Other Scripts 52](#_Toc318795948)

[maillist-staff.pl 52](#_Toc318795949)

[Transactions generated 52](#_Toc318795950)

[maillist-student.pl 52](#_Toc318795951)

[Transactions generated 52](#_Toc318795952)

[mailstaffchanges.pl 52](#_Toc318795953)

[Transactions generated 52](#_Toc318795954)

[Process 52](#_Toc318795955)

[moverename.pl 53](#_Toc318795956)

[noshow.pl 53](#_Toc318795957)

[Transactions generated 54](#_Toc318795958)

[pregen\_associate.pl 54](#_Toc318795959)

[Transactions generated 54](#_Toc318795960)

[selfReg.pl 55](#_Toc318795961)

[staffmultileft.pl 55](#_Toc318795962)

[Transactions generated 55](#_Toc318795963)

[Process 55](#_Toc318795964)

[staffsequential.pl 56](#_Toc318795965)

[Transactions generated 56](#_Toc318795966)

[Register.pl 57](#_Toc318795967)

[Introduction 57](#_Toc318795968)

[Applies account settings from student account assignment and staff self-registration. 57](#_Toc318795969)

[Links an account record to its person record & many other things such as setting up email, setting quotas etc 57](#_Toc318795970)

[Process 57](#_Toc318795971)

[.NET Processes 59](#_Toc318795972)

[TempusCohortCreateUpdateUser 59](#_Toc318795973)

[ActivateDirectoryNewUser 59](#_Toc318795974)

[ImportAssociate 59](#_Toc318795975)

# Introduction

There is no single, definitive source of departmental information at the university. Different systems have their own list of department names, codes etc. Some even have multiple sources of department information.

Accman imports data from various systems such as Banner and SAP, and pushes data out to many more (AD, Blackboard, Talis). While it is performing many of these tasks, it has to deal with the many formats that the departmental data can be in.

Over the years, Accman has evolved various mechanisms to cope with these different sources of departmental data. It contains a number of tables for storing and mapping e.g. SAP departments to UoS departments. This has led to brittle code that is prone to error and is difficult to maintain. It also makes it very hard to accommodate some types of changes that the business might like to make.

It also makes it nearly impossible for the business to restructure departments *without* Accman breaking in some way.

Ideally, there should be one definitive source of departmental data that all systems reference.

This document attempts to show how departmental data is stored and used by Accman.

# Sources of Departmental Data

Below is a list of databases that Accman uses, alongside the tables / columns that contain departmental data.

### Tables

DEPT

SAP\_DEPT 4 different department lists

HR\_DEPARTMENTS

BANNER\_DEPT

DEPARTMENT\_STATUS

DEPT\_PERSON\_ACCOUNT

### Other tables/fields

PERSON.DEPT\_CODE

PERSON.SAP\_DEPT\_CODE

TEMPUSCOHORTPROGRAMMES.COHORT\_DEPT\_ID

TEMPUSCOHORTPROGRAMMES.COHORT\_DEPT\_NAME

Mail

Associates

### Tables

ASSOCIATE.DepartmentCode

### Tables

DEPARTMENT

IDM

# Overview: SAP Import Process

1. SAP writes its export data to the ZACCMAN table. Each record represents a member of staff who has arrived / left / changed departments etc and therefore needs updating in Accman.
2. The sap\_import.bat file runs on a schedule starts the staff\_import.pl script.
3. Sap\_import.pl sees that the ZACCMAN table contains fresh data, it reads the new records.
4. Sap\_import.pl can create or update accounts in Accman depending on certain rules. In both cases, the department code read from the ZACCMAN DEPT table is translated into an Accman-friendly department code, and then written to the person.dept\_code column.

**Go.Bat**

|  |  |  |
| --- | --- | --- |
| **Process** | **Script** | **Parameters** |
| Pregenerate accounts | pregen.pl | Transaction ID = 1 |
| Pregenerate Associates | pregen\_associate.pl | Transaction ID = 88 |
| Quota | quota | Transaction ID = 4 |
| Password | password | Transaction ID = 5 |
| Enable Disable | enableDisable | Transaction ID = 12 |
| Force | forcelogout | Transaction ID = 21 |
| Move Rename | moverename.pl | Transaction ID = 28 |
| Expiry Date | expiryDate | Transaction ID = 33 |
| New Staff | newstaff | Transaction ID = 34 |
| Gen password.txt | password\_txt | Transaction ID = 39 |
| Extend | extend | Transaction ID = 40 |
| Reconcile | reconcile | Transaction ID = 41 |
| Register account | register | Transaction ID = 43 |
| Copy logfile | copylog | Transaction ID = 45 |
| Copy EPP logfile | copyEPPlog | Transaction ID = 46 |
| Assign account to student | assign | Transaction ID = 47 |
| Live@Edu - CreateUsers | create\_users\_processor | Transaction ID = 48 |
| Live@Edu - ChangePassword | change\_password\_processor | Transaction ID = 61 |
| Live@Edu - DisableUsers | disable\_users\_processor | Transaction ID = 62 |
| Live@Edu - EnableUsers | enable\_users\_processor | Transaction ID = 63 |
|  |  |  |

## Banner\_import.bat

|  |  |
| --- | --- |
| **Process** | **Script** |
| Import student data from Banner into AccMan banner\_import table | banner4import.pl |
| Import program data from Banner into AccMan prog table | banner4prog.pl |
| Check for no-shows of 'expected' students | noshow.pl |
| Import from banner\_import into person table | banner4person.pl |
| Enable accounts that have been disabled previously that are now active studentautostate.pl |  |
| Restore future UFAA status of 'left' students (has to be run after banner4person) It probably doesn't matter if this doesn't get through to Talis (users will be 'expected' rather than 'present') | restoreUFAA.pl |
| Import address data from Banner into AccMan banner\_address table | banner4address.pl |
| Export student data and address from AccMan to su\_bbreg\_fullNNN | banner4talis.pl |
| Export staff reconciliation data from AccMan to su\_bbreg\_recNN reconcileTalis.pl |  |
| Export staff data from AccMan to su\_bbreg\_staff | r#talis\_staff.pl |
| FTP su\_bbreg to Talis to be processed by borr\_import prog run overnight by crontab | banner4ftp.pl |

# Sap\_import.bat

Runs at 13:10 every day

Executes the following Perl scripts in order

c:\accman\scripts\scheduled\sap\_import.pl

c:\accman\scripts\scheduled\staffmultipost.pl

c:\accman\scripts\scheduled\staffmultileft.pl

c:\accman\scripts\scheduled\staffsequential.pl

c:\accman\scripts\scheduled\staffautostate.pl

# sap\_import.pl

## Introduction

This script is executed by sap\_import.bat.

Import users from the SAP database into Accman.

## Transactions Generated

97 (Update Janus)

## Process

1. Keeps looping until it spots that the data in the SAP db is at least 4 minutes old (compares the TIMESTAMP field in ZACCMAN with the timestamp in the sap\_import table. This is so we only do anything when there is something worth processing.
2. Delete everything from the sap\_import table so we have a clean slate.
3. Generates a new import ID for this import operation
4. Builds a list of reference data used to map SAP values to Accman values for the following items: Title (Reverend 🡪Rev), Department (CAIT 🡪UI) and Grace (last possible date for leavers to have a chance to return)
5. Execute the following SQL to obtain the accounts needing to be processed

select

PERNR, INITS, SURNAME, FORENAME, TITLE, PREFNAME, MIDNAME, DEPT, SDATE, EDATE, DOB, STATUS, MULTIPOST, TIMESTAMP

from

ZACCMAN

where

MANDT = 200 and

PERNR != '00000000'

1. Inserts the data into the MAIL.SAP\_IMPORT table using the following SQL

insert into sap\_import

values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, encrypt(?, ?), ?, ?, ?)

1. Reads the data back from the MAIL.SAP\_IMPORT table and processes each record it finds
2. Calls updateStaff()
3. Checks that the personnel number is valid (otherwise no further processing is done)
4. Calculate what the person state should be
5. Check that the names are correct
6. Convert the SAP title to an Accman title
7. Convert the SAP department name to an Accman department name
8. We have the personnel number, so look up their person record by looking in the rollmap table (which maps roll/personnel numbers to personIDs
9. If no person record is found, create a new one and generate a new rollnumber for them
   1. Insert the details we have into the person table
   2. Generate a new rollnumber: this is actually just the personID + 200000000 so if the personID was 1234, their rollnnumber would be 200001234.
   3. Insert the new rollnumber into the rollmap table
   4. Update the new person record with the generated rollnumber

insert into person (initials, realname, source\_code, person\_status\_code, dept\_code, forenames, person\_state, importID, first\_importID, title, preferredName, start\_date, end\_date, secret, multi\_post, b\_date) values (?, ?, 'P', 'S', ?, ?, ?, $importID, $importID, ?, ?, ?, ?, ?, ?, ?)

1. Otherwise, if a person record is found
   1. Check that this isn’t a duplicate i.e. there are not two records with the same personnel number in this import
   2. If the person has already left and they have not been extended, don’t proceed any further
   3. If an account is due to be extended, set their state to ‘extended’
   4. Update their details in Accman with the following SQL. This updates their department code

update person

set importID = $importID,

source\_code = 'P',

person\_status\_code = 'S',

initials = ?,

realname = ?,

dept\_code = ?,

sap\_dept\_code = ?,

forenames = ?,

person\_state = ?,

title = ?,

preferredName = ?,

start\_date = ?,

end\_date = ?,

secret = ?,

multi\_post = ?,

b\_date = ?

where personID = ?

1. Creates a Janus (type 97) transaction
2. Mark the import as complete

# Banner\_import.bat

## Introduction

This script is run as a windows scheduled task. There are two run modes (both are scheduled to run)

1. Incremental: every weekday, at 11:30am.
2. Full: every weekday, at 7:30pm

A command line parameter called “-inc” is passed in to specify whether the script should run in full or incremental mode. If “-inc” is present as the first command line parameter, the script will run in *incremental* mode, otherwise it will run in *full* mode.

Looking at the configuration for the scheduled task, it looks something like this (note the “-inc” parameter is passed in this example, signifying that this task runs the script in incremental mode:

*cmd /c start "Banner import" /DC:\accman\logs C:\accman\batch\banner\_import.bat -inc*

## Process

This batch script is a lightweight wrapper around many other Perl scripts. It calls the various scripts in the following order

|  |  |  |
| --- | --- | --- |
| Script Name | Incremental | Purpose |
| banner4imort.pl | X | Copies data from Banner to Accman |
| banner4prog.pl |  | Import program data from Banner into AccMan prog table |
| noshow.pl |  | Check for no-shows of 'expected' students |
| banner4person.pl |  | Import from banner\_import into person table |
| assign.pl |  | Import from banner into banner\_syraccs table and assign any allocated accounts to the correct student |
| studentautostate.pl | X | Enable accounts that have been disabled previously that are now active and disable ones where student has now left or is absent |
| restoreUFAA.pl |  | Restore future UFAA status of 'left' students (has to be run after banner4person) It probably doesn't matter if this doesn't get through to Talis (users will be 'expected' rather than 'present') |
| banner4address.pl |  | Import address data from Banner into AccMan banner\_address table |
| banner4talis.pl |  | Export student data and address from AccMan to su\_bbreg\_fullNNN |
| reconcileTalis.pl |  | Export staff reconciliation data from AccMan to su\_bbreg\_recNN |
| talis\_staff.pl |  | Export staff data from AccMan to su\_bbreg\_staff |
| banner4ftp.pl |  | FTP su\_bbreg to Talis to be processed by borr\_import prog run overnight by crontab |
| banner4talis.pl | X | Export incremental student data from AccMan to su\_bbreg\_incNNN |
| banner4ftp.pl | X | FTP su\_bbreg\_incNNN to Talis to be processed by borr\_import prog run by crontab |

# Staffautostate.pl

## Introduction

Executed as part of the SAP import process. Enable or disable accounts affected by the most recent SAP import depending on their status.

## Process

1. Get the latest import information from the person\_import table
2. Get a list of staff accounts which are candidates for being re-enabled i.e. their account records are disabled

select

a.accID, a.username, p.personID

from

account a, person p, department\_status ds, dept\_person\_account d

where p.personID = a.personID

and p.person\_state = 'present'

and p.source\_code = 'P'

and p.person\_status\_code = 'S'

and a.status\_code = 'S'

and a.state in ('disabled', 'deletable')

and a.accountType = 'personal'

and a.department\_status\_code = ds.department\_status\_code

and p.dept\_code = d.person\_dept\_code

and ds.dept\_code = d.account\_dept\_code

and (a.DisabledReason in ('Person has left', 'User has left - extension expired')

or a.DisabledReason like 'User has left%');

1. See if there are any other valid account records with the same personID but are different to the account already retrieved in step 2. I.e. Bob has two accounts, one that’s disabled (step 2), and one that’s valid (step 3)

select

a.username

from

account a, person p, department\_status ds, dept\_person\_account d

where p.personID = ?

and p.personID = a.personID

and p.person\_status\_code = 'S'

and a.status\_code = 'S'

and a.state = 'valid'

and a.accountType = 'personal'

and a.department\_status\_code = ds.department\_status\_code

and p.dept\_code = d.person\_dept\_code

and ds.dept\_code = d.account\_dept\_code

and a.accID != ?

1. If Bob does have a valid account (retrieved in step 3) then disable the first one (but it’s already disabled isn’t it?) because they are now using the one from step 3. Calls disableAccount() on the account from step 2.
2. If Bob does *not* have another valid account, the one from step 2 can be re-enabled because they must have returned. Calls enableAccount() on the account from step 2
3. Finally, find any accounts that definitely need to be disabled by running the following query. Call disableAccount() on each account that is retrieved.

select a.accID, a.username

from account a, person p

where p.personID = a.personID

and p.person\_state = 'left'

and p.source\_code = 'P'

and a.state = 'valid'

and a.accountType = 'personal'

# Banner4import.pl

## Introduction

This script is executed by banner\_import.bat, which runs as a windows scheduled task. There are two run modes, and both are

1. Incremental: every weekday, at 11:30am.
2. Full: every weekday, at 7:30pm

The purpose of Banner4import.pl is to copy information out of the Banner database, and into the Accman (Mail) database. Once this is done, other scripts carry out further processing.

## In Pictures

Accman

(banner\_import)

Banner

banner4import.pl

Student

data

Student

data

## Examples

1. When run in **incremental** mode, assuming today’s date is 28th Feb 2012

current\_sess = 2012

first\_sess = 2012

term = 201200

maxterm = 201299

So, students will only be retrieved if (among other things) their term code is between 201200 and 201299

1. When run in **full** mode on 28th Feb 2012

current\_sess = 2012

first\_sess = 2011

term = 201100

maxterm = 201299

So, students will only be retrieved if (among other things) their term code is between 201100 and 201299

1. When run in **incremental** mode, assuming today’s date is 10th Nov 2012

current\_sess = 2013

first\_sess = 2013

term = 201300

maxterm = 201399

So, students will only be retrieved if (among other things) their term code is between 201300 and 201399

1. When run in **full** mode on 10th Nov 2012

current\_sess = 2013

first\_sess = 2012

term = 201200

maxterm = 201399

So, students will only be retrieved if (among other things) their term code is between 201200 and 201399

## Transactions Generated

None

## Process

1. Connect to the Mail database
2. Connect to the Banner database
3. Define *current\_sess(ion)* as e.g.
   1. 2012 (if today < 01 July 2012)
   2. 2013 (today > after 30 June 2012).
4. Define *first\_sess(ion)* as
   1. *current\_sess* (if incremental) e.g. 2012
   2. *current\_sess – 1* (otherwise) e.g. 2011
5. Define term as ‘<first\_sess>00’
   1. e.g. 201200
6. Define maxterm as ‘<current\_sess>99’
   1. e.g. 201299
7. Delete all records from the banner import table in the Mail database
   1. Delete from mail.banner\_import
8. Prepare the select script. Reads student data from the BANNER database, ready to be inserted into the MAIL.BANNER\_IMPORT table

SELECT DISTINCT

SPRIDEN\_PIDM,

SPRIDEN\_ID,

SPRIDEN\_LAST\_NAME,

SPRIDEN\_FIRST\_NAME,

SPRIDEN\_MI,

SPBPERS\_BIRTH\_DATE,

SPBPERS\_SEX,

SPBPERS\_NAME\_PREFIX,

SGBSTDN\_TERM\_CODE\_EFF,

SGBSTDN\_STST\_CODE,

SGBSTDN\_LEVL\_CODE,

SGBSTDN\_STYP\_CODE,

SGBSTDN\_DEGC\_CODE\_1,

SGBSTDN\_MAJR\_CODE\_1,

SGBSTDN\_BLCK\_CODE,

SGBSTDN\_DEPT\_CODE,

SGBSTDN\_PROGRAM\_1,

SFBETRM\_AR\_IND,

SFBETRM\_ADD\_DATE,

SFBETRM\_ESTS\_CODE,

SKBHINS\_ENDDATE

FROM

spriden JOIN spbpers

ON spriden\_pidm = spbpers\_pidm

JOIN sgbstdn

ON spriden\_pidm = sgbstdn\_pidm

JOIN sfbetrm

ON spriden\_pidm = sfbetrm\_pidm

LEFT JOIN

(SELECT ins1.skbhins\_pidm, stdn1.sgbstdn\_term\_code\_admit,

MAX(ins1.skbhins\_enddate) AS skbhins\_enddate

FROM skbhins ins1, sgbstdn stdn1

WHERE NOT EXISTS (SELECT 1

FROM skbhins ins2

WHERE ins2.skbhins\_pidm = ins1.skbhins\_pidm

AND ins2.skbhins\_enddate IS NULL

AND ins2.skbhins\_term\_code = ins1.skbhins\_term\_code

AND ins1.skbhins\_active\_ind = 'Y'

AND ins2.skbhins\_active\_ind = 'Y')

AND ins1.skbhins\_term\_code = stdn1.sgbstdn\_term\_code\_admit

AND ins1.SKBHINS\_PIDM = stdn1.sgbstdn\_pidm

GROUP BY ins1.skbhins\_pidm, sgbstdn\_term\_code\_admit

UNION

SELECT ins1.skbhins\_pidm, stdn1.sgbstdn\_term\_code\_admit,

NULL AS skbhins\_enddate

FROM skbhins ins1, sgbstdn stdn1

WHERE EXISTS (SELECT 1

FROM skbhins ins2

WHERE ins2.skbhins\_pidm = ins1.skbhins\_pidm

AND ins2.skbhins\_enddate IS NULL

AND ins2.SKBHINS\_TERM\_CODE = ins1.SKBHINS\_TERM\_CODE

AND ins1.skbhins\_active\_ind = 'Y'

AND ins2.skbhins\_active\_ind = 'Y')

AND ins1.skbhins\_term\_code = stdn1.sgbstdn\_term\_code\_admit

AND ins1.skbhins\_pidm = stdn1.sgbstdn\_pidm) ains

ON spriden\_pidm = skbhins\_pidm AND ains.sgbstdn\_term\_code\_admit = sgbstdn.sgbstdn\_term\_code\_admit

WHERE

sgbstdn\_term\_code\_eff = sfbetrm\_term\_code

AND spriden\_change\_ind IS NULL

AND SUBSTR(sfbetrm\_term\_code,1,4) =

(SELECT

MAX(SUBSTR(sfbetrm\_term\_code,1,4))

FROM

spriden spri2, sfbetrm

WHERE

spri2.spriden\_pidm = spriden.spriden\_pidm

AND sfbetrm\_ests\_code != 'EL'

AND sfbetrm\_term\_code BETWEEN <term> AND <maxterm>

AND spri2.spriden\_pidm = sfbetrm\_pidm)

ORDER BY spriden\_id

1. For each record retrieved in the previous step, insert them into the MAIL.BANNER\_IMPORT table

INSERT INTO BANNER\_IMPORT

(

SPRIDEN\_PIDM,

SPRIDEN\_ID,

SPRIDEN\_LAST\_NAME,

SPRIDEN\_FIRST\_NAME,

SPRIDEN\_MI,

SPBPERS\_BIRTH\_DATE,

SPBPERS\_SEX,

SPBPERS\_NAME\_PREFIX,

SGBSTDN\_TERM\_CODE\_EFF,

SGBSTDN\_STST\_CODE,

SGBSTDN\_LEVL\_CODE,

SGBSTDN\_STYP\_CODE,

SGBSTDN\_DEGC\_CODE\_1,

SGBSTDN\_MAJR\_CODE\_1,

SGBSTDN\_BLCK\_CODE,

SGBSTDN\_DEPT\_CODE,

SGBSTDN\_PROGRAM\_1,

SFBETRM\_AR\_IND,

SFBETRM\_ADD\_DATE,

SFBETRM\_ESTS\_CODE,

SKBHINS\_ENDDATE

)

VALUES (<values retrieved from the select statement>)

# Assign.pl

## Introduction

Assign is a Perl script that is executed as a standard transaction processor. It is started via go.bat and once running, periodically checks for any type 47 transactions in the Mail database.

## Purpose

The purpose of this script is to read account information that has recently come into Accman via the *banner4import* process (described in another section in this document), and converts that loosely-structured information into proper account/person records.

## Transactions Generated

None

## Process

1. Executes the following SQL query

SELECT

SYRACCS\_ROWNUM, SYRACCS\_ACCMAN\_STATUS, SYRACCS\_USERNAME, SYRACCS\_PIDM, SYRACCS\_DEPT\_CODE, SYRACCS\_LEVL\_CODE

FROM

SYRACCS

WHERE

SYRACCS\_SIS\_STATUS = 'A' and SYRACCS\_ACCMAN\_STATUS in ('N', 'R') and ROWNUM = 1

1. Get the number of records affected by running a similar SQL query (with a count this time). Stored in *count\_matching\_records*.

SELECT

COUNT(\*)

FROM

SYRACCS

WHERE

SYRACCS\_SIS\_STATUS = 'A' and SYRACCS\_ACCMAN\_STATUS IN ('N', 'R')

1. For each account retrieved in step 1, do the following
   1. Get the student details based on the PIDM [processAccount, getStudentDetails]

select distinct

SPRIDEN\_ID, SPRIDEN\_LAST\_NAME, SPRIDEN\_FIRST\_NAME, SPRIDEN\_MI,

SPBPERS\_BIRTH\_DATE, SPBPERS\_SEX, SPBPERS\_NAME\_PREFIX

from

SPRIDEN join SPBPERS on SPRIDEN\_PIDM = SPBPERS\_PIDM(+)

where

SPRIDEN\_CHANGE\_IND is NULL

and SPRIDEN\_PIDM = ?

* 1. Get the course details for the student

select distinct

SGBSTDN\_TERM\_CODE\_EFF, SGBSTDN\_STST\_CODE, SGBSTDN\_LEVL\_CODE, SGBSTDN\_STYP\_CODE,

SGBSTDN\_DEGC\_CODE\_1, SGBSTDN\_MAJR\_CODE\_1, SGBSTDN\_BLCK\_CODE, SGBSTDN\_DEPT\_CODE, SGBSTDN\_PROGRAM\_1

from SGBSTDN S1

where SGBSTDN\_TERM\_CODE\_EFF =

(select max(SGBSTDN\_TERM\_CODE\_EFF) from SGBSTDN S2 where S1.SGBSTDN\_PIDM = S2.SGBSTDN\_PIDM and S2.SGBSTDN\_DEPT\_CODE = ? and S2.SGBSTDN\_LEVL\_CODE = ?)

and SGBSTDN\_PIDM = ?

* 1. If we have a value for SGBSTDN\_TERM\_CODE\_EFF, retrieve the student’s registration details for this term if they exist and they are not ‘EL’ (eligible to register).

select distinct SFBETRM\_AR\_IND, SFBETRM\_ADD\_DATE, SFBETRM\_ESTS\_CODE

from SFBETRM

where SFBETRM\_TERM\_CODE = ? and SFBETRM\_ESTS\_CODE != 'EL' and SFBETRM\_PIDM = ?

* 1. If the course details exist i.e. we have a value for SGBSTDN\_TERM\_CODE\_EFF, get the student’s end date

select COUNT(\*) from SKBHINS where SKBHINS\_ENDDATE is NULL and SKBHINS\_ACTIVE\_IND = 'Y' and SKBHINS\_PIDM = ?

If the query above returns 0 (because there is not SKBHINS record) or there is an end date, try to get the end date using this query

select MAX(SKBHINS\_ENDDATE)

from SKBHINS

where SKBHINS\_ACTIVE\_IND = 'Y' and SKBHINS\_PIDM = ?

* 1. If we don’t have a value for SGBSTDN\_TERM\_CODE\_EFF, we need to set certain values ourselves (some of which are hardcoded, and some of which come from the SYRACCS table (see earlier queries)

$SGBSTDN\_TERM\_CODE\_EFF = '000000';

$SGBSTDN\_STST\_CODE = 'AS';

$SGBSTDN\_LEVL\_CODE = $SYRACCS\_LEVL\_CODE;

$SGBSTDN\_STYP\_CODE = '';

$SGBSTDN\_DEGC\_CODE\_1 = '';

$SGBSTDN\_MAJR\_CODE\_1 = '';

$SGBSTDN\_BLCK\_CODE = '';

$SGBSTDN\_DEPT\_CODE = $SYRACCS\_DEPT\_CODE;

$SGBSTDN\_PROGRAM\_1 = '';

* 1. **Note**: at this point, we can say that the department code has come either from the SGBSTDN table (see the banner4import.pl script) or if it wasn’t available from there, from the SYRACCS table.
  2. If we don’t have a value for SFBETRM\_ESTS\_CODE at this point, set a few default values

$SFBETRM\_AR\_IND = 'N';

$SFBETRM\_ADD\_DATE = datenow ();

$SFBETRM\_ESTS\_CODE = 'EL';

* 1. We can now start constructing Accman data based on what we retrieved from Banner. This is done using methods in the **studentPerson.pl** script. There is a separate section in the document that explains this script in detail.
  2. First, create the person record by executing:

**studentPerson.parsePerson**

* 1. Check that the rollnumber has a value. If not, this record is invalid and we should not proceed any further with this student.
  2. Next, work out what the Accman values for their course should be by executing

**studentPerson.parseCourse**

* 1. Get the Accman values for their registration details by executing

**studentPerson.parseRegistration**

* 1. Now retrieve the ID of the last successful import

select max(importID) from person\_import where source\_code = 'S' and completed = 'Y'

* 1. *Reject*: the three methods above (parsePerson, parseCourse, parseRegistration) make a note of any student data that for some reason is deemed invalid. Any invalid fields are written to a log file for checking (presumably)
  2. Call **studentPerson.updateStudentPerson**, which takes all the values we have derived from the Banner database, and writes them to the Accman table

1. Get the acID, personID & state fields from the account table where the username matches the username stored in the SYRACCS table. Note: this will only continue if an account is found. The account must exist, however the person record attached to the account does not necessarily need to exist. Calls the method **current()** to work this out**.**
2. If the account ID exists…

If PersonID = -1

SUCCESS: Call **assignAccount()** to assign a new account, this is a new person

If account.PersonID != personID

FAIL: log warning, the account is assigned to someone else

If *state* matches 'new', 'printed', or 'used'

SUCCESS: Call **assignAccount()** to assign a new account

If *state* matches 'valid', 'assigned'

SUCCESS: no need to assign the account, its already done

If *state* matches: 'disabled', 'deletable', 'expired', 'failed'

SUCCESS: account needs to be reactivated

Calls **enableAccount()**

If *state* matches 'deleting', 'deleted', 'invalid'

FAIL: account cannot be enabled

else

FAIL: generic error message

1. Disable any other person records that this account might have been linked to
   1. Execute the following SQL query

select

accID, username

from

account a

where

(state = 'valid' or (state = 'disabled' and (disabledReason in ('Student has left', 'Student is absent', 'User has left - extension expired', 'Student has probably left') or (disabledReason like 'Now using%' and disabledReason != concat('Now using ', ?)) or disabledReason like 'Already using%'))) and accountType = 'personal' and personID = ? and accID != ?

* 1. Call the standard **disable()** method to disable each returned record

1. Update the SISassign table to log the outcome of this assign record (for auditing).
2. Finally, update the banner SYRACCS table

update

SYRACCS

set

SYRACCS\_ACCMAN\_STATUS = 'A',

SYRACCS\_ACTIVITY\_DATE = SYSDATE

where

SYRACCS\_ROWNUM = ?

# allocateLib.pl

## Introduction

Routines to allocate and optionally update accounts in SYRACCS

## In Pictures

## Examples

## Depends On

### Perl Packages

* Warnings
* Strict
* DBI

### Modules

### Tables

* MAIL
  + ACCOUNT
  + ACCOUNT\_DETAILS
  + DEPARTMENT\_STATUS
  + PERSON

## Methods

1. *newAccountToSYRACCS ()*
   1. Get account information where account type is NDS for a specified student
      1. select plain\_text\_password, d.dept\_code, a.status\_code, state, PIDM from account a join account\_details ad on (a.accID = ad.accID) join department\_status d on (d.department\_status\_code = a.department\_status\_code) left join person p on (a.personID = p.personID) where ad.type = 'NDS' and username = ?
   2. Filter out any accounts which aren’t ‘new’ or aren’t Student accounts
   3. Determine the Banner Dept Code and Level
      1. Via *deptAndLevel()*
   4. Read the current Banner account information for the student
      1. select SYRACCS\_ROWNUM, SYRACCS\_ACCMAN\_STATUS, SYRACCS\_PIDM, SYRACCS\_SIS\_STATUS, SYRACCS\_INIT\_PASSWORD, SYRACCS\_DEPT\_CODE, SYRACCS\_LEVL\_CODE from SYRACCS where SYRACCS\_USERNAME = ?
   5. If the account does not exist in Banner create it
      1. insert into SYRACCS (SYRACCS\_USERNAME, SYRACCS\_ACCMAN\_STATUS, SYRACCS\_PIDM, SYRACCS\_SIS\_STATUS, SYRACCS\_INIT\_PASSWORD, SYRACCS\_DEPT\_CODE, SYRACCS\_LEVL\_CODE, SYRACCS\_ACTIVITY\_DATE) values (?, ?, ?, ?, ?, ?, ?, SYSDATE)
   6. If the account does exist in Banner and ????? no PIDM and $SYRACCS\_ACCMAN\_STATUS ne 'N' or $SYRACCS\_SIS\_STATUS ne 'N' or $SYRACCS\_DEPT\_CODE ne $dept\_code or $SYRACCS\_LEVL\_CODE ne $level\_code the Update the Banner account
      1. update SYRACCS set SYRACCS\_USERNAME = ?, SYRACCS\_ACCMAN\_STATUS = ?, SYRACCS\_PIDM = ?, SYRACCS\_SIS\_STATUS = ?, SYRACCS\_INIT\_PASSWORD = ?, SYRACCS\_DEPT\_CODE = ?, SYRACCS\_LEVL\_CODE = ?, SYRACCS\_ACTIVITY\_DATE = SYSDATE where SYRACCS\_ROWNUM = ?
2. *studentAccountToSYRACCS ()*
   1. Get the Student account information
      1. select ' ' as 'password', d.dept\_code, a.status\_code, state, PIDM from account a join department\_status d on (d.department\_status\_code = a.department\_status\_code) left join person p on (a.personID = p.personID) where username = ?
   2. Skip account unless:
      1. It has a PIDM
      2. The account state is VALID, DISABLED or DELETABLE
      3. The account status code indicates it is a Student
   3. Determine the Banner Dept Code and Level
      1. Via *deptAndLevel()*
   4. Skip any student where the pidm, DEPT CODE AND LEVEL already exist for a different username.
      1. select SYRACCS\_ROWNUM from SYRACCS where SYRACCS\_PIDM = ? and SYRACCS\_DEPT\_CODE = ? and SYRACCS\_LEVL\_CODE = ? and SYRACCS\_USERNAME != ?
   5. Check if the username already exists
      1. select SYRACCS\_ROWNUM, SYRACCS\_ACCMAN\_STATUS, SYRACCS\_PIDM, SYRACCS\_SIS\_STATUS, SYRACCS\_INIT\_PASSWORD, SYRACCS\_DEPT\_CODE, SYRACCS\_LEVL\_CODE from SYRACCS where SYRACCS\_USERNAME = ?
   6. If it doesn’t exist insert it:
   7. If it does exist AND
      * 1. The PIDM matches AND
        2. The dept code has changed OR
        3. The level has changed OR
        4. The AccMan status not ‘A’ OR
        5. The SIS status is not ‘A’
      1. Update it
         1. update SYRACCS set SYRACCS\_USERNAME = ?, SYRACCS\_ACCMAN\_STATUS = ?, SYRACCS\_PIDM = ?, SYRACCS\_SIS\_STATUS = ?, SYRACCS\_INIT\_PASSWORD = ?, SYRACCS\_DEPT\_CODE = ?, SYRACCS\_LEVL\_CODE = ?, SYRACCS\_ACTIVITY\_DATE = SYSDATE where SYRACCS\_ROWNUM = ?
3. *deptAndLevel()*
   1. Translate AccMan dept codes and levels to Banner equivalents
      1. If AccMan Status\_Code ‘P’ use Banner level code ‘PG’ else use ‘UG’
      2. If AccMan dept code = ‘AF’ and Banner level code = ‘UG’ use Banner Dept Code = ‘JA’
4. *prepare\_allocate()*
   1. define SQL statements
5. *finish\_allocate()*
   1. release DB handles.

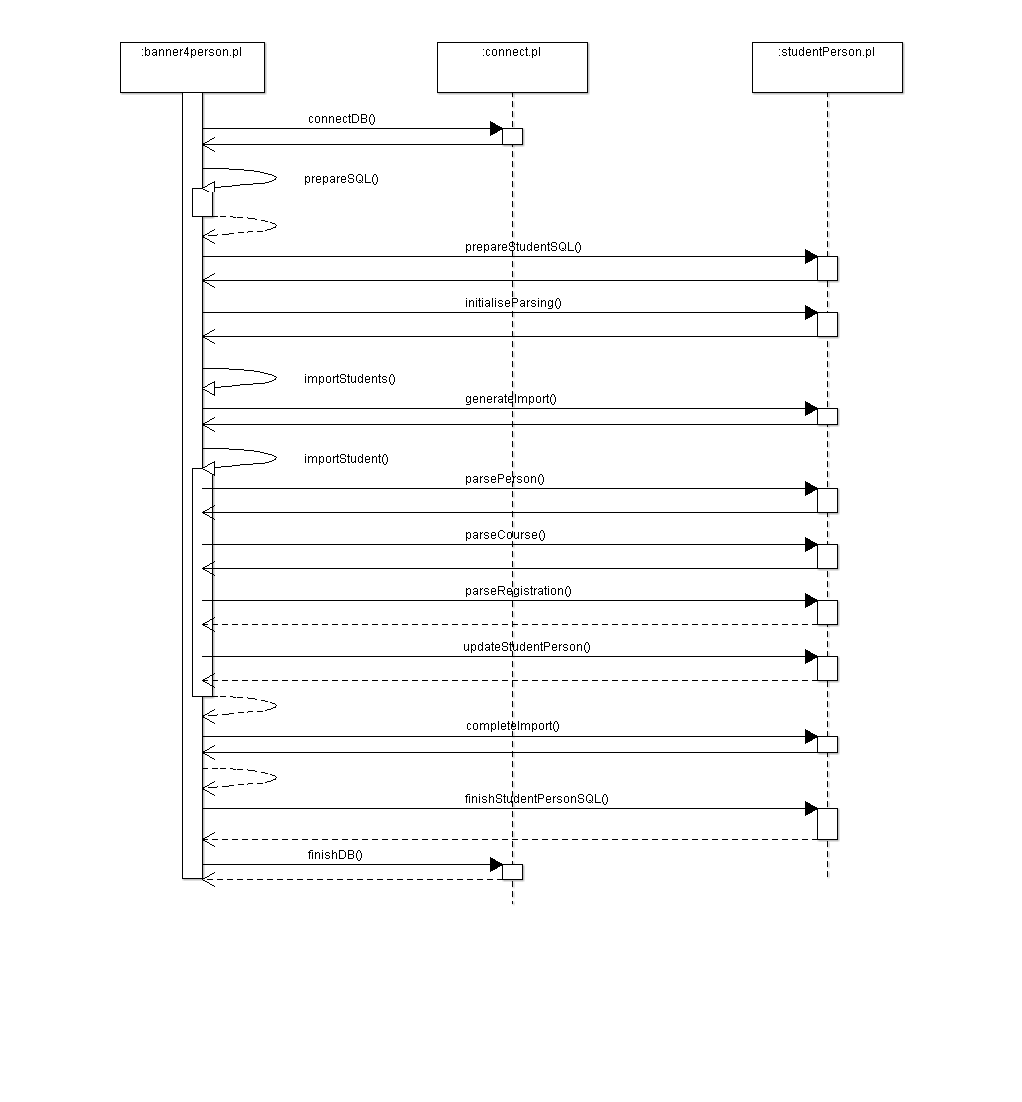
# Banner4person.pl

## Introduction

This script reads student information from the MAIL.BANNER\_IMPORT table.

It parses each Student record it finds; for each record it finds it converts Banner reference information to its AccMan equivalent and then either inserts a new MAIL.PERSON record or updates an existing one.

## In Pictures



## Examples

TBC

## Depends On

### Perl Packages

* Warnings
* Strict
* DBI

### Libraries

* **connect.pl**
* **studentPerson.pl**
* **paths.pl**
* **trLogging.pl**

### Database Tables

* MAIL
  + BANNER\_IMPORT

## Process

1. Define local variables
2. Connect to Banner database
   1. Via *connectDB()* in **connect.pl**
3. Define SQL handles to read students imported from Banner held in MAIL.BANNER\_IMPORT
   1. Via *PrepareSQL()*
4. Define SQL handles to read and write Student information
   1. Via *prepareStudentPersonSQL()* in **studentPerson.pl**.
5. Initialise reference data for Student Titles, States and Department code mappings
   1. Via *initialiseParsing()* in **studentPerson.pl**
6. Import Student Information
   1. Via *importStudents()*
   2. Record the start of the import process
      1. Via *generateImport()* in **studentPerson.pl**
   3. Read Student Information from MAIL.BANNER\_IMPORT:
      1. select SPRIDEN\_PIDM, SPRIDEN\_ID, SPRIDEN\_LAST\_NAME, SPRIDEN\_FIRST\_NAME, SPRIDEN\_MI, SPBPERS\_BIRTH\_DATE, SPBPERS\_SEX, SPBPERS\_NAME\_PREFIX, SGBSTDN\_TERM\_CODE\_EFF, SGBSTDN\_STST\_CODE, SGBSTDN\_LEVL\_CODE, SGBSTDN\_STYP\_CODE, SGBSTDN\_DEGC\_CODE\_1, SGBSTDN\_MAJR\_CODE\_1, SGBSTDN\_BLCK\_CODE, SGBSTDN\_DEPT\_CODE, SGBSTDN\_PROGRAM\_1, SFBETRM\_AR\_IND, SFBETRM\_ADD\_DATE, SFBETRM\_ESTS\_CODE, SKBHINS\_ENDDATE from banner\_import
   4. For each Student Record update AccMan Student Details
      1. Via *updateStudentDetails()*
      2. Convert or reject a Student’s Personal Details from Banner to AccMan equivalents. E.g. tidy names, select AccMan person title and clean birth dates.
         1. Via *parsePerson()* in **studentPerson.pl**
      3. Convert or reject a Student’s Course Details from Banner to AccMan equivalents. E.g. determine student state, select AccMan department codes and set student codes.
         1. Via *parseCourse()* in **studentPerson.pl**
      4. Convert or reject a Student RegistrationDetails from Banner to AccMan equivalents. E.g. determine student registration state from enrolment status
         1. Via *parseRegistration()* in **studentPerson.pl**
      5. Adjust person state for Nursing UGs
      6. Log any rejected Students where their registration state is ‘present’
      7. Ignore any students with registration status = ‘EL’ or with an invalid Student Id. Update Student Information held in AccMan
         1. Via *updateStudentPerson()* in **studentPerson.pl**
   5. Record the completion of the import process.
      1. Via *completeImport()* in **studentPerson.pl**
7. Close Student Information SQL handles
   1. Via *finishStudentPersonSQL()* in **studentPerson.pl**
8. Disconnect from Banner database
   1. Via *finishDB()* in **connect.pl**

# Banner4Talis.pl

## Introduction

For students, generate Talis Borrower Import file from Accman data.

Extracts selected data fields from all student account records in Accman

## In Pictures

## Examples

## Depends On

### Perl Packages

* Warnings
* Strict
* DBI

### Modules

### Tables

* MAIL
  + PERSON\_IMPORT
  + PERSON
  + DEPT
  + ACCOUNT
  + TRANSACTIONS

## Process

1. Connect to AccMan Database
   1. Via *connect\_db()*
2. Get the ID of the last successful Banner Import
   1. select importID from person\_import where source\_code = 'S' and completed = 'Y' order by importID desc limit 1
3. Open an export file
4. Generate SQL statements
5. Get a list of all imported students
   1. select per.RealName, per.rollnumber, per.SPRIDEN\_ID, per.person\_status\_code, per.forenames, per.title, per.dept\_code, per.leaver\_flag, per.stu\_att, dpt.site\_code from person per, dept dpt where per.dept\_code = dpt.dept\_code and per.importID = $importID and per.first\_importID = $first\_importID
6. For each Student
   1. Get email address
      1. Via *get\_email\_address()*
   2. Get Talis status
      1. Via *get\_talis\_special\_status()*
   3. Get Tails email optout
      1. Via *get\_talis\_email\_optout()*
   4. Write the student information to the export file.
      1. Via *print\_borrower\_type\_id(), print\_expiry\_date\_future(), print\_borrower\_site(), print\_borrower\_surname(), print\_borrower\_title(), print\_borrower\_forenames(), print\_borrower\_department(), print\_home\_and\_term\_addresses(), print\_borrower\_registration\_number(), print\_email\_address()*
7. Get a list of all students who have had their NDS accounts have been disabled recently.
   1. select per.RealName, per.rollnumber, per.SPRIDEN\_ID, per.person\_status\_code, per.forenames, per.title, per.dept\_code, per.leaver\_flag, per.stu\_att, dpt.site\_code from person per, dept dpt, account acc, transactions tx where per.dept\_code = dpt.dept\_code and acc.personID = per.personID and acc.status\_code < 'R' and tx.affectsID = acc.accID and tx.tr\_type\_id = 12 and tx.tr\_state = 'processed' and tx.tr\_when >= '$first\_date\_string'
8. For each Student
   1. If it has already been processed by the code above then skip it.
      1. select per.RealName, per.rollnumber, per.SPRIDEN\_ID, per.person\_status\_code, per.forenames, per.title, per.dept\_code, per.leaver\_flag, per.stu\_att, dpt.site\_code from person per, dept dpt where per.dept\_code = dpt.dept\_code and per.importID = $importID and per.first\_importID = $first\_importID and per.rollnumber = ?
   2. Get email address
      1. Via *get\_email\_address()*
   3. Get Talis status
      1. Via *get\_talis\_special\_status()*
   4. Get Tails email optout
      1. Via *get\_talis\_email\_optout()*
   5. Write the student information to the export file.
      1. Via *print\_borrower\_type\_id(), print\_expiry\_date\_future(), print\_borrower\_site(), print\_borrower\_surname(), print\_borrower\_title(), print\_borrower\_forenames(), print\_borrower\_department(), print\_home\_and\_term\_addresses(), print\_borrower\_registration\_number(), print\_email\_address()*
9. Disconnect from AccMan Database
   1. Vi *disconnect\_DB()* in **connect.pl**

# deDuplicateWeekly.pl

## Introduction

Disable all valid personal student accounts where person has left

Disable all personal student accounts in the wrong school or at the wrong level (provided the student has a valid account at the correct school and level)

## In Pictures

## Examples

## Depends On

### Perl Packages

* Warnings
* Strict
* DBI

### Modules

* Connect.pl
* disableAccount.pl
* paths.pl

### Tables

* MAIL
  + PERSON
  + ACCOUNT
  + DEPARTMENT\_STATUS

## Process

1. Connect to AccMan database
   1. Via connectDB() in **connect.pl**
2. Prepare SQL statements
   1. Via *prepare\_SQL()*
3. Disable accounts for students who have left
   1. Via *disableLeft()*
   2. Read list of students who have left
      1. select a.accID, a.username from person p, account a where p.personID = a.personID and p.source\_code = 'S' and p.person\_state in ('left', 'missing') and a.state = 'valid' and a.accountType = 'personal'
   3. Disable the account of each student who has left.
      1. Vai *disableAccount*() in **disableAccount.pl**
4. Disable duplicate accounts for students who have moved school/level
   1. Via *disableDuplicates()*
   2. Read a list of students who have duplicate accounts where one has the wrong dept/level
      1. select a1.username, a2.accID, a2.username from person p, account a1, account a2, department\_status ds1, department\_status ds2 where p.personID = a1.personID and p.personID = a2.personID and a1.department\_status\_code = ds1.department\_status\_code and a2.department\_status\_code = ds2.department\_status\_code and p.source\_code = 'S' and a1.state = 'valid' and a1.accountType = 'personal' and a2.accID != a1.accID and a2.accountType = 'personal' and ds1.dept\_code = p.dept\_code and ((p.person\_status\_code in ('U', 'N') and a1.status\_code < 'P') or (p.person\_status\_code = 'P' and a1.status\_code = 'P')) and (ds2.dept\_code != p.dept\_code or (p.person\_status\_code in ('U', 'N') and a2.status\_code = 'P') or (p.person\_status\_code = 'P' and a2.status\_code < 'P')) and (a2.state = 'valid' or (a2.state = 'disabled' and (a2.disabledReason in ('Student has left', 'Student is absent', 'User has left - extension expired', 'Student has probably left') or (a2.disabledReason like 'Now using%' and a2.disabledReason != concat('Now using ', a1.username)) or a2.disabledReason like 'Already using%')))
   3. Disable each duplicate account.
      1. Via *disableAccount*() in **disableAccount.pl**
5. Disconnect from AccMan database
   1. Via *disconnectDB()* in **connect.pl**

# DisableAccount.pl

## Introduction

## In Pictures

## Examples

## Depends On

### Perl Packages

* Warnings
* Strict
* DBI

### Libraries

* **connect.pl**
* **generateTransaction.pl**

### Transactions

* 12
* 62

### Database Tables

* MAIL
  + ACCOUNT
  + ACCOUNT\_DETAILS
  + DEPARTMENT\_STATUS
  + EMAIL\_DOMAIN
  + PERSON
  + STUDENT\_ESTS

## Methods

1. *disableAccount()*
   1. Get Account State
      1. select state from account where accID = ?
   2. Stop processing if the account does not exist or the state is one of:
      1. deleted, invalid
   3. Convert account state to disabled, failed, expired, deletable or deleting or NULL
   4. If account state is not NULL
      1. Update the account record disabling the account:
         1. update account set state = ?, disabledReason = ?, disabledCure = ? where accID = ?
   5. Update the account details
      1. update account\_details set disabled = 'Y', disdate = now() where accID = ? and (disabled != 'Y' or disdate is null)
   6. Create an Disabling, type 12, Transaction with parameters:
      1. select username, defaultContext context, disabledReason from account a, department\_status d where a.department\_status\_code = d.department\_status\_code and accID = ?
      2. via *generateTransaction()* in **generateTransaction.pl**
   7. Create an Disable Live@Edu account, type 62, Transaction with parameters:
      1. select concat(a.primary\_mailname, '@', ed.domainName) identity from account a, emailDomain ed where a.primary\_domainID = ed.domainID and accID = ?
      2. via *generateTransaction()* in **generateTransaction.pl**
   8. Close Database handles
      1. Via *finishDB()* in **connect.pl**

# EnableAccount.pl

## Introduction

## In Pictures

## Examples

## Depends On

### Perl Packages

* Warnings
* Strict
* DBI

### Libraries

* **connect.pl**
* **generateTransaction.pl**
* **trLogging.pl** 
  + **Indirectly referenced**

### Database Tables

* MAIL
  + ACCOUNT
  + DEPARTMENT\_STATUS
  + DEPT\_PERSON\_ACCOUNT
  + PERSON
  + STUDENT\_ESTS

### Transactions

* 13
* 33
* 63

## Methods

1. Get Username and information about why the account is currently disabled.
   1. select username, DisabledReason, DisabledCure from account where accID = ?
2. Log information
   1. Via apb\_logactivity\_tofile() in trLogging.pl
3. Get Account State
   1. select state from account where accID = ?
   2. Stop processing if the account state is one of:
      1. deleted, deleting or invalid
4. Convert account state to valid, used, new or NULL
5. Check reason account is disabled.
6. Allow enabling if reason is one of:
   1. Undefined
   2. Blank
   3. Student is absent
   4. Student has left
   5. User has left - extension expired
7. Do **not** allow enabling if the reason is one of:
   1. Now using…
   2. Account moved to…
   3. Anything else
8. If this was called from **enable.pl** override any earlier logic to allow enabling.
9. If the account has been manually blocked then prevent any disabling
   1. select manually\_disabled from account where accID = ?
10. If the account can be enabled
    1. Update the account state
       1. update account set state = ?, disabledReason = null, disabledCure = null where accID = ?
    2. Update the account details
       1. update account\_details set disabled = 'N', disdate = null where accID = ? and (disabled != 'N' or disdate is not null)
    3. Create an ExpiryDate, type 33, Transaction with parameters dependent on the Account state
       1. select username, defaultContext context, 'none' as 'date’ from account a, department\_status d where a.department\_status\_code = d.department\_status\_code and accID = ?
          1. OR
       2. select username, defaultContext context, date\_add(curdate(), interval 1 month) as 'date' from account a, department\_status d where a.department\_status\_code = d.department\_status\_code and accID = ?
       3. Via *generateTransaction()* in generateTransaction.pl
    4. Create a ???????, type 13, Transaction with parameters dependent a optional Reason
       1. select username, defaultContext context from account a, department\_status d where a.department\_status\_code = d.department\_status\_code and accID = ?
          1. OR
       2. select username, defaultContext context, '$reason' as enabledReason from account a, department\_status d where a.department\_status\_code = d.department\_status\_code and accID = ?
11. Via *generateTransaction()* in generateTransaction.pl
    1. Create an Enable Live@Edu account, type 63, Transaction with parameters
       1. select concat(a.primary\_mailname, '@', ed.domainName) identity from account a, emailDomain ed where a.primary\_domainID = ed.domainID and accID = ?
       2. Via *generateTransaction()* in generateTransaction.pl
12. Close Database handles
    1. Via finishDB() in **connect.pl**

# EnableDisable.pl

## Introduction

## In Pictures

## Examples

## Depends On

### Perl Packages

* Warnings
* DBI

### Libraries

* **mailAccount.pl**
* **generateTransaction.pl**
* **trCommon.pl**
  + **implicitly referenced**

### Database Tables

* MAIL
  + ACCOUNT
  + EMAILDOMAIN
  + MAILSERVERS
  + ACCOUNT\_DETAILS
* IDM/IDMDEV
  + MAIL\_DOMAIN
  + MAIL\_ACCOUNT

### Transactions

* 103
* 104

## Methods

1. *prepare\_specific\_SQL()*
2. *specificInitialisation()*
3. *transaction()*
   1. Check username is valid
      1. Via *checkParam\_username*() in **trCommon.pl**
   2. Update the email account
      1. Via *updateEmailAccount()* in **mailAccount.pl**
      2. Select the current email information for the Student
         1. SELECT A.USERNAME, 'ACCMAN' AS 'DATA\_SOURCE', P.DEPT\_CODE, P.TITLE, P.REALNAME, P.FORENAMES, P.PREFERREDNAME, P.INITIALS, P.PERSON\_STATUS\_CODE, IF(AD.DELETED = 'Y', 'DELETED', IF(A.STATE = 'VALID' AND AD.DISABLED = 'N', 'VALID', 'DISABLED')) AS MAIL\_STATUS, IF(A.DISPOSE\_DOMAINID = 266, 'LIVE\@EDU', IF(A.DISPOSE\_DOMAINID = 5, 'LIVE\@EDU', IF(A.DISPOSE\_DOMAINID = 183, 'WEBMAIL', IF(A.DISPOSE\_DOMAINID = 178, 'EXCHANGE', 'POP3')))) AS MAIL\_TYPE, A.DISPOSE\_MAILNAME, ID1.DOMAIN\_ID, IF(A.DISPOSE\_DOMAINID != 178, AD.PASSWORD, NULL) AS MAILBOX\_PASSWORD, ID2.DOMAIN\_ID, IF(A.DISPOSE\_DOMAINID = 183, CONCAT(M.MAILDIR, '/', LEFT(DISPOSE\_MAILNAME, 1), '/', MID(DISPOSE\_MAILNAME, 2, 1), '/', MID(DISPOSE\_MAILNAME, 3, 1), '/', MID(DISPOSE\_MAILNAME, 4, 1), '/', ID(DISPOSE\_MAILNAME, 5, 1), '/', MID(DISPOSE\_MAILNAME, 6, 1)), IF(A.DISPOSE\_DOMAINID != 178, CONCAT(M.MAILDIR, '/', LEFT(DISPOSE\_MAILNAME, 2), '/', MID(DISPOSE\_MAILNAME, 3, 1), '/', DISPOSE\_MAILNAME), NULL)) AS PATH, IF(A.DISPOSE\_DOMAINID != 178, AD.FSQUOTA, NULL) AS QUOTA FROM PERSON P JOIN ACCOUNT A ON P.PERSONID = A.PERSONID JOIN EMAILDOMAIN E ON A.DISPOSE\_DOMAINID = E.DOMAINID JOIN $IDMDB\.MAIL\_DOMAIN ID1 ON E.DOMAINNAME = ID1.DOMAIN\_NAME LEFT JOIN MAILSERVERS M ON A.MAILSERVERID = M.MAILSERVERID LEFT JOIN $IDMDB\.MAIL\_DOMAIN ID2 ON M.SERVERNAME = ID2.DOMAIN\_NAME JOIN ACCOUNT\_DETAILS AD ON A.ACCID = AD.ACCID WHERE AD.TYPE = 'EMAIL' AND A.USERNAME = ?
      3. Update the Students account
         1. UPDATE $IDMDB\.MAIL\_ACCOUNT SET USERNAME = ?, DATA\_SOURCE = ?, DEPT\_CODE = ?, TITLE = ?, SURNAME = ?, FIRST\_NAMES = ?, PREFERRED\_NAME = ?, INITIALS = ?, USER\_STATUS\_CODE = ?, MAIL\_STATUS = ?, MAIL\_TYPE = ?, DELIVERY\_NAME = ?, DELIVERY\_DOMAIN\_ID = ?, MAILBOX\_PASSWORD = ?, MAILBOX\_DOMAIN\_ID = ?, MAILBOX\_LOCATION = ?, MAILBOX\_QUOTA = ? WHERE USERNAME = ?
   3. Check context is valid
      1. Via *checkParam\_context()* in **trCommon.pl**
   4. If disabling, type 12
      1. Create a Disable AD Account, type 103, Transaction
         1. Via *generateTransaction()* in **generateTransaction.pl**
   5. If enabling, type 13
      1. Create a Ensable AD Account, type 104, Transaction
         1. Via *generateTransaction()* in **generateTransaction.pl**

# restoreUFAA.pl

## Introduction

Restore future UFAA status for students after they have left in this session

Restore present/future UFAA status for students after they have left in previous session

Restore UFAA status for students after being erroneously assigned an account for a school/level they are not studying in ( => acad\_sess = 0000)

## In Pictures

## Examples

## Depends On

### Perl Packages

* Warnings
* Strict
* DBI

### Modules

* Connect.pl
* Paths.pl
* trLogging.pl

### Tables

* MAIL
  + PERSON
* BANNER
  + SGBSTDN
  + SYRACCS

## Process

1. Connect to AccMan Database
   1. Via *ConnectAccman()*
      1. Vi *ConnectDB()* in **connect.pl**
2. Connect to Banner Database
   1. Via *ConnectBanner()*
      1. Vi *ConnectDB()* in **connect.pl**
3. Prepare SQL Statements
   1. Via *prepareSQL()*
4. Restore UFAA status in the **next** session for students who left in the **current** session
   1. Via *restoreUFAA()*
   2. Find all students who left in the relevant session
      1. select PIDM, program from person where source\_code = 'S' and acad\_sess = ? and person\_state = 'left'
   3. For each student check if they are still active in banner for target session on a new or different dept.
      1. select SGBSTDN\_DEPT\_CODE, SGBSTDN\_LEVL\_CODE, SGBSTDN\_TERM\_CODE\_EFF from SGBSTDN where SGBSTDN\_TERM\_CODE\_EFF > ? and SGBSTDN\_TERM\_CODE\_ADMIT > ? and SGBSTDN\_STST\_CODE = 'AS' and SGBSTDN\_PROGRAM\_1 != ? and SGBSTDN\_PIDM = ?
   4. If they are re-assign the account
      1. update SYRACCS set SYRACCS\_ACCMAN\_STATUS = 'R', SYRACCS\_ACTIVITY\_DATE = SYSDATE where SYRACCS\_PIDM = ? and SYRACCS\_DEPT\_CODE = ? and SYRACCS\_LEVL\_CODE = ?
5. Restore UFAA status in the **current** session for students who left in the **last** session
   1. Via *restoreUFAA()* see above
6. Restore UFAA status in the **current** session for students in the wrong dept/level
   1. Via *restoreUFAA()* see above
7. Disconnect from Banner Database
   1. Via *disconnectBanner()*
      1. Vi *disconnectDB()* in **connect.pl**
8. Disconnect from AccMan Database
   1. Via *disconnectAccman()*
      1. Vi *disconnectDB()* in **connect.pl**

# StudentAutoState.pl

## Introduction

## In Pictures

## Examples

## Depends On

### Perl Packages

* Warnings
* Strict
* DBI

### Libraries

* **connect.pl**
* **disableAccount.pl**
* **enableAccount.pl**
* **trLogging.pl**
* **paths.pl**

### Database Tables

* MAIL
  + ACCOUNT
  + DEPARTMENT\_STATUS
  + DEPT\_PERSON\_ACCOUNT
  + PERSON
  + STUDENT\_ESTS

## Process

1. Define local variables
   1. Define the current academic session year
      1. Optionally decrement the year by 1.
2. Connect to Banner database
   1. Via *connectDB()* in **connect.pl**
3. Define SQL handles to read persons Account, Person, Department and status information, based on the current academic session.
   1. Via *PrepareSQL()*
4. Find the most recent Person Import **NOT USED**
   1. select importID from person\_import where source\_code = 'S' and completed = 'Y' order by importID desc limit 1
5. Enable accounts for returning students
   1. Via *enable()*
   2. Get all details about accounts potentially to be enabled.
      1. Filter by current academic year and defined account and person states, status codes, dept codes and reasons for disabling the account.
         1. select a.accID, a.username, p.person\_state, p.personID from account a, person p, department\_status ds, dept\_person\_account d where p.personID = a.personID and p.person\_state in ('present', 'graduating') and p.source\_code = 'S' and p.acad\_sess >= $session and ((p.person\_status\_code in ('N', 'U') and a.status\_code < 'P') or (p.person\_status\_code = 'P' and a.status\_code in ('P', 'Q'))) and a.state in ('disabled', 'deletable') and a.accountType = 'personal' and a.department\_status\_code = ds.department\_status\_code and p.dept\_code = d.person\_dept\_code and ds.dept\_code = d.account\_dept\_code and a.disabledReason in ('$leftReason', '$absentReason', '$extendReason', 'Student has probably left')
   3. For each Account
      1. Check if the account is currently valid:
         1. select a.username from account a, person p, department\_status ds, dept\_person\_account d where p.personID = ? and p.personID = a.personID and ((p.person\_status\_code in ('N', 'U') and a.status\_code < 'P') or (p.person\_status\_code = 'P' and a.status\_code in ('P', 'Q'))) and a.state = 'valid' and a.accountType = 'personal' and a.department\_status\_code = ds.department\_status\_code and p.dept\_code = d.person\_dept\_code and ds.dept\_code = d.account\_dept\_code and a.accID != ?
   4. If there is a valid account disable it.
      1. Via *disableAccount()* in **disableAccount.pl**
   5. If the current account is not valid enable it.
      1. Via *enableAccount()* in **enableAccount.pl**
      2. Log the change
         1. Via *apb\_logactivity\_tofile()* in **trLogging.pl**
6. Disable accounts for departed of inactive students
   1. Get Cure codes, %cure
      1. Via *getCurehash*()
         1. select stu\_ests, ests\_desc, person\_state from student\_ests
   2. Get all details about accounts to be disabled.
      1. Filter by current academic year and defined account and person types, states, source codes, dept codes and session.
         1. select a.accID, a.username, p.person\_state, p.stu\_ests from account a, person p, department\_status ds where p.personID = a.personID and p.person\_state in ('absent', 'left') and p.source\_code = 'S' and p.acad\_sess >= $session and a.state = 'valid' and a.accountType = 'personal' and a.department\_status\_code = ds.department\_status\_code
      2. Disable the account
         1. Via *disableAccount()* in **disableAccount.pl**
7. Log counts of accounts processed.
   1. Via logactivity() and apb\_logactivity\_tofile() in **trLogging.pl**
8. Disconnect from Banner
   1. Via *disconnectDB()* in **connect.pl**

# StudentPerson.pl

## Introduction

## In Pictures

## Examples

## Depends On

### Perl Packages

* Warnings
* Strict
* DBI

### Libraries

* **connect.pl**
* **generateTransaction.pl**

### Database Tables

* MAIL
  + TITLEMAP
  + STUDENT\_ESTS
  + BANNER\_DEPT
  + SY\_TO\_EL\_PROGRAM\_CODES
  + PERSON\_IMPORT
  + BANNER\_IMPORT
    - (indirectly as values from this are passed into the functions)
  + PERSON
  + PERSON\_ADD
  + EXTENSION
  + PERSON\_REJECTED\_DATA
  + PERSON\_CHANGE
  + PROG
  + MIDYEARPROG

### Transactions

* 97: UPDATE\_JANUS\_USER\_DETAILS (.NET)

## Process

1. Update Person
   1. As implemented in *Banner4Person.pl*
      1. *prepareStudentPersonSQL()*
      2. *initialiseParsing()*
      3. *generateImportID()*
      4. For each Student record in MAIL.BANNER\_IMPORT
         1. *parsePerson()*
         2. *parseCourse()*
         3. *parseRegistration()*
         4. *updateStudentPerson()*
      5. *completeImport()*
      6. *finishStudentPersonSQL()*

## Methods

1. Initialization
   1. *@graduationDate*
   2. %psc
   3. %pgr
2. *prepareStudentPersonSQL()*
   1. Defines SQL Statements used in the following methods.
3. *initialiseParsing()*
   1. Initialises the AccMan person reference data, by calls to:
      1. *getTitleHash*()
      2. *getPStateHash()*
      3. *getDeptHash()*
      4. *getSYdeptHash()*
      5. *finishDB*() in **connect.pl**
4. *getTitleHash*()
   1. initialises map of person titles, *$titleMap*, using:
      1. select title, source\_title from titleMap
5. *getPStateHash()*
   1. initialises a map of person states, %pstate, using:
      1. select stu\_ests, person\_state from student\_ests
6. *getDeptHash()*
   1. initialises a map of Banner Depts codes, $depthash, to AccMan codes using:
      1. select SGBSTDN\_DEPT\_CODE, dept\_code from banner\_dept
7. *getSYdeptHash()*
   1. initialises a map of obsolete SY Banner chemistry codes, %SYdepthash, using:
      1. select program from SY\_to\_EL\_program\_codes
8. *generateImportID()*
   1. Creates a new record in the MAIL.PERSON\_IMPORT table and returns the ID value of the new record, using:
      1. insert into person\_import (source\_code) values ('S')
9. *completeImport()*
   1. Signals the import is complete by updates the MAIL.PERSON\_IMPORT record, using:
      1. update person\_import set completed = 'Y' where importID = ?
10. *parsePerson()*
    1. Check and convert Personal Details from Banner to AccMan equivalents.
    2. Rejects the student if:
       1. Student ID, SPRIDEN\_ID, is invalid
       2. Student last name SPRIDEN\_LAST\_NAME is invalid.
       3. Student first name SPRIDEN\_FIRST\_NAME is invalid.
       4. Student middle name SPRIDEN\_MI is invalid.
       5. Student title, SPBPERS\_NAME\_PREFIX, cannot be mapped to AccMan
       6. Date of birth, SPBPERS\_BIRTH\_DATE, is invalid.
    3. Record any rejections:
       1. insert into person\_rejected\_data (importID, rollnumber, field\_name, bad\_field\_value) values (?, ?, ?, ?)
    4. Sets the following fields:
       1. $*pidm* set from $*SPRIDEN*\_*PIDM*
       2. $*rollnumber* set from $*SPRIDEN*\_*ID*
          1. Check the Student ID is of the form “@########” and extract the number portion
       3. $*realname* set from $*SPRIDEN*\_*LAST*\_*NAME*
          1. Replace “.” With space, remove excess whitespace, replace accented characters
       4. $*firstName* set from $*SPRIDEN*\_*FIRST*\_*NAME*
          1. Replace “.” With space, remove excess whitespace, replace accented characters
       5. $*middleName* set from $*SPRIDEN*\_*MI*
          1. Replace “.” With space, remove excess whitespace, replace accented characters
       6. $*preferredname* set from $*SPRIDEN*\_*FIRST*\_*NAME*
       7. $*forenames* set from $*firstname* $*middleName*
       8. $*initials* set from $*forenames*
          1. Remove apostrophes, uppercase and initilise
       9. $b\_date set from $*SPBPERS\_BIRTH\_DATE*
       10. $title set from $*SPBPERS\_NAME\_PREFIX*
           1. Remove dots and find AccMan value from *$titleMap*
11. *parseCourse()*
    1. Check and convert Course Details from Banner to AccMan equivalents.
    2. Rejects the student if:
       1. student status, SGBSTDN\_STST\_CODE, is not one of:
          1. AS|IS|DE|IG|RR|SU|WD|WU|IN|WN
       2. Student level, SGBSTDN\_LEVL\_CODE, is not one of:
          1. UG, PG, FE, PQ, CE
       3. Student term code, SGBSTDN\_TERM\_CODE\_EFF, is < 2000 or > 2020.
       4. Student attendance code, SGBSTDN\_PROGRAM\_1, is not
          1. [FSPEDLO]|LI
       5. Department code, $SGBSTDN\_DEPT\_CODE, is not included in $depthash
       6. Department code, $SGBSTDN\_ STYP\_CODE, is not one of:
          1. FNCDET
       7. Year Code, SGBSTDN\_BLCK\_CODE, is not 1-9.
    3. Record any rejections:
       1. insert into person\_rejected\_data (importID, rollnumber, field\_name, bad\_field\_value) values (?, ?, ?, ?)
    4. Sets the following fields:
       1. $stu\_levl set from $SGBSTDN\_LEVL\_CODE;
       2. $person\_status\_code set from $SGBSTDN\_LEVL\_CODE;
       3. $stu\_term set from $SGBSTDN\_TERM\_CODE\_EFF;
       4. $acad\_sess set from substr ($SGBSTDN\_TERM\_CODE\_EFF, 0, 4);
       5. $program set from $SGBSTDN\_PROGRAM\_1 || 'X/X/X';
       6. $stu\_att set from $SGBSTDN\_PROGRAM\_1 or F
       7. $sos\_code set from $SGBSTDN\_MAJR\_CODE\_1
       8. $dept\_code set from $SGBSTDN\_DEPT\_CODE;
       9. $stu\_styp set from $SGBSTDN\_STYP\_CODE;
       10. $stu\_degc set from $SGBSTDN\_DEGC\_CODE\_1;
       11. $qual\_aim\_code set from *$pgr*
       12. $stu\_block set from $SGBSTDN\_BLCK\_CODE;
       13. $yr\_course set from $SGBSTDN\_BLCK\_CODE
       14. $yr\_student set from *$stu\_styp*
       15. $leaver\_flag set from '';
12. *parseRegistration()*
    1. Check and convert Registration Details to AccMan
    2. Rejects Student if:
       1. Registration status is not one of:
          1. Y, C, N
       2. Enrolment Status, SFBETRM\_ESTS\_CODE, is not in $*pstate*
       3. Start Date, SFBETRM\_ADD\_DATE, is invalid
       4. End date, SKBHINS\_ENDDATE, is invalid
    3. Record any rejections:
       1. insert into person\_rejected\_data (importID, rollnumber, field\_name, bad\_field\_value) values (?, ?, ?, ?)
    4. Sets the following fields:
       1. $reg\_status set from $SFBETRM\_AR\_IND;
       2. $stu\_ests set from $SFBETRM\_ESTS\_CODE;
       3. $person\_state set from $pstate/$SFBETRM\_ESTS\_CODE/$SKBHINS\_ENDDATE
       4. $start\_date set from $SFBETRM\_ADD\_DATE;
       5. $end\_date set from $SKBHINS\_ENDDATE;
13. *updateStudentPerson ()*
    1. Check to see if there is an existing Person record by using the StudentID/RollNumber
       1. select personID, Initials, RealName, sos\_code, person\_status\_code, dept\_code, forenames, qual\_aim\_code, person\_state, importID, yr\_student, yr\_course, acad\_sess, reg\_status, leaver\_flag, pidm, title, program, stu\_term, stu\_levl, stu\_styp, stu\_degc, stu\_block, stu\_stst, stu\_ests, stu\_att, start\_date, end\_date, preferredName, b\_date from person where rollnumber = ? and person\_status\_code in ('N', 'U', 'P', 'X')}
    2. If the Person record **does not** exist:
       1. Calculate the final year status
          1. Via *set\_Leaver()*
       2. Insert a new Person record
          1. insert into person (Initials, RealName, source\_code, rollnumber, SPRIDEN\_ID, sos\_code, person\_status\_code, dept\_code, forenames, qual\_aim\_code, person\_state, importID, first\_importID, yr\_student, yr\_course, acad\_sess, reg\_status, leaver\_flag, pidm, title, program, stu\_term, stu\_levl, stu\_styp, stu\_degc, stu\_block, stu\_stst, stu\_ests, stu\_att, start\_date, end\_date, preferredName, b\_date) values (?, ?, 'S', ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
       3. Log the addition of a Person
          1. insert into person\_add (importID, rollnumber) values (?, ?)
    3. If the Person record does exist check the data to see if an update is required:
       1. Optionally force an update regardless of data mismatches.
       2. don't import 'expected' for following year if 'present' for db year on same programme and yr\_course > db\_yr\_course
       3. Do not update if the new data is for an older academic year.
       4. Do not update if new data is older than the current data or worse(!?!)
       5. Do not update if the Term Code is **not** the same in the new and old data, i.e. they have changed course (????) and
          1. They were present in the lst course but not present on the new one.
             1. Or
          2. The existing aimed for qualification code is less than the new one.
    4. Potentially adjust fields:
       1. *$person\_state*
          1. Potentially depends on “extended” state of student
             1. select personID from extension where personID = ?
       2. $*leaver\_flag*
       3. $*preferredName*
    5. Log changes.
       1. Via *logchange()*
    6. Update Person record:
       1. update person set importID = ?, source\_code = 'S', Initials = ?, RealName = ?, SPRIDEN\_ID = ?, sos\_code = ?, person\_status\_code = ?, dept\_code = ?, forenames = ?, qual\_aim\_code = ?, person\_state = ?, yr\_student = ?, yr\_course = ?, acad\_sess = ?, reg\_status = ?, leaver\_flag = ?, pidm = ?, title = ?, program = ?, stu\_term = ?, stu\_levl = ?, stu\_styp = ?, stu\_degc = ?, stu\_block = ?, stu\_stst = ?, stu\_ests = ?, stu\_att = ?, start\_date = ?, end\_date = ?, preferredName = ?, b\_date = ? where personID = ?
    7. Add a new Transaction, type 97, to update Janus
       1. Via *generateTransaction()* in **generateTransaction.pl**
14. *finishStudentPersonSQL* *()*
    1. Closes SQL handles used in the methods described above.
15. *Set\_leaver()*
    1. Read the relevant Leaver flag for the student
       1. select p.SPLENGTH, p.UOMS, m.PROGRAM from prog p left join midYearProg m on m.PROGRAM = p.PROGRAM where p.PROGRAM = ?
16. *Logchange()*
    1. insert into person\_change (importID, personID, field\_name, old\_field\_value, new\_field\_value) values (?, ?, ?, ?, ?)

# Other Scripts

The following scripts were not initially identified as being crucially important to this analysis but since they do contain references to department names & codes, they are included here

# maillist-staff.pl

This is probably not used any more since the last modification was made in 2010. It creates a file containing user details that can be imported into Exchange. We are now using AD/Exchange integration so can’t see this being used. The file it generates includes department details.

Transactions generated: None

# maillist-student.pl

Performs the same task as maillist-staff, but for students.

Transactions generated: None

# mailstaffchanges.pl

Transactions generated: None

## Process

Compiles a list of staff whose details have changed in the last 24 hours and emails the changes to various people.

Certain processes (sap\_import.pl, studentPerson.pl etc) insert records into the **person\_change** table to record details that have changed.

This script looks for changes to the field “dept\_code” field and records their old dept\_code, and the dept\_code of the department they have moved to.

select a.username, c.old\_field\_value as old, d1.dept\_name as old\_name, c.new\_field\_value as new, d2.dept\_name as new\_name, date\_format(i.import\_time,'%Y-%m-%d') as 'when'

from

account a

join person\_change c on a.personID = c.personID

join person\_import i on c.importID = i.importID,

dept d1,

dept d2

where

i.importID > 12653

and i.import\_time >= @timescale

and i.source\_code = 'P'

and c.field\_name = 'dept\_code'

and a.state = 'valid'

and c.old\_field\_value = d1.dept\_code

and c.new\_field\_value = d2.dept\_code

order by a.username, i.importID

Also reads from the reconciliation table to find people whose username has changed. The reconcile.pl script inserts records into the reconciliation table. The dept\_code field is used within the query.

select a.username, p1.dept\_code as old, d1.dept\_name as old\_name, p2.dept\_code as new, d2.dept\_name as new\_name, date\_format(r.reconciliation\_timestamp, '%Y-%m-%d') as 'when'

from

account a,

person p1,

person p2,

reconciliation r,

dept d1,

dept d2

where

a.personID = p2.personID

and p1.rollnumber = r.old\_rollnumber

and p2.rollnumber = r.rollnumber

and p1.dept\_code != p2.dept\_code

and a.state = 'valid'

and a.status\_code = 'S'

and p1.dept\_code = d1.dept\_code

and p2.dept\_code = d2.dept\_code

and r.reconciliationID > 3300

and r.reconciliation\_timestamp >= @timescale

order by a.username, r.reconciliationID

# moverename.pl

This script is no longer needed. It was switched off during transformation because it does not support AD.

# noshow.pl

Looks for students that Accman expected to be here, but who have not turned up. For any students it finds, it re-reads the student details from Banner and updates their details in Accman. This includes department information.

Transactions generated: none

# pregen\_associate.pl

Transactions generated

82 (Complete Associate Generation)

This script relies heavily on the department details. When a new associate is created, they will write a transaction to the DB with parameters specifying their department code. This script then finds the next available username, from a pool of pre-created accounts, to grant the new user. It then performs an initial setup of the account in NDS. Example:

Transaction specifies that a user with the following details needs to be created:

account\_prefix: AAA i.e. what their username should start with

department\_code: AD i.e. their two-letter department code

rollnumber: 400124566 (the user’s rollnumber)

status\_code: S (staff)

This script then reads that transaction and finds the next available matching username (based on the prefix, department, and numeric part of the username. E.g. ‘find the next available username for department code UI, with a prefix UIS’ might come up with 1058 if there have been 1057 members of staff in UIS previously, so the new user will be assigned a username of UIS1058.

The account is created in an NDS context that is built up using information retrieved from the department\_status table (the fields defaultContext, defaultServer and defaultVolume). An example row from the department\_status table would be:

SELECT

ds.department\_status\_code, ds.active, ds.defaultContext, ds.defaultServer, ds.defaultvolume

FROM department\_status ds

LEFT OUTER JOIN dept\_person\_account dpa ON dpa.account\_dept\_code = ds.dept\_code

WHERE ds.status\_code = ?

AND (dpa.person\_dept\_code = ? OR ds.dept\_code = ?)

order by department\_status\_id desc;

The parameters to this query are the status code (‘S’ for staff) and the department code (params 2 *and* 3), all of which come from the transaction parameters.

Once the context has been derived from this query, the user is created in NDS and is forever linked to this department-specific context.

NB There is another variant of this script called pregen.pl which is used for non-associates. Pregen.pl is executed independently of user-creation, and is used to create pools of available users that can be used later (still based on departments). Pregen-associate.pl does the same thing, but forms a step in the ‘create associate’ process itself, and instead of creating a pool of e.g. 10 accounts, creates a single account that is snapped up there and then by the user being set up.

# selfReg.pl

Transactions Generated: 43 (Register),

This script is used to manually generate a type 43 (register) transaction for a user. The operator enters the username and the script presents them with some information about the user. It then asks them if the details are correct, and if so, creates the type 43 transaction.

One of the values displayed to the operator is the user’s department. That seems to be all this script does with the department.

# staffmultileft.pl

Transactions generated: 41 (Reconcile)

## Process

This is one of the scripts executed by sap\_import.bat. It uses the dept codes in a fairly niche but straightforward way.

1. Looks for all people who are multi-post holders but their person record states that they have ‘left’

select distinct p.personID, a.accID, a.username from person p, account a, person\_multi\_post m where p.personID = a.personID and p.personID = m.personID\_1 and p.multi\_post = 'Y' and a.state = 'valid' and a.accountType = 'personal' and p.person\_state in ('left', 'absent', 'extended') and p.source\_code = 'P'

1. Count the number of records each person has in the person\_multi\_post table (to work out how many additional posts they have)

select count(m.personID\_2) from person\_multi\_post m, person p where p.personID = m.personID\_2 and p.person\_state = 'present' and m.personID\_1 = ?

1. If there is only one other account, reconcile it i.e. check what state the second account is in, and if it’s the only account left, make the account record point to this person record (since the previous person record is redundant because it is ‘left’)
2. If there are multiple other person records, do a more complicated version of reconcile. This one takes the department code into account i.e.
   1. if two person records belong to the same department, try and reconcile them
   2. otherwise try to reconcile them another way (i.e. disable the extra account)

# staffsequential.pl

Transactions generated: 41 (Reconcile)

This is one of the scripts executed by sap\_import.bat. It’s almost identical to staffmultileft.pl apart from the fact that it acts on people who do not have multiple posts. It also uses the department code in a fairly minor way.

# Register.pl

## Introduction

## Applies account settings from student account assignment and staff self-registration.

## Links an account record to its person record & many other things such as setting up email, setting quotas etc

Contains a lot of NDS functionality that will be (has been) phased out after transformation

Takes 4 transaction parameters: accID, personID, username and disabledReason (optional)

Since it performs many different tasks, and it quite often breaks half way through doing these things, the register process keeps track of where it is using a fifth transaction parameter called trState. Each time a step is completed, this number is incremented up to the current final state of 24. Until this state is reached, the tr\_state of the transaction remains at ‘pending’, and on reaching state 24, it is changed to ‘processed’.

This allows the register script to ‘recover’ any half-processed transactions that would otherwise have failed and been left in a state of limbo. It looks for any ‘pending’ transactions, checks what state they reached before the script encountered a problem, and then continues from where it left off.

## Process

State 0: links the account and person records together by setting the personID column in the account table

State 1: Get the default email system by calling getDefaultDisposeDomain().

State 2: Generate a random email password and insert it into the account\_details table (for the EMAIL system)

State 3: Create a mail account by calling createMailAccount() from mailAccount.pl. This inserts a record into mail\_account table in the IDM database.

State 4: Generates the email address which takes the form of the initials followed by the surname (with some checking and formatting). Works out their domain (e.g. salford.ac.uk or live.salford.ac.uk) based on whether they are staff or student. Adds the primary and alias addresses to the account.

State 5: Generates a random password for Athens and inserts it into the account\_details table

State 6: Insert a new transaction to create the Athens account (type 7)

State 7: Insert a new transaction to create the password.txt file in the user’s F: drive (type 39)

State 8: Set the property ‘NetAPPIMAPhomeDirectory’ on the users NDS account (students only)

State 9: Set the email quota by updating the account\_details table (where type = ‘EMAIL’)

State 10: Set the GPAS Balance (obsolete) for printing

State 11: Set the NDS file quota by updating the account\_details table (where type = @NDS’)

State 12: Remove the account expiry date from NDS

State 13: Sets some more NDS attributes (username, rollnumber, name).

State 14: Sets the ACLs in NDS so that the user cannot change the rollnumber and lastname attributes.

State 15: Create the import and control files for the NDS account, and then run JRBImprt to set all the relevant values for this user.

State 16: Update the account so that it is valid.

State 17: Disable the account if there is a disabled reason associated with the account

State 18: Update the mail account (calls updateMailAccount()) to update table mail\_account in the IDM database.

State 19: A type 42 (set email quota) transaction used to be generated here, but it doesn’t happen any more.

State 20: Insert a transaction to create the Live@Edu account for students (type 48).

State 21: Update the account table with some email-related values

State 22: Insert a transaction to create a new Janus account (type 97)

State 23: Insert a transaction to create the AD account (type 93)

State 24: Finished!

# .NET Processes

The following .NET processors in AccmanBackEndLibrary make use of the department. The classes (processors) in AccmanBackEndLibrary are roughly equivalent to the older Perl scripts.

# TempusCohortCreateUpdateUser

This has its own list of departments which reside in a different database. A new table in Accman called TempusCohortProgrammes contains references to these departments.

# ActivateDirectoryNewUser

This makes use of the department code stored in HR\_DEPARTMENTS and if a matching one cannot be found, it will fall back to using the department in the dept table. The department name is pushed into ActiveDirectory.

# ImportAssociate

Reads the department code from the Associates database (in the Associate.DepartmentCode column) and inserts it into person.dept\_code column in the Mail database.