

## CARPROC FUNCTION

Melinda Varian

Office of Information Technology  
Princeton University  
87 Prospect Avenue  
Princeton, NJ 08544 USA

—.—  
Email: [melinda@princeton.edu](mailto:melinda@princeton.edu)  
Web: <http://pucc.Princeton.EDU/~melinda/>  
Telephone: 1-609-258-6016

February 22, 2005

### I. INTRODUCTION

This document describes the function implemented in CARPROC, the suite of programs used to extract information from various University databases to drive Whale to populate account information for OIT's Active Directory, LDAP, alumni LDAP, IMAP, Webmail, Exchange, unix, P-synch, Kerberos, and mainframe systems. This document does not discuss the Whale function, which is described in another document.

This document does not attempt to provide a definitive description of CARPROC's error handling. Suffice it to say that it is copious; whenever a comparison between files is made, for example, and a match is not found when it should be, an error file is written to be examined by the programmer. Nor does this document describe CARPROC's procedures for storing multiple generations of files to facilitate backout and restart in case of failure. It is assumed that a replacement program would implement both error detection and checkpoint/backout in a robust way but in too different a manner to make it worth documenting these details of CARPROC.

### II. THE MAIN META-DATA FILES

#### The \$File

The "\$File" is Whale's database in which it stores the information used to populate directory entries and computer account information in LDAP, NIS, Active Directory, Kerberos, *etc.* The \$File currently contains some account information, such as disk quota billing, which is stored nowhere else. A more complete description can be found in the Whale document. In CARPROC's processing, \$File data are usually handled in a flattened form with filenames having an extension in the form "**dictoutX**".

There is a second \$File, the "**old \$File**", which contains data as of July 18, 2000. The use of this second file dates from the era when all computer accounts had to be renewed once a year, which necessitated the use of two concurrent files, for two fiscal years. The old \$File is still processed by CARPROC (and Whale) for historical reasons.

### **recent.userids**

The recent.userids file is a list of all netids assigned during the past few decades, along with the last known information about the holder of each netid. Its primary use is in preventing the accidental reassignment of a netid.

### **campcomm.file**

The campcomm.file is a flattened subset of the Campus Community database. CARPROC uses it for understanding when a computer user becomes active or inactive, changes status or department, and so forth. campcomm.file is basically a file of University people and their affiliations and status within the University. There can be more than one entry for a given person, if that person has multiple affiliations with the University. For example, one person might have entries as a special student, as an employee, as a museum docent, and as an alumnus. There is only one entry per person per affiliation and that shows the current status of the affiliation. Only “recent” alumni/ae are included. CARPROC does not add a record to campcomm.file for a future-dated person until he has been assigned a netid. Records for people who have become inactive do not generally include the netid, so campcomm.file is not definitive on that field.

In CARPROC’s processing, several files are used that are “augmented derivatives” of campcomm.file. That is, the derivative files contain additional records for computer users who are not in Campus Community. These include both non-human entities and humans whose affiliation with the University is not one noted in Campus Community. Some of these augmented derivatives are **tesspers.file**, **uaid.file**, and **person.file**.

### **uaidadd files**

The files tesspers.uaidadd and uaidplus.uaidadd list the computer users not found in Campus Community. They are used to augment the files derived from Campus Community.

### **Other files**

- : There are a good many **manually-maintained files** containing information not found in Campus
- : Community but still required by CARPROC.

## **III. NIGHTLY PROCESSING (proctxtg.exec)**

The nightly processing is scheduled to start at 21:15, by which time Campus Community has built its file of the day’s transactions. The primary purposes of the overnight process are to update the campcomm.file, to add netids and accounts for new people, and to pick up status changes. The basic steps (described in greater detail below) are:

1. The Campus Community transaction file, address file, and phone file are retrieved using FTP.
2. The LDAP directory is retrieved in flattened form (an LDIF dump), which is parsed to create several working files.
3. The \$File is retrieved in flattened form (a “dictout” file).

4. Newly-added \$File records (in the dictout file) are updated from a flattened copy of the Student Records database and from a campcomm.file derivative to complete or correct data entered by a human operator of Whale.
5. The recent.userids file is updated from the dictout file (to pick up Whale changes made during the day).
6. Newly-added \$File records (in the dictout file) are updated from the telephone directory to complete or correct data entered by a human operator of Whale.
7. Newly-added \$File records (in the dictout file) are updated to make addresses mixed case.
8. Newly-added \$File records (in the dictout file) are updated to put the telephone numbers into a canonical form.
9. The uaidadd files are updated with any new netids added by Whale operators for entities not found in Campus Community.
10. The recent.userids file is updated directly from the \$File to pick up new “universityid reference” fields added during the day.
11. Computer accounts and directory entries are removed for any users whose “netid termination date” has been reached.
12. The ldaptxt.file (the Campus Community transaction file), the ldaptxt.delayed file (which contains Campus Community transactions CARPROC has held over from a previous run), and the ldaptxt.pending file (which contains Campus Community transactions that had not reached their effective date in an earlier run) are processed to select transactions that can be posted to campcomm.file today, depending on the effective date of the transaction.
13. Whale commands are issued to apply changes to names, department numbers, and Social Security numbers found in the Campus Community transactions.
14. Netids are selected for new people in the campcomm.file who are near enough to beginning their University affiliation. A starter password is also assigned.
15. LDAP entries and computer accounts are created and “sealed envelopes” containing the starter password are printed.
16. Newly-assigned email addresses are FTP’d to dmsdrop to be read into Campus Community.

The nightly processing is driven by the script proctxtg.exec, which invokes a number of other scripts:

### **ldaptxtg.exec**

Three files created by Campus Community are FTP’d from dmsdrop:

1. The file /dmsdrop/campcomd/prod\_files/ldap.txt (which lists the day’s Campus Community transactions) is processed by the script ldaptxt4.exec to create ldaptxt.file, which has all accented characters changed to the unaccented equivalent, and ldapacc.file, which contains

just the records that have accented characters (in unicode). The translation currently used is shown here (with the hexadecimal representation of the ASCII character translated):

- BF S
- C1 A
- C7 C
- D6 O
- E1 a
- E3 a
- E4 a
- E7 c
- E8 e
- E9 e
- EB e
- ED i
- EF i
- F1 n
- F3 o
- F6 o
- F8 o
- FA u
- FC u

- : 2. The file /dmsdrop/campcomd/prod\_files/ldap\_addr.txt (which lists all addresses known to Campus Community) is stored as address.file.
- : 3. The file /dmsdrop/campcomd/prod\_files/ldap\_phone.txt (which lists all phone numbers known to Campus Community) is stored as phone.file.

#### **ldifget.exec**

: /var/local/ns-home/slapd411/slapd-dumper/ldif/dumper.ldif, the LDIF dump of the main LDAP directory, is FTP'd from dumper.Princeton.EDU as ldif.dump.

#### **ldiupdad.exec**

The records in ldif.dump are processed:

- : 1. Records split during the dumping process are rejoined (continuation records begin with a single blank).
- : 2. Attribute names are translated to lower case.
- : 3. Encoded values are decoded. (Encoded values are denoted by the use of “:”, rather than “:”, in the dump. The attribute values are converted from Base64 encoding. The record is rebuilt with the attribute name, a single colon, and the decoded attribute value with trailing blanks removed.)
- : 4. The records for each entity in ldif.dump are joined to form a single record.

5. If an entry has a universityid attribute, the universityid is prefixed to the record containing all attributes. If an entry has no universityid attribute, it is checked to be sure it has a dn attribute. If not, it is written to an error file. If the entry has a dn but has no universityid, it may be necessary to assign a dummy universityid. The file ldiupdad.uidall is searched, matching on the distinguished name:
- If a match is found, the previously assigned dummy universityid is prefixed to the record.
  - If no match is found but the dn specifies a list (“ou=Lists”), the script ldiupda2.rexx is invoked to assign a dummy universityid suitable for a list (in the range 098210000 through 098299999), which is prefixed to the record.
  - If no match is found and the dn does not specify a list, the entry must be one built and maintained by the LDAP server; the script ldiupda0.rexx is invoked to assign a dummy universityid (in the range beginning 040000000), which is prefixed to the record.

(ldiupdad.uidall is an accumulation of the records generated in steps b. and c. above.)

- Entries with universityids starting “0400” are not processed further. Entries with universityids starting with “09919” are also ignored. (These are entries created by Whale as test data.)
- The entries are sorted into universityid order. Any cases of duplicate universityids (there should be none) are written to an error file.
- The full records are reformatted so that there is one record per attribute containing just the universityid, the attribute name, and the attribute value. These records are written to the file ldiupdad.file.
- The records are then reformatted so that there is one record per attribute containing just the universityid and the attribute value for only the attributes of interest. These records are processed as follows:
  - uid: The records are written to the file uid.file.
  - dn: The distinguished names are written to one of four files, depending on the entry type:
    - ldiupdad.xaliases (dn contains “ou=Xaliases”);
    - ldiupdad.lists (dn contains “ou=Lists”);
    - ldiupdad.limbo (dn contains “ou=Limbo”);
    - ldiupdad.accounts (all others).
  - campusid: The records are written to the file campid1.file.
  - emailbox: If there are multiple entries for a given universityid, the records are written to an error file; otherwise, they are written to mailbox.file.
  - emailboxalternate: All records for a given universityid are joined into a single blank-delimited record and written to the file emailboa.file. (Note that the resulting record will always have the individual items in the same order as in the LDIF dump.)
  - emailrewrite: If there are multiple entries for a given universityid, the records are written to an error file; otherwise, they are written to emailrew.file.

- facsimiletelephonenumber: If there are multiple entries for a given universityid or if the telephone number is badly formatted (it doesn't have hyphens in positions 4 and 8), the records are written to error files; otherwise, they are written to fax.file.
- mail: If there are multiple entries for a given universityid, the records are written to an error file; otherwise, they are written to email.file.
- ou: The records are written to the file ou.file.
- : • pts42: The records are written to the file pts42.file.
- pucoordinator: All records for a given universityid are joined into a single blank-delimited record and written to the file pucoord.file.
- : • pudegrees: If there are multiple entries for a given universityid, the records are written to  
: an error file; otherwise, they are written to pudegree.file.
- pugoingaway: The records are written to the file pugoaway.file.
- puhomedepartmentnumber: The records are written to the file homeddept.file.
- puinternalinfo: The records are written to the file puintern.file.
- : • pointerofficeaddress: If there are multiple entries for a given universityid, they are written  
: to an error file; otherwise, they are written to the file pintoff.file.
- pumailquotagrace: The records are written to the file pmqgrace.file.
- puofficelocation2: If there are multiple entries for a given universityid, the records are  
written to an error file; otherwise, they are written to puofloc2.file.
- puofficetelephone2: If there are multiple entries for a given universityid or if the telephone  
number is badly formatted, the records are written to error files; otherwise, they are written  
to puoftel2.file.
- pupsynch: The records are written to the file pupsynch.file.
- : • purescollege: If there are multiple entries for a given universityid, they are written to an  
: error file; otherwise, they are written to the file purescol.file.
- pustatus: If there are multiple entries for a given universityid, the records are written to an  
error file; otherwise, they are written to pustatus.file.
- puunitnumber: If there are multiple entries for a given universityid, the records are written  
to an error file; otherwise, they are written to unitnumb.file.
- street: If there are multiple entries for a given universityid, the records are written to an  
error file; otherwise, they are written to puofloc1.file.
- studentstreet: If there are multiple entries for a given universityid, the records are written to  
an error file; otherwise, they are written to pustuloc.file.

- studenttelephonenumber: If there are multiple entries for a given universityid or if the telephone number is badly formatted, the records are written to error files; otherwise, they are written to pustutel.file.
  - telephonenumber: If there are multiple entries for a given universityid or if the telephone number is badly formatted, the records are written to error files; otherwise, they are written to puoftel1.file.
  - : • title: If there are multiple entries for a given universityid, the records are written to an error file; otherwise, they are written to the file titles.file.
  - : • voicemailbox: If there are multiple entries for a given universityid, the records are written to an error file; otherwise, they are written to voicebox.file.
- 10.The uid.file and the campid1.file are matched for consistency. Records from uid.file for which there is no corresponding record in campid1.file are written to an error file. (There should be no such records, as every user's uid should also be listed among his campusids.) Records from campid1.file for which there is no corresponding record in uid.file are written to campid2.file.
- 11.The uid.file and the campid2.file are merged to build records containing a universityid followed by a blank-delimited list of all campusids for that universityid. These records are written to the file campusid.file.
- 12.The file uid.file is examined for duplicate uids, and the duplicate uids are written to an error file. (There should be no such records.)
- 13.The uid.file and the campid2.file are merged and sorted on the uid/campusid to find any campusids that appear in multiple LDAP entries. The duplicate records are written to an error file. (There should be no such records.)
- 14.The email.file is examined to find any mail attributes that appear in multiple LDAP entries. The duplicate records are written to an error file. (There should be no such records.)

### **allupdat.exec**

The files campusid.file, email.file, emailboa.file, mailbox.file, emailrew.file, fax.file, homedepartment.file, unitnumb.file, and voicebox.file are merged (by universityid) into the existing files with the same name and the extension .filecurr, *e.g.*, campusid.filecurr. The merge process adds new entries and updates entries for which the attribute value has been changed in LDAP. No entries are discarded from the .filecurr files, so they retain the history of people who no longer have LDAP entries.

### **updatmov.exec**

- The overnight processing is aborted at this point unless the following files are determined to have been newly created:
- : campusid.filecurr, email.file, email.filecurr, emailboa.file, emailboa.filecurr,
  - : mailbox.file, mailbox.filecurr, emailrew.file, emailrew.filecurr, fax.filecurr, homedepartment.file,
  - : homedepartment.filecurr, ldiupdad.accounts, ldiupdad.file, ldiupdad.lists, ldiupdad.xaliases, ou.file,
  - : pmqgrace.file, pts42.file, pucoord.file, pudegree.file, pugoingaway.file, puintern.file,

: puofloc1.file, pintoff.file, puofloc1.file, puofloc2.file, puoftel1.file, puoftel2.file, prescol.file,  
: pupsynch.file, pustatus.file, pustuloc.file, pustutel.file, titles.file, uid.file, unitnumb.file,  
: unitnumb.filecurr, voicebox.file, and voicebox.filecurr.

**dictcopy.exec**

An MVS job is run to produce allcurr.dictouta, a flattened \$File. The records in this (and other "dictionary" files having a "dictoutX" extension) are in the following format:

```
1- 8 Netid
10  "1"
12- 43 User's name
44  "A" for alias
45- 76 Street address
77  "C" for campusid conflict (deprecated)
78  "O" for old file
79  "O" for obsolete
80  "I" for inactive
-----
81- 88 Netid
90  "2"
92-123 City, state, zipcode
125-133 Universityid
135-136 "CM" for campus mail; "US" for non-campus mail
138-140 Status
142  "Y" address update flag (deprecated)
144-150 Unix uidnumber
152-160 SSN
-----
161-168 Netid
170  "3"
172-203 Remainder of address
205-216 Telephone number
218-225 Plaintext password (deprecated)
227  Unix system code (deprecated)
228-240 Unix crypted password (deprecated)
--- Following only in allcurr.dictouta and alllast.dictoutd ---
241-248 Netid
250  "4"
252-260 Universityid reference
262-263 Reference type: "DI", "NR", or "AE"
265-272 Netid termination date (YYYYMMDD)
274-276 Department number
278-281 Birthday (MMDD)
282-320 Unused
```

The same information is extracted from the old \$File and is written to the file alllast.dictoutd.

### **dicupdat.exec**

The script `dicupdat.exec` updates the current and old \$File dictionaries with SSN, `universityid`, and campus address data from `studtr.file` (a flat file from the legacy Student Records system) and from `person.file` and `uaid.file` (both created from `tesspers.file` by `tesupdat.exec`). The purpose here is to correct hand-added information where possible; this seldom happens nowadays because of Whale's ability to look information for new users up in LDAP. If differences are found, Whale alter commands are generated to update the \$File (and LDAP, AD, NIS, *etc.*).

`dicupdat.exec` generates the corrected `allcurr.dictoutb` and `alllast.dictoute` and merges them to produce `allcurr.dictout`. It also generates a list of `netids` that are to be terminated because they have reached their termination date.

The steps involved in `dicupdat`'s processing:

- The `studtr.file` is read and any records with blank `universityid` fields are written to an error file. The remaining records are passed to the next step.
- The passed records are merged with records from the `person.file`. The steps of the merge process:
  1. If a record with the same `universityid` and SSN occurs in both files, the one from `studtr.file` is ignored.
  2. If two of the remaining records have the same `universityid` but different SSNs, they are written to an error file. If the SSN in the record from `studtr.file` is a dummy (one that starts with "999"), the record from the `person.file` is retained.
  3. All remaining records are written to `dicupdat.work1` and passed to the next step.
- The passed records are merged with the `uaid.file`, which contains a list of all `universityid` and SSN pairs ever used. The merge process:
  1. If a record with the same `universityid` and SSN occurs in both files, the one from `uaid.file` is ignored.
  2. Records with all-zero SSNs are written to an error file.
  3. If two of the remaining records have the same SSNs but different `universityids`, they are written to an error file.
  4. All remaining records are passed to the next step.
- The passed records are merged with `allcurr.dictouta` to fill in `universityids` missing from `allcurr.dictouta`; the corrected file is written as `dicupdat.work3`. The merge process (`dicupda0.rexx`):
  1. Records from the two files are paired based on the SSN field.
  2. If the passed record does not have a `universityid`, the surnames are compared. If the surnames are not the same, the `dictouta` record is written to the output file, but the mis-matched record pair is also written to an error file. If the surnames match, the

universityid from the passed record is overlaid on the dictouta record, which is written to the output file.

3. If both records have universityids and they match, the dictouta record is written to the output file.
4. If the universityids do not match, but the dictouta record has the “alter-ego” flag on and its universityid reference field matches the universityid in the passed record, the dictouta record is written to the output file.
5. Otherwise, the dictouta record is written to the output file, but the mismatched record pair is also written to an error file.

This process is repeated for alllast.dictoutd, producing dicupdat.work4.

- The files dicupdat.work3 and dicupdat.work1 are compared to fill in SSNs missing from dicupdat.work3; the corrected file is written as dicupdat.dictoutb. The merge process (dicupda1.rexx):

1. Records from the two files are paired based on the universityid field.
2. If the work3 record does not have an SSN, the surnames are compared. If the surnames are not the same, the work3 record is written to the output file, but the mis-matched record pair is also written to an error file. If the surnames match, the SSN from the work1 record is overlaid on the work3 record, which is written to the output file.
3. Otherwise, the work3 record is written to the output file.

This process is repeated for dicupdat.work4, producing alllast.dictoute.

- The file dicupdat.dictoutb is processed to attempt to fill in campus addresses for newly-arrived users. The corrected file is written to allcurr.dictoutb. The correction process:

1. Records from dicupdat.dictoutb with a street address that does not start “Not here until” are copied directly to the output file.
2. Records from dicupdat.dictoutb that do have a street address starting “Not here until” are matched (by universityid) against records from tesspers.file that have the same SSN in columns 10 and 100 (these are the “current SSN records”):

— If no match is found in the tesspers.file, the campus address is set to “Address unknown” and the record is written to the output file.

— If a match is found in the tesspers.file and the tesspers record indicates “pending” or “impending”, the dictoutb record is copied unchanged to the output file.

— If a match is found in tesspers.file and the user is neither pending nor impending, the universityid is matched to homedepd.file to get the department number. If no match is found in homedepd.file, the department number from the tesspers.file is used instead. In either case, the department number is matched against the deptaddr.file (a manually maintained file) to obtain the departmental address. That address is inserted into the record, which is written to the output file.

— Records with changed addresses are also written to the file dicupdt.updates to be used as input to Whale to correct missing addresses in the \$File. (Note that Whale feeds these updates to both the \$File and the LDAP directory.)

- The files `allcurr.dictoutb` and `alllast.dictoute` are merged to produce `allcurr.dictout` (using the newer record for netids that occur in both files).
- The original \$File dictionary file (`allcurr.dictouta`) is compared with the updated dictionary file (`allcurr.dictoutb`) to generate a file (`allcurr.updates`) to be used as input to Whale to correct SSNs and universityids. Any records that occur in `dictoutb` but not in `dictouta` or *vice versa* are written to error files. (Neither of these should occur.)
- The original \$File dictionary file (`allcurr.dictouta`) is examined to find netids that are not marked obsolete or inactive but that have a termination date which has been reached. Those netids are written to the file `delete.userids`.

### **rusupdat.exec**

The script `rusupdat.exec` updates the `recent.userids` file from new information in `allcurr.dictoutb` and generates Whale commands to add new users found in `allcurr.dictoutb`. (These Whale commands are no longer executed.)

The records in `recent.userids` are in the form:

```

1- 48 Netid
50- 51 Version code ("00" for primary; "0n" for others)
52     Campusid conflict indicator (deprecated)
53- 55 Status
56     "A" for alias
57     "I" for inactive; "O" for obsolete
58     "e" for alter-ego
59- 62 FYnn for fiscal year assigned
64- 67 Fxnn for most recent fiscal year in use
69- 77 Universityid
79- 87 SSN
89- 97 Universityid reference
99-148 User's name

```

The steps in the update process are:

- The `recent.userids` file is split into two files, `rusupdat.work0` for records in which the version code is "00" (indicating that this is the primary or only use of this netid) and `rusupdat.previds` for all others. (Historically, some netids have been assigned twice, although this practice is strongly discouraged.)
- The files `rusupdat.work0` and `allcurr.dictoutb` are matched on the netid:
  1. Records that occur only in `rusupdat.work0` (that is, records for netids that are no longer in use) are copied directly to the output file, `rusupdat.work1`.

2. Records that occur only in `allcurr.dictoutb` (records for newly-added netids) are copied directly to `rusupdat.work2` (to be added to `recent.userids`) unless the obsolete flag is on. (Obsolete netids are ones that were added incorrectly in the first place, such as due to misspellings, or ones that are known no longer to be needed for the original purpose, such as for mainframe services that are no longer available. There is no point in adding *new* obsolete netids to `recent.userids` as “00” records, as they are clearly mistakes.)
3. Records that occur in both files are processed by the `rusupda1.rexx` script, which tests them to see if the `recent.userids` record needs correction:
  - If the user name (upper-cased), SSN, universityid, status, alias state, and inactive or obsolete state are the same in the records from both files, a record is written to `rusupdat.work1` using the user name and last-fiscal-year-used fields from the `dictoutb` record.
  - If the user names are different and the SSN or universityid also differs:
    - If the netid is marked “obsolete”, the record from `rusupdat.work0` is copied to the output file, `rusupdat.work1`.
    - If the most recent fiscal year in use in the `rusupdat` record is not the current one (which for historical reasons is encoded “FZ01”), the combined records from the two input files are written to an error file (there should be no such records). The `rusupdat` record is corrected to make the fiscal year “01” and is written (otherwise unchanged) to the output file, `rusupdat.work1`. The `dictoutb` record is written to `rusupdt.work5` to be added to `recent.userids`.
    - Otherwise, the combined records from the two input files are written to an error file (there should be no such records). The universityid, SSN, and user name are copied from the `dictoutb` record to the `rusupdat` record, which is written to the output file, `rusupdat.work1`.
  - Otherwise, the output record is corrected:
    - If the universityids are the same but the SSNs are different, the SSN from the `dictoutb` record is used.
    - If the SSNs are the same but the universityids are different, the universityid from the `dictoutb` record is used.
    - If the universityids are the same but the inactive or obsolete state differs, the state from the `dictoutb` record is used.
    - If the universityids are the same but the alias state is different, the state from the `dictoutb` record is used.
    - If the universityids are the same but the status differs, the status from the `dictoutb` record is used.
    - If the user names (upper-cased) are the same but the SSNs or universityids differ, the SSN or universityid from the `dictoutb` record is used.

The corrected record is written to `rusupdat.work1` using the user name and last-fiscal-year-used fields from the `dictoutb` record. (Note that this process does not update the `universityid` reference field nor the alter-ego state, neither of which is contained in the 240-byte records in `allcurr.dictoutb`.)

- The files `rusupdat.work2` and `rusupdat.work5` (both of which contain `dictoutb` records that need to be added to `recent.userids`) are merged and processed by the `rusupda2.rexx` script, which converts them to the format of `recent.userids` records. In the reformatting, the obsolete state takes precedence over the inactive state; the last-fiscal-year-used field is set to “FZ01” unless the “old file” indicator is on in the `dictoutb` record, in which case it is set to “FZ00”. The version code is set to “00” in all cases.

The reformatted records are merged with the files `rusupdat.work1` and `rusupdat.prevuids` and sorted on the `netid` and version code. (The sort algorithm is a stable one; that is, the records written by `rusupda2.rexx` appear in the output before the records with the same sort key from `rusupdat.work1` and `rusupdat.prevuids`.) All resulting records are written to the new `recent.userids` file. In addition, any records with the same `netid` and version code are written to an error file.

- The new `recent.userids` is compared with `allcurr.dictoutb` to generate the Whale commands to add any new `netids` to the `$File`. Duplicate records in `recent.userids` (records with the same `netid` and version code) are ignored, as are records with version codes not set to “00”. Only records for the current fiscal year are used. Inactive and obsolete `netids` are skipped. Records with `netids` longer than 8 characters are ignored, as are `netids` containing the characters underscore, period, single quote, or ampersand (such `netids` are not legal in Whale, so there would be no `$File` record to update). Records with a status of “lsp” (ListProc `netids`) are also ignored, as those records are maintained only in LDAP, not in the `$File`.

The formatting of the Whale commands is done by the script `rusupda3.rexx`. The Whale commands are written to `rusupdat.updates`. In generating the address field for the Whale updates, `rusupda3` uses “Xalias” for `netids` with a status of “xal” and “Generated by RUSUPDAT” for all others.

Note that the Whale commands generated by this script are no longer executed, but that portion of the script is still run for the purpose of detecting errors.

### **adrupdat.exec**

The script `adrupdat.exec` generates Whale alter commands to update campus addresses and telephone numbers for faculty, staff, and students who have an SSN in `allcurr.dictoutb`. The new campus addresses and telephone numbers come from `stf.direct`, `gsphones.direct`, and `stu.direct`, which are derived from the campus telephone directory, and from a manually maintained file, `adrupdat.deptaddr`.

Note that the Whale commands generated by this script are no longer executed, but the script is still run for the purpose of detecting errors.

The steps in `adrupdat`’s processing:

1. Reports are generated showing faculty, staff, undergraduates, and graduate students whose dictionary records are missing either the SSN or the `universityid`.

2. The dictionary entries for faculty and staff for whom the SSN is known are compared (on the SSN) with the file stf.direct. (If there are multiple entries for a given SSN in stf.direct, all but the first are ignored.) Entries not found in stf.direct are written to an error file.

If a match is found in stf.direct, the stf.direct record is appended to the dictionary entry. The combined records are processed by the adrupda0.rexx script:

- a. The correct phone number is determined. If a phone number was found in the stf.direct record, that is stripped of extraneous blanks and left-adjusted in a 12-character field. Otherwise, the string "NONE" is used.
- b. The correct address is determined. The address from stf.direct is selected and is assumed to be a campus-mail address.
- c. Whale update commands are generated only if there is a difference between the "corrected" phone number or address or campus-mail indication and those in the portion of the record derived from the dictoutb record.

The Whale commands are written to adrupdat.fsuptds, but are not executed.

3. The dictionary entries for graduate students for whom the SSN is known are compared (on the SSN) with the file stf.direct. If a match is found in stf.direct, the stf.direct record is appended to the dictionary entry. The combined records are put through adrupda0.rexx (as above) to generate Whale update commands to correct the phone number or address.

If no match is found in stf.direct, the records are compared (on the SSN) with the file gsphones.direct. (If there are multiple entries for a given SSN in gsphones.direct, all but the first are ignored.) Entries not found in gsphones.direct are written to an error file.

If a match is found in gsphones.direct, the gsphones.direct record is appended to the dictionary entry. The combined records are matched (on the department code) with records from the file adrupdat.deptaddr, which contains a list of department addresses. The matching deptaddr record is appended to the combined record, which is then processed by the adrupda1.rexx script, as follows:

- a. The correct phone number is determined. If a home phone number was found in the gsphones.direct record, that is stripped of extraneous blanks and left-adjusted in a 12-character field. If no home phone number was found, but an office phone number was found, that is stripped of extraneous blanks and left-adjusted in a 6-character field. Otherwise, the string "NONE" is used.
- b. The correct address is determined. The department address is used unless the gsphones.direct record shows the address as "Recently graduated". In either case, the address is assumed to be a campus-mail address.
- c. Whale update commands are generated only if there is a difference between the "corrected" phone number or address or campus-mail indication and those in the portion of the record derived from the dictoutb record.

All of the Whale commands for correcting graduate student information are written to adrupdat.gsupdts, but are not executed.

4. The dictionary entries for undergraduate students for whom the SSN is known are compared (on the SSN) with the file `stu.direct`. (If there are multiple entries for a given SSN in `stu.direct`, all but the first are ignored.) Entries not found in `stu.direct` are written to an error file.

If a match is found in `stu.direct`, the `stu.direct` record is appended to the dictionary entry. The combined records are processed by the `adrupda2.rexx` script:

- a. The correct phone number is determined. If a phone number was found in the `stu.direct` record, that is stripped of extraneous blanks and left-adjusted in a 12-character field. Otherwise, the string “NONE” is used.
- b. The correct address is determined based on the address field from `stu.direct`:
  - If there is none, the address is set to “Undergraduate student” with campus-mail specified.
  - If it a College designator (a single character between left and right parentheses), the designator is removed and the address is marked as campus mail.
  - If it is “Field studies”, “Field study”, “Foreign studies”, “Foreign study”, “Recently graduated”, “Undergraduate student”, or “Not currently enrolled”, it is marked as campus mail.
  - Otherwise, it is marked as U.S. mail and a second address line “Princeton, NJ 08540” is added for the Whale update.
- c. Whale update commands are generated only if there is a difference between the “corrected” phone number or address or campus-mail indication and those in the portion of the record derived from the `dictoutb` record.

The Whale commands are written to `adrupdat.ugupdts`, but are not executed.

### **lowaddrs.exec**

The script `lowaddrs.exec` updates the address fields in `allcurr.dictoutb` to be in mixed case. It also generates Whale commands to correct the case of addresses in the `$File`. Those Whale commands are no longer executed, but the script is still run for the purpose of detecting errors.

Each address in `allcurr.dictoutb` is passed through the script `lowaddr0.rexx` to convert the address to mixed case, if required. If the 96-character address field contains at least one lower-case letter, it is simply written to the new copy of `allcurr.dictoutb` unchanged. Otherwise, each of the three 32-character address subfields is processed. Letters immediately following any of “,-\_0123456789” are upper-cased; all others are lower-cased. The first substring is scanned for the following blank-delimited words, which are converted as shown:

- Ce to CE
- Cees to CEES
- Ceor to CEOR
- Cvcs to CVCS
- C/o to C/O
- Ewa to EWA
- Fm to FM
- Gfdl to GFDL

- Lob to LOB
- Macmillan to MacMillan
- Mae to MAE
- Ntpu to NTPU
- Ppl to PPL
- Pmi to PMI
- Sei to SEI
- Stk to STK
- Wprb to WPRB
- Wwnff to WWNFF

If these permutations resulted in no change, the original record is simply copied to the new file. Otherwise, the updated dictoutb is copied to the new file and the Whale command to correct the address is written to lowaddrs.updates. The Whale command is not executed.

### **editphon.exec**

The script editphon.exec reformats telephone numbers in allcurr.dictoutb to be in a standard form. It also generates Whale commands to correct the telephone number formats in the \$File. Those Whale commands are no longer executed, but the script is still run for the purpose of detecting errors.

Each telephone number in allcurr.dictoutb is passed through the script editpho0.rexx to convert to the canonical form, if required. The file allcurr.dictoutb is rewritten to reflect the corrected phone numbers. Records with a bad phone number are written both to an error file and to allcurr.dictoutb.

- If the length of the phone number is 12, a blank or slash in column 4 is converted to a hyphen. “609” at the beginning is deleted.
- If the length of the phone number is now 10, the string “C/O 8-6072” is converted to “8-6072”.
- If the length of the phone number is now 8, these conversions are performed:
  - “C/O 6072” is converted to “8-6072”.
  - “243-2” and “243-3” have the “243-” converted to “125-” (internal phone number for PPPL).
  - “258-” is converted to “8-” (internal phone number for main-campus phones).
  - “986-” is converted to “6-” (internal phone numbers for students).
- If the length of the phone number is now 7, “258” at the beginning is replaced by “8-”, and “986” at the beginning is replaced by “6-”. “STUDENT” is replaced by “NONE”.
- If the length of the phone number is now 5, a leading “8” is converted to “8-”; a leading “6” is converted to “6-”; and a leading “X” is converted to “8-”.
- If the length of the phone number is now 4 and it is not “NONE”, it is prefixed by “8-”.
- If the length of the phone number is now 6, a leading “2-” is converted to “8-” (to convert an old-style University phone number), and “8-0000”, “8-XXXX”, or a number with a leading “4-” (an old student number) is replaced with “NONE”.
- If the length of the phone number is 0, it is replaced by “NONE”.

The resulting phone numbers are then tested for validity:

- If the length of the phone number is now 12, it is tested to be sure that the fourth and eighth characters are hyphens and that the others are all digits. If not, the record is written to the error file.

- If the length of the phone number is now 8 and it is not “WITHHELD”, it is tested for a hyphen in the fourth position and digits in all others. If not, the record is written to the error file, as above.
- If the length of the phone number is now 6, it is tested to be sure that it begins with “8-” or “6-” and that the rest of the string is digits. If not, it is written to the error file, as above.
- If the length of the phone number is now 4 and it is not “NONE” or if it is any other length, it is written to the error file, as above.

In cases where phone numbers have been corrected, the appropriate Whale update commands are written to editphon.updates, but they are not executed.

### **uaidwhal.exec**

There are two .uaidadd files (one in the format of the tesspers.file and the other in the format of the uaid.file). The .uaidadd files list all entities not known to Campus Community that have computing accounts. The files are used by various utilities, such as those to format the campus telephone directory. The uaidwhal.exec rebuilds the two files to pick up any new users with dummy universityids added by Whale since the previous night. (Dummy universityids are defined as those starting “09”, but not with “0980” (which is used for the organizational entries that come from deptdir.filecurr).)

The file allcurr.dictoutb is read and the records for non-obsolete, non-inactive netids with dummy universityids are examined to update the tessplus.uaidadd and uaidplus.uaidadd files. These records from allcurr.dictoutb are compared (on universityid) with the records already in tessplus.uaidadd:

- *Records that match in allcurr.dictoutb and tessplus.uaidadd:* The tessplus.uaidadd record is appended to the allcurr.dictoutb record, which is then matched on universityid with the records already in the file uaidplus.uaidadd:
  - *Records that match in allcurr.dictoutb and tessplus.uaidadd and uaidplus.uaidadd:* The uaidplus record is appended to the dictoutb/tessplus record, and the netid fields derived from the dictoutb record and the tessplus record are compared:
    - *Dictoutb and tessplus netids the same:* The dictoutb portion of the combined records is processed by the uaidwha3.rexx script, which builds updated tessplus and uaidplus records for these existing dummy universityids. These records are written to the revised tessplus.uaidadd and uaidplus.uaidadd files.
    - *Dictoutb and tessplus netids different:* If the netids differ, there must be more than one campusid for the user. The existence of the primary campusid (netid) must be confirmed. The netid from the tessplus record is searched for in allcurr.dictoutb:
      - *Tessplus netid found in dictoutb:* The universityid in the dictoutb portion of the combined record is compared with the universityid in the dictoutb record with the matching netid:
        - *The universityids are the same:* The original tessplus and uaidplus records are copied to the revised tessplus.uaidadd and uaidplus.uaidadd files.

- *The universityids are different:* This should not happen. If it does, the primary campusid exists but has changed its universityid. The record is written to an error file, and the original tessplus and uaidplus records are copied to the revised tessplus.uaidadd and uaidplus.uaidadd files.
- *Tessplus netid not found in dictoutb:* This should not happen. If it does, the primary campusid no longer exists. The record is written to an error file, and the original tessplus and uaidplus records are copied to the revised tessplus.uaidadd and uaidplus.uaidadd files.
- *Records from allcurr.dictoutb and tessplus.uaidadd that do not exist in uaidplus.uaidadd:* The combined dictoutb/tessplus record is written to an error file; the tessplus portion of the record is then copied directly to the output file, the revised version of tessplus.uaidadd.
- *Records from uaidplus.uaidadd that do not exist in allcurr.dictoutb and tessplus.uaidadd:* These records are copied directly to the output file, the revised version of uaidplus.uaidadd.
- *Records from allcurr.dictoutb that do not exist in tessplus.uaidadd:* These records are sorted on the universityid and only the last record for a given universityid is processed. (The sort is stable, so the last record will be the last one from the dictoutb file.) These are compared on universityid with the tesspers.file. No further action is taken if the record already exists in the tesspers.file.

If the record does not exist in the tesspers.file, it is processed by the uaidwha3.rexx script, which builds tessplus and uaidplus records for these new dummy universityids added by Whale. These records are written to the revised tessplus.uaidadd and uaidplus.uaidadd files.

- *Records from tessplus.uaidadd that do not exist in allcurr.dictoutb:* These records are copied directly to the output file, the revised version of tessplus.uaidadd.

Note that only the first occurrence of a given universityid is written to the two uaidadd files. (As before, all sorts are stable, so these will be the first records for the universityid from the dictoutb file.) Any duplicate records are written to error files.

The records in the uaidplus files are in the following format:

```

1- 9 Universityid
10- 18 SSN
19- 38 Surname
39- 53 First name
54 Middle initial
55- 62
63- 65 Department number
66
67- 80 Netid

```

The records in the tessplus files are in the following format:

1-	9	Universityid
10-	18	SSN
19-	38	Surname
39-	58	First name
59-	78	Middle name
79-	81	Status Code
82-	95	Netid
96-	98	Department number
100-	108	Current SSN

The uaidwha3.rexx script uses the following strategy to determine a department number for a user:

- If there is no comma in the name field from the dictoutb record (in other words, if this is a record for a non-human entity), the department is set to “992” (“Non-affiliates”) if the status is “out”; it is set to “995” (“Lists”) if the status is “lsv”; and it is set to “994” (“Generic and Group Affiliates”) otherwise.
- If there is a comma in the name field, the department is set to “992” if the status is “out” and to “993” (“Other Affiliates”) otherwise.

The uaidwha3.rexx script uses the following strategy to determine the name fields for a user:

- If there is no comma in the name field (a non-human entity), the middle name/initial is always empty. The first and last names are determined by:
  - If the name contains only one blank-delimited word, that word is used as the last name, and the first name is blank.
  - Otherwise, the name is split so that there is at least one word in the first name and one in the last. For the last name, as many words are used from the right end of the user name as will fit in 20 bytes. What remains is used for the first name; it may be truncated to 20 bytes for the tesspers file and to 15 bytes for the uaidplus file.
- If there is a comma in the name field:
  - Everything up to the first comma is treated as the last name, although, if necessary, it is truncated to 20 bytes.
  - The first two words of the remainder are used as the first and middle names. If necessary, the first name is truncated to 20 bytes for tesspers and to 15 bytes for uaidplus. The middle name is truncated to 20 bytes for tesspers and to a single initial for uaidplus.

### uaidqref.exec

The uaidqref.exec script picks up any new universityid reference fields added by Whale and enters them into recent.userids.

It begins by building and validating the uaidqref.file, which lists netids that have universityid references. The format of the file is:

```
1- 9 Universityid
10- Reference type ("e" for "alter-ego"; blank otherwise)
11- 19 Universityid reference
20- 27 Netid
```

The logic to rebuild the file:

- The \$File is read to extract all records for live netids that have a non-empty universityid reference field. The netid and universityid reference fields are written to an output record along with a one-byte field that contains an "e" if the alter-ego flag is on and a blank otherwise.
- These records are compared (on the netid) against the version "00" records from recent.userids to get the universityid for the netid. If there is no such netid in recent.userids, the unmatched record is written to an error file.
- The universityid reference is then compared against the version "00" records for the current fiscal year in recent.userids to find the referenced netid. If there is no such universityid in recent.userids, the unmatched record is written to an error file.
- If any self-references are found, they are written to an error file.
- All remaining records are written to uaidqref.file.
- After any records that have the same universityid, alter-ego flag, and universityid reference field have been reduced to only one record, any remaining cases of duplicate universityids are written to an error file.

The new uaidqref.file is then used to update recent.userids:

- The version "00" records from recent.userids are compared with the records in uaidqref.file on the netid. If a match is found, the reference-type flag and the universityid reference are copied from the uaidqref record to the recent.userids record.
- All recent.userids records are then examined to find self-references (universityid the same as the universityid reference). Any such records are updated to remove the reference-type flag and the universityid reference.

**allcumov.exec**

The overnight processing is aborted at this point unless the following files are determined to have been newly created: allcurr.dictout, allcurr.dictouta, allcurr.dictoutb, recent.userids, tessplus.uidadd, uaidplus.uidadd, and uaidqref.file.

**delother.exec**

If the file delete.userids exists, the script delother.exec is executed. (delete.userids is created by dicupdat.exec in the overnight processing and by other scripts in special-purpose processing; it lists netids that should be deleted because their termination date has passed.)

delother.exec builds delothed.carfile, which contains a list of actions to be taken to delete the netids, including deleting the subid records in Whale, marking the netid inactive in Whale, deleting the LDAP entry, and, if mail forwarding information is available, adding an X-alias entry to LDAP. The processing steps:

- The passdate.exec script is invoked to refresh the file passdate.fileall, which lists the CMS and unix password change dates for all active accounts (subids) found in the \$File.
- The delete.userids file is matched (on the netid) with the records for the current fiscal year with 1-8 character netids from recent.userids. Unmatched records are written to an error file.
- For each netid that is found in both delete.userids and recent.userids, two or more records are written to delothed.carfile (sorted by name, action code, and subid, if any):
  1. A record containing the action code “D” (delete the .STUDY subid and mark the netid inactive in Whale) is written for each netid.
  2. Each netid is searched for among the non-.STUDY subids listed in passdate.fileall. For each netid.subid combination found, a record with the action code “B” (delete subid from Whale) is written.
  3. Each netid is searched for in the files mailbox.file and emailboa.file (created by ldiupdad.exec) to determine whether any mail-forwarding information is available. The delothe1.rexx script makes this decision based on the mail address from the mailbox.file, unless that address is for delivery to an OIT host (arachne, aragon, arelia, ariel, arizona, arundel, ernie, exchange, flagstaff, imap, mail, phoenix, pucc, pucc2, student, tucson, or yuma), in which case it instead bases the decision on the mailbox alternate address from the emailboa.file, if there is one.

In no mail-forwarding information is found (netids delivering to an OIT host (same list as above), netids delivering to forward.Princeton.EDU, and netids with no mailbox), a record containing the action code “L” (delete the netid from LDAP) is written to delothed.carfile.

If mail-forwarding information is found (netids with delivery to another netid with a mailbox on Princeton.EDU, netids with delivery to the alumni system, netids with delivery to a non-OIT Princeton system, and netids with delivery to a non-Princeton system), the netid is searched for (by universityid) in records from the campusid.file that have been reformatted to have the universityid and one campusid in each record (but excluding any

campusids longer than 18 characters). For each campusid found, a record containing the action code “L” (delete the netid with this campusid from LDAP) is written to delothed.carfile. In addition, for each campusid found, a record containing the action code “X” (build an X-alias LDAP entry) is written to delothed.carfile.

### **delodat1.exec**

If the file delothed.carfile (created in the previous step) exists, the delodat1.exec script is executed. For each record found in delothed.carfile, the delodat1.rexx script is invoked to format and execute the appropriate Whale or LDAP commands:

- *Action Code B:* Whale commands are issued to set all disk quotas for the subid to zero and to delete the account (subid). (Whale updates LDAP as appropriate.)
- *Action Code D:* Whale commands are issued to set all disk quotas for the .STUDY subid to zero, to delete the account, and to mark the netid inactive. (Whale updates LDAP as appropriate.)
- *Action Code L:* The universityid is searched for in LDAP. One and only one entry should be found. If that is the case, the LDAP command to delete that entry is issued.
- *Action Code X:* An end date for the X-alias entry is determined:
  - If the status is “ret” (retiree), no date is set.
  - If the status is “gra” (recently graduated), the date is set to the beginning of the year following graduation.
  - Otherwise, the date is set to one year from the current day.

LDAP commands are issued to create an entry with this format to cause the person’s email to be forwarded to the specified mailbox:

```
dn: uid=netid,ou=Xaliases,o=Princeton University,c=US
campusid: netid
cn: Xalias
emailbox: emailbox
mail: netid@Princeton.EDU
objectclass: inetOrgPerson
objectclass: organizationalPerson
objectclass: person
objectclass: puPerson
objectclass: top
ou: Xaliases
puinternalinfo: mm/dd/yyyy delete Xalias entry
sn: Xalias
uid: netid
universityid: universityid
```

**addrupdt.exec**

The script addrupdt.exec processes address.file, the daily Campus Community address updates. The format of the file is:

```
1- 9 Universityid
10- 14 Address type:
      CAMP -- Campus Interoffice Mail Address
      HOME -- Home Address (Employees)
           -- Local Address (Students)
      MAIL -- Student's Permanent Address
      OL1  -- Primary Office Address
      OL2  -- Secondary Office Address
15- 69 Address Line 1
      (or "DELETE" to indicate that there is no longer
      an address of this type)
70-124 Address Line 2
125-179 Address Line 3
180-209 City
210-211 Province/State Abbreviation -- U.S., Canada, Mexico only
212-241 Country
242-251 Postal/Zip Code
```

The processing is:

1. The address updates from address.file are sorted on universityid and address type. If more than one entry is found for a given universityid and address type, the last one is used and the others are written to an error file.
2. The address updates are applied to address.ccfile, the cumulative file of addresses from Campus Community. Any in which the first character of the first address line is blank are also written to an error file.

**phonupdt.exec**

The script phonupdt.exec processes phone.file, the daily Campus Community telephone number updates. The format of the file is:

```
1- 9 Universityid
10- 14 Phone number type:
      ASTR -- Asterisk Phone (Assistant's Phone)
      FAX  -- Facsimile Phone
      HOME -- Home Phone (Employees)
           -- Local Phone (Students)
      MAIL -- Student's Permanent Phone
      OL1  -- Primary Office Phone
      OL2  -- Secondary Office Phone
      PAC  -- Phone Access Code
      VCML -- Voicemail Box
15- 29 Telephone Number
      (or "DELETE" to indicate that there is no longer
      a phone number of this type)
```

The processing is:

1. The phone updates from phone.file are sorted on universityid and phone type. If more than one entry is found for a given universityid and phone type, the last one is used and the others are written to an error file.
2. The phone updates are applied to phone.ccfile, the cumulative file of phone numbers from Campus Community. Any in which the first character of the phone number is blank are also written to an error file.

**campldap.exec**

The campldap.exec script updates campcomm.file and ssncurr.file from the daily ldaptxt.file of Campus Community transactions. Any future-dated status changes to Active are held pending in the ldaptxt.pending file if they are more than seven days in the future for PPL employees or more than three months (92 days) in the future for Human Resources and Dean of the Faculty employees and students; any other other future-dated status changes are held until their effective date. (This keeps netids from being assigned and LDAP entries from being built too far in advance for incoming faculty, staff, and students.)

Once future-dated people have been given netids, their records are also kept in the ldaptxt.pending file with a status of "a" until their activation date arrives, when Campus Community again writes an "A" transaction record. The purpose of this delay is to allow any possible intervening status updates, such as termination of Casual Hourly status, to be processed when they come through without undoing the new status.

The format of the Campus Community transaction records is:

Columns	Length	Field Name	Field Value
1-9	9	Person ID (UAID)	Numeric
10-18	9	Current SSN	Numeric
19-43	25	Last Name	Alphabetic, may contain blanks
44-68	25	First Name	Alphabetic, may contain blanks, may be just an initial
69-93	25	Middle Name	Alphabetic, may contain blanks, may be just an initial
94-95	2	Affiliation	PS Affiliation of the person
96-98	3	Job Function	PS Job Function for an employee (or else blank)
99-100	2	Affiliation group	PS Affiliation Group of the person
101	1	Status	Status of the person's affiliation
102-109	8	Status Date	CCYYMMDD - date of the status change
110-111	2	Class Year	Current class year for students (e.g., 03 for undergrad or G2 for graduate students)
112-114	3	Department Code	Home department (numeric) for employees and (alpha) for graduate students; rescol for undergrads, if known; otherwise, blanks.
115-116	2	Cohort Year	Year + 1 of entry for graduate students (e.g., 02 for entry in Fall of 2001)
117-124	8	Generals Date	Date graduate student passed General Exams, CCYYMMDD; may be blank if not yet passed Generals
125-128	4	Frist Mailbox	Frist mailbox number for undergrads (and graduate students living in undergrad dorms as advisor)
129-176	48	netid	netid (today will be only 8 characters for people, but will be longer in the future. Already longer for non-people)
177-247	71	Email address	Full Email address (e.g., scoletti@Princeton.EDU)
248-255	8	Birthdate	Date of birth, CCYYMMDD; may be blank if not known
256-285	30	Res College	Residential college (e.g., UBTULER or URKEFELLER)

The format of the campcomm.file records is:

Columns	Length	Field Name	Field Value
1-9	9	Person ID (UAID)	Numeric
10-18	9	Current SSN	Numeric
19-43	25	Last Name	Alphabetic, may contain blanks
44-68	25	First Name	Alphabetic, may contain blanks, may be just an initial
69-93	25	Middle Name	Alphabetic, may contain blanks, may be just an initial
94-95	2	Affiliation	PS Affiliation of the person
96-98	3	Job Function	PS Job Function for an employee (or else blank)
99-100	2	Affiliation group	PS Affiliation Group of the person
101	1	Status	Status of the person's affiliation
102	1	netid status	Should this person should have a netid.
103	1	Campus LDAP status	List this entry in the campus LDAP.
104	1	Alumni LDAP status	List this entry in the alumni LDAP.
105	1	Printed Directory status	List this entry in the printed Campus Directory.
106-113	8	Start Date	CCYYMMDD - date of the start of this status.
114-121	8	End Date	CCYYMMDD - date of the end of this status.
122-123	2	Class Year	Current class year for students (e.g., 03 for undergrad or G2 for graduate students)
124-126	3	Department Number	Home department number for employees and graduate students; may be blanks if unknown or not an employee or graduate student.
127-129	3	Department Code	Department code for graduate students.
130-131	2	Cohort Year	Matriculation class year for undergraduate students; year + 1 of entry for graduate students (e.g., 02 for entry in Fall of 2001)
132-139	8	Generals Date	Date graduate student passed General Exams, CCYYMMDD; may be blank if not yet passed Generals
140-143	4	Frist Mailbox	Frist mailbox number for undergraduates (and graduate students living in undergraduate dorms as advisor)
144-151	8	Birthdate	Date of birth, CCYYMMDD; may be blank if not known

The file `ldaptxt.file` (created by `ldaptxtg.exec`) contains the day's Campus Community transactions. The file `ldaptxt.delayed` contains transactions from earlier days that have not yet been processed (because multiple requests for a given user were received on a previous day). These two files are combined (with the records from `ldaptxt.delayed` first, so that any delayed transactions will be processed before transactions for the same universityids received today).

The records from this combined file are processed as follows:

- The combined file may need correcting before it can be processed:
  - Records that match a record from `ldaptxt.fixdel` are discarded.
  - Records that match the “before” portion of a record from `ldaptxt.fixrep` are replaced by the “after” portion of the same record.
- Records indicating an employee update that do not have a job function (“EM” in columns 94-95 and blank in columns 96 and 97) are discarded to an error file.
- Records indicating PPL casual hourly employees (“EMCH\_PLA” in columns 94-101) with the department number set to “999” are corrected to have a department number of “395”. This is necessary in order for the initial password envelopes to be sent to PPL.
- : • Records for undergraduates (“ST\_\_UG” in 94-100) are corrected to have a 3-character residential college code in the department code field.
- : • Records for alumni/ae are corrected. “AL\_\_GRG” in 94-101 becomes “AL\_\_GRA” and “AL\_\_UGG” becomes “AL\_\_UGA”.
- Records with “AB\_” or “BSE” in the department code field are corrected to have blanks in that field.
- Additional records are brought in from the `ldaptxt.pending` file. Those with an effective date no later than today and a status of “a” (active pending) have their status changed to “A”.
- Records for people changing to any status other than “A” (active) are examined to determine whether the status date field is before the current date (or blank). If the date has not yet been reached, the records are prefixed with the date in which they appeared in `ldaptxt.file` and are written to the revised `ldaptxt.pending` for processing at the appropriate time.
- The decision as to whether to delay processing of records for people changing to a status of “A” is made as follows:
  - : — Records for Human Resources and Dean of Faculty employees, students, and special
  - : students are processed if the status date is no more than 92 days in the future.
  - Records for PPL employees and non-students are processed if the status date is no more than 7 days in the future.
  - An exception to the above is made if the status date is in the future and the record is for a “benefits-eligible non-employee” (“EMNE” in columns 94-97). The status code in these records is changed to “a” (from “A”) and the SSN is blanked out (in case it is updated before the transaction is replayed); then the records are written to `ldaptxt.pending`. In other words, “EMNE” activations are treated like terminations to keep them from getting into

campcomm.file until they become effective, in case the person is presently an active employee as well as being eligible for benefits.

— All other records are written to ldaptxt.pending for processing in the future.

— Even though records are processed (so that a netid will be added), if the status date is in the future, some processing is deferred to the next time Campus Community presents an “A” record for the person. A copy of the record, with the status set to “a” (rather than “A”) and the SSN blanked out, is written to ldaptxt.pending for later processing. The presence of this record allows a clean-up of any possible intervening status updates, such as termination of casual hourly status. The SSN is blanked in case it is updated before the transaction is replayed.

- All of the records that are to be processed today (whether from ldaptxt.file, ldaptxt.delayed, or ldaptxt.pending) are compared on universityid with the uaidcurr.file (an historical list of universityids that have changed). If the universityid being processed matches an old universityid in uaidcurr.file, the record being processed is updated with the new universityid from the uaidcurr.file record.
- The records are then sorted on universityid and all but the first record for each universityid are written to ldaptxt.delayed (so that only one record for a given user is processed in each run). Because the sort is stable, the record selected for processing in this run will be the one that was earliest in the combined file created by appending ldaptxt.file to ldaptxt.delayed. Thus, the earliest of the available updates from Campus Community will be processed in this run.
- A sort key is appended to the records. It consists of the universityid followed by the contents of columns 94-101 of the records followed by three blanks. The records are sorted on this key.
- Records with department code fields that are not blank and not all numeric are compared with ldaptxt.deptcode, a manually-maintained table of department codes (or residential college codes), department numbers, corrected department codes (or corrected residential college codes), and department (or residential college) names. If no match is found, the record is written to an error file and not processed further. If a match is found, the department code field in the record is replaced with the numeric equivalent, and the three blanks at the end of the sort key are replaced with the corrected alphabetic code.
- Records are then processed differently according to their contents:
  1. *Records with no affiliation, job function, affiliation group, or status:* Records are matched (by universityid) with campcomm.file. If there is no match, the record is not processed further; it is written to an error file (except records with universityids beginning with “02” (alumni/ae), which are discarded). If there is a match, the campcomm record is appended to the end of the record for further processing. If there are multiple matches, as many copies of the record as necessary are made so that each of the matching campcomm records can be appended to a copy of the record.
  2. *Records with no affiliation:* These records are written to an error file and not processed further.
  3. *All other records:* The remaining records are matched (on the sort key) with the campcomm.file records that did not match a record in category 1 above.

- a. *Matches*: If there are one or more matches in campcomm, they are appended, as above.
- b. *Non-matches*: Records for which there is no match in campcomm.file are matched again on a less complete sort key. The sort keys in both the unmatched ldaptxt records and the as-yet-unmatched campcomm records are changed so that the job function and affiliation group for “EM” records is blanked and the affiliation for student records is changed from “ST” to “AL” (alumnus):
  - i. *Matches*: Matching records are appended, as above.
  - ii. *Non-matches*: The unmatched records are matched (by universityid) with the as-yet-unmatched campcomm records:
    - *Matches*: All of the campcomm records matched in this way are copied to the output file, the revised campcomm.file. The match with the lowest sort key is also appended to the ldaptxt file, as above, for further processing.
    - *Non-matches*: Records that have an SSN and a last name specified, are reformatted to make a new campcomm.file record and are written to the output file, the revised campcomm.file. Those that do not have both an SSN and a last name are matched (by universityid) against cctxt.file, a list of all people in the most recent full dump from Campus Community):
      - *Matches*: The SSN and the first, middle, and last name fields from the cctxt.file are used to reformat the ldaptxt record into a new campcomm.file record, which is written to the output file, the revised campcomm.file.
      - *Non-matches*: These records are written to an error file.
    - *Still-unmatched campcomm records*: These records are copied to the output file, the revised campcomm.file.

If the paired record resulting from the matches in 3.a. or 3.b.i. is for an employee (“EM” in columns 94-95), and the portion of the record from campcomm.file is one with “A” in the status field, further processing is required unless the effective date for the transaction is later than the active status date in the campcomm record:

- If the effective date of the transaction and the active date in the campcomm record are the same:
  - If the transaction is for a faculty member becoming emeritus (“EMFR\_DFR” in columns 94-101), the transaction should be processed.
  - If the transaction is for a termination of a casual hourly employee (“EMCH\_HRT” in columns 94-101) and the matching campcomm record shows that the person became an active casual hourly employee on the same date, the transaction is discarded and the original record from campcomm.file is copied to the output file, the revised campcomm.file. Otherwise, the termination is processed.
  - All other records are written to the file campcomm.chekupst to be checked by hand, as they represent updates that are not being applied because the employee already has an active status with the same date. The original campcomm.file records are

copied to the output file, the revised campcomm.file, and the transaction is discarded.

— If the effective date in the transaction is earlier than the active date in the campcomm record, the original campcomm record is copied to the output file, the revised campcomm.file, and the transaction is discarded.

- Processing continues with all records that have not been discarded. (Each transaction record has a matching campcomm.file record appended to it.)
- If the transaction is for an update of the last name (column 19 is non-blank) and any of the three name fields in the transaction portion of the record differs from the corresponding field in the campcomm.file portion of the record, the record is copied to campcomm.nameupd and the three name fields are copied from the transaction portion of the record to the campcomm.file portion of the record. (This approach allows the blanking out of the middle name.)
- If the affiliation field in the transaction record (column 94) is non-blank and any of the affiliation, job function, and affiliation group fields differs between the transaction record and the campcomm record, the record is copied to campcomm.afflupd and these three fields are copied from the transaction portion of the record to the campcomm portion of the record.
- If the status field in the transaction record (column 101) is non-blank and differs between the transaction record and the campcomm record, the record is copied to campcomm.statupd and the status field is copied from the transaction portion of the record to the campcomm portion of the record.
- If there is a status date in the transaction and the status field is “A” and the status date in the transaction is not the same as the status date in the campcomm record, the status date is copied from the transaction portion of the record to the status date field in the campcomm portion of the record. In addition, if the status date in the transaction is earlier than the status date in the campcomm record, the record is copied to an error file.
- If there is a status date in the transaction and the status field is not “A” and the status date in the transaction is not the same as the status date in the campcomm record, the status date is copied from the transaction portion of the record to the *end date* field in the campcomm portion of the record.
- Similarly, the transaction record is examined for updates to class year, department number, department code, cohort year, generals date, Frist mailbox, and birthdate. Transactions are copied to the files campcomm.dlyrupd, campcomm.depnupd, campcomm.depcupd, campcomm.cohoupd, campcomm.geneupd, campcomm.frisupd, and campcomm.bdatupd, respectively, and the updated information is copied from the transaction portion of the record to the campcomm portion of the record.
- SSN updates are looked for after all other changes have been made. Records with SSN changes are written to campcomm.ssnupd and the new SSN is copied from the transaction portion of the record to the campcomm portion of the record.

The SSN updates are also used to update the ssncurr.file, replacing any record for the same universityid/SSN combination. (Replaced records are written to an error file.)

- The transaction portion of the records is removed and the revised campcomm record is written to the revised campcomm.file.

### camplupd.exec

The script camplupd.exec processes three of the transaction files created by campldap.exec (campcomm.nameupd, campcomm.depnupd, and campcomm.ssnupd) to drive Whale to apply the name, department number, and SSN updates to the \$File and LDAP. Its processing:

- Periods are removed from any first names specified in campcomm.nameupd. Names are formatted as “last, firstmiddle”, where “firstmiddle” is either the full first name and the middle initial or (if only the first letter of the first name is known) the first initial and the full middle name.
- Changes to department number 999 specified in campcomm.depnupd are ignored.
- The records are matched (on universityid) with records from uid.file (derived from ldif.dump) to pick up the netid.
- The script camplup0.rexx is invoked to format and issue the appropriate Whale alter commands.

### campactv.exec

The campactv.exec script builds a file of active faculty, staff, and students to be used to identify newly-activated people who may need to be issued netids. Its processing:

- The newly-rebuilt campcomm.file is compared on columns 94-101 (affiliation, job function, affiliation group, and status) with campless.tessstat, a hand-maintained file that maps Campus Community status with tesspers status. The non-comment records in campless.tessstat are in the following format:

```

1- 8 Affiliation, job function, affiliation group, and status
    (same format as columns 94-101 of campcomm.file).
10- 12 Tesspers constituency.
14- 15 Precedence for status (see list below).
17- 19 Department number for affiliates.
```

This match determines a precedence for the status:

- : — 00 = active or pending employee (or emeritus faculty)
- : — 01 = active undergraduate, active graduate student, DCC graduate student, ETDCC
- : graduate student, ABSE graduate student, or trustee
- : — 02 = temporarily inactive employee (impending, leave)
- : — 03 = active casual hourly employee
- : — 04 = active affiliate, active departmental computer user, or active outside computer user

- : — 05 = active special student
- : — 06 = retired employee
- : — 07 = inactive undergraduate or graduate student
- : — 08 = separated or deceased staff
- : — 09 = graduated student
- : — 10 = separated casual hourly employee
- : — 11 = inactive affiliate, inactive departmental computer user, inactive outside computer user
- : — 12 = inactive special student
- : — 13 = active kin, active miscellaneous
- : — 14 = deceased student, no-show student
- : — 15 = inactive kin, inactive miscellaneous

It also determines the “tesspers constituency”, which is appended to the record.

- The lowest precedence-number record for each universityid is processed:
  - Records with status other than “A” are ignored.
  - Records for benefits-eligible non-employees (“EMNE” in columns 94-97) are ignored.
  - Records for any other categories than employees (“EM” in columns 94-95), students (“ST” in columns 94-95), and special students (“MS\_\_SL” in columns 94-100) are ignored.
  - The remaining records are written to campactv.file, which lists all active University people.
  - These same records are compared (on universityid) with the previous day’s campactv.file. Any new records are written to campactv.newactiv.
- Any higher precedence-number records are processed:
  - Records with status other than “A” are ignored.
  - Records for benefits-eligible non-employees (“EMNE” in columns 94-97) are ignored.
  - Records for any other categories than employees (“EM” in columns 94-95) and students (“ST” in columns 94-95) are ignored. Note that special students are ignored in this case because they may well be active employees.
  - All remaining records are written to an error file.
- The script campamov.exec checks for the existence of the new campactv.file and terminates processing with an error if the file does not exist.

### **campnadd.exec**

The script campnadd.exec processes the campactv.newactiv file to determine whether there are already active netids for the new people and, if there are not, to select netids for them (which may mean retrieving the netid they had in a previous incarnation). Its processing:

- The campactv.newactiv file is read and the users’ names are massaged to remove all periods and to strip off the suffixes “Jr”, “Sr”, “III”, “IV”, and “V”.
- Department numbers are inserted into the records for undergraduate and special students, “78n” for undergraduates (where “n” is the last digit of the class year) and “790” for special students.
- An extract of recent.userids is prepared:
  - Netids longer than 8 characters are ignored.

- All-numeric netids are ignored.
  - Old-style “qstore numbers” (in the format “Xnnnn”) are ignored.
  - Records with version codes other than “00” are ignored.
  - Obsolete netids (“O” in column 57) are ignored.
- The records from `campactv.newactiv` are compared (by `universityid`) with `uid.file` (derived from `ldif.dump`) to find any netids already active for the new users:
    - *Uid record found:* The `uid.file` record is appended to the `campactv` record. The netid is then looked up in the extract from `recent.userids` with non-active users removed (`recent.userids` records that are not blank in column 57 and records that have anything other than “FZ01” in columns 64-67 are ignored). If the netid is found, this is an active user; no further processing is needed, except for records with a status of “dcu”, which are written to the file `campnadx.existdcu`, which lists existing netids that were originally added for departmental computer users and should now be upgraded from DCU status.
    - *Uid record not found:* The SSN is looked up in the extract from `recent.userids` with non-active users removed (`recent.userids` records that are not blank in column 57 and records that have anything other than “FZ01” in columns 64-67 are ignored). If the netid is found, the record is written to the file `campnadd.fixuaid`, which is a list of new people who need their dummy `universityids` updated in Whale (since the `universityid` should have matched).
  - If no `uid.file` match is found in the preceding step or if a `uid.file` match is found but no match is found in the extract from `recent.userids`, the record is processed further.
  - The department name is extracted from the hand-maintained file `deptall.filecu72` (which lists department numbers, codes, and names), and the department address is extracted from `deptaddr.file`. (The file `deptall.filecu72` has records in three formats. In all cases, the department name begins in column 8. Some records have a three-character department code in columns 1-3. Others have a three-digit department number in columns 1-3. The third type has a seven-digit University account number (Project/Grant number) in columns 1-7; the first three digits are the department number.)
  - The `campnad0.rexx` script is invoked to reformat the record into the following fields (which will be required for adding the user to LDAP and the \$File):

1-	32	name in the format "last, first middle-initial" or "last, first-initial middle"
33-	92	"cn", the name in the format "first middle-initial last" or "first-initial middle last"
93-	112	"givenname", the first token in the first and middle names that is not just an initial or, if only initials are known for the first two names, the two initials
113-	176	"ou", the department name
177-	196	"sn", the last name
197-	205	"ssn", the Social Security number
206-	208	"pustatus", which is derived from the tesspers constituency field:
		EF or ED: fac
		EH: cas
		A_: uNN
		G_: g0N
		SP: sps
		All others: stf
:		Except that a department number of 886 implies a status of "trs" (Trustee) and department number 581 implies a status of "pup" (PU Press).
:		
209-	243	"street", the department's address (which is "Undergraduate student" for departments 78n).
244-	252	universityid
253-	255	"puhomedepartmentnumber", the department number
256		Campus Community transaction status ("A")
257		first initial
258		middle initial
:		
259		"V" if visiting faculty (constituency is ED or EV)
260-	267	birthdate in "yyyymmdd" format
268-	269	hireoffice (the affiliation group)
270-	277	start date of the active status (in "yyyymmdd" format)

- GFDL staff (department 468) are not automatically issued netids. For all others, netid assignment proceeds with these steps:
  1. If the universityid is found in alumuid.file (created from the alumldif.file by the almupdad.exec, which is run weekly) and the netid there is not longer than 8 characters and is not all-numeric, the netid is looked up in the extract of recent.userids to see if the netid has ever been used in the OIT namespace. If the alumuid.file netid has not been used or if it was used by this person (same universityid), that becomes a candidate netid for this person.
  2. If the universityid is found in the extract of recent.userids, the previously-assigned netid becomes a candidate netid for this person.

3. If the universityid is not found in the extract of recent.userids but the SSN is found among the inactive userids (“I” in column 57) in the extract from recent.userids, the previously-assigned netid becomes a candidate netid for this person. (It is assumed that no match was found on the universityid because the one in recent.userids was a dummy netid.)
4. The campnad1.rexx script is invoked to suggest as many candidate netids as can be derived by following these rules (which are applied to the first and last names lower-cased and with apostrophes removed and with portions beginning with a hyphen or blank, if any, removed):
  - a. If the last name is exactly 8 characters in length, use the last name.
  - b. If the last name is no more than 7 characters long, use the first initial followed by the last name.
  - c. If the last name is no more than 7 characters long, use the last name.
  - d. If the last name is longer than 8 characters, use the first initial followed by the first 7 characters of the last name.
  - e. If there is a middle initial, use the first initial, the middle initial, and the last name (truncated to 8 characters).
  - f. If the total length of the first name and the last name is no more than 8, use the first name followed by the last name.
  - g. Use the first name followed by the last initial (truncated to 8 characters).
  - h. If the last name is longer than 8 characters, use the first 8 characters of the last name.
  - i. Use the first, middle, and last initials (where the middle may be missing).
  - j. If the status is “fac”, “stf”, “trs.”, “pup”, “aff”, or “cas”, use the initials followed by “2” through “9”. For others (students), use the initials followed by “two” through “nine”.
5. All candidate netids generated by these processes are looked up in the extract of recent.userids and conflicts are discarded. The candidate netids for students are looked up in the alumuid file and those conflicts are also discarded.
6. The new netids are then winnowed down to one for each person (the first generated by the steps above that has not already been assigned to another new person in this run).
7. The records for which no netid has been derived by these procedures are written to campnadd.noassign to be assigned manually.
8. The records for which a netid has been selected (and the records for GFDL staff) are processed by the campnad5.rexx script, which invokes the script whalgenp.exec to assign a starter password, using the same rules as are used in Whale. (If neither the SSN nor the birthdate are known, whalgenp.exec generates a random password, which will not be communicated to the new user; instead, the user will need to contact the Help Desk to have a password reset done.) campnad5.rexx appends the record with a password-type flag (“S” for SSN, “B” for birthday, or “H” for Help Desk) and the password. The records are then written to campnadd.toadd.

**campnadl.exec**

The campnadl.exec adds incoming faculty, staff, and students to LDAP (with name, address, and department) and formats the Whale add commands to add a .STUDY account for each new person. It then prints the “sealed envelopes” with the password clues for each newly-added .STUDY account. Its processing:

- The campnadd.toadd file (created by campnadd.exec) is read and processed by the campnad2.rexx script as follows:
  - The Whale database (the \$File) is searched for a netid (“qstore”) record for this netid. If one is found and it is not marked inactive, the record is written to an error file. (This should not happen, as any active netids that Whale knows about should have been found in recent.userids.)
  - The LDAP directory is searched for the universityid of the new user. If an entry is found and the uid (netid) is not the same (and is not null), the record is written to an error file. (This also should not happen.) If the uid is the same and the entry is for an Xalias (“ou=Xaliases”), the existing LDAP entry is deleted.
  - If the new person is at GFDL (department 468), a base LDAP entry is added by populating the following attributes in an entry with a distinguished name of the format “cn=**cn**,o=Princeton University,c=US” (the boldface values are taken from the record formatted by campnad0.rexx):

```

datasource: iPlanet Messaging Server 5.0 Admin Console
displayname: cn
givenname: givenname (humans only)
inetuserstatus: active
mailallowedserviceaccess: +all:*
maildeliveryoption: mailbox
mailhost: imap1.Princeton.EDU (for undergraduates)
mailhost: imap2.Princeton.EDU (for all others)
mailuserstatus: active
objectclass: inetLocalMailRecipient
objectclass: inetMailUser
objectclass: inetOrgPerson
objectclass: inetUser
objectclass: ipUser
objectclass: nsManagedPerson
objectclass: nsMessagingServerUser
objectclass: organizationalPerson
objectclass: person
objectclass: puPerson
objectclass: top
objectclass: userPresenceProfile
ou: ou
preferredlanguage: en
puhomedepartmentnumber: puhomedepartmentnumber
pumailstore: IMAPVP1 (for undergraduates)
pumailstore: IMAPVP2 (for all others)
pustatus: pustatus
sn: sn

```

ssn: **ssn**  
 studentstreet: **street** (special students only)  
 street: **street** (not undergrads or special students)  
 universityid: **universityid**

Any records that cannot be added to LDAP are written to an error file.

- There is no further processing for GFDL staff members. For all others, if the netid (“qstore”) was found in the \$File and the entry was marked inactive, Whale commands are issued to update the name, ssn, pustatus, street, universityid, puhomedepartmentnumber, and birthdate. Whale reflects this information to LDAP.
- If an inactive netid was found in the \$File, a Whale command is issued to add the default (.STUDY) account for the user. The command specifies the starter password generated by campnadd.exec. Whale will turn off the inactive flag and remove any netid termination date as part of its processing.

Otherwise, Whale commands are issued to add both the netid and the default account for the user. The input to Whale includes the netid assigned by campnadd.exec, the name, ssn, pustatus, street, universityid, puhomedepartmentnumber, and birthdate fields from the record formatted by campnad0.rexx, and the password generated by campnadd.exec (whalgenp.exec). The telephone number is specified as “NONE”.

In both cases, Whale makes further updates to the LDAP entry. In either case, the record is added to the file campnadd.addeduid.

Note that before campnadd.addeduid is written, records from any existing file with that name are read and added to the new output file. This allows “hand fix-ups” and the regeneration of the sealed envelopes when needed.

- To format the sealed envelopes, the campnadd.addeduid file is read and the records are processed according to the new user’s University position:

: — For Dean of Faculty employees (hireoffice is “DF”) and Human Resources employees  
 : (hireoffice is “HR”), two sealed-envelope records are generated:

: 1. To format the envelope sent to the manager of the new user’s department, the  
 : department number is looked up in the cardofdm.file, which is a table of departmental  
 : managers (department number, manager name, and manager universityid). The  
 : department manager’s universityid is looked up in the “OL1” records from  
 : address.ccfile (from /dmsdrop/campcomd/prod\_files/ldap\_addr.txt) to get the Campus  
 : Community OL1 address. The matching records from cardofdm.file and address.ccfile  
 : are appended to the record being processed. The script campnas0.rexx is invoked to  
 : reformat the record into a semi-colon-delimited string needed to drive the formatting of  
 : the password envelopes. The string contains the following fields:

- a. Sort key (“2” for Campus Mail or “3” for US Mail)
- b. Password type
- c. Envelope type (“Campus Mail”)
- d. Netid
- e. Magic4 (first four characters of starter password)
- f. Name of user

- g. Address line 1 (“c/o” department manager’s name)
  - h. Address line 2 (department name)
  - i. Address line 3 (department manager’s address line 1)
  - j. Address line 4 (department manager’s address line 2)
  - k. Address line 5 (null)
- : 2. To format the envelope to be mailed to the new DoF or HR employee, the new  
 : employee’s universityid is looked up in the “HOME” records from address.ccfile to get  
 the Campus Community HOME address, and the matching record is appended. The  
 campnas1.rexx script formats this combined record into the string needed to drive the  
 formatting of the password envelopes:
- a. Sort key (“2” for Campus Mail or “3” for US Mail)
  - b. Password type
  - c. Envelope type (“US Mail”)
  - d. Netid
  - e. Magic4 (first four characters of starter password)
  - f. Name of user
  - g. Address line 1 (from address.ccfile)
  - h. Address line 2 (from address.ccfile)
  - i. Address line 3 (from address.ccfile)
  - j. Address line 4 (city state zipcode from address.ccfile)
  - k. Address line 5 (country or null if USA)

If the zipcode is 08544, address line 4 is set to null and the envelope type is changed to “Campus Mail”. If any of the address lines are null, subsequent non-null address lines are pushed up so that there are no gaps.

— For PPL employees (hireoffice is “PL”), one sealed-envelope record is generated (by the campnas3.rexx script):

1. Sort key (“1”)
2. Password type
3. Envelope type (“PPL mm/dd/yy” — date is start date)
4. Netid
5. Magic4 (first four characters of starter password)
6. Name of user
7. Address line 1 (“c/o Andrea Moten”)
8. Address line 2 (“Princeton Plasma Physics Laboratory”)
9. Address line 3 (“M33 C-Site B165 PPL”)
10. Address line 4 (null)
11. Address line 5 (null)

— For Princeton University Press employees (hireoffice is “PU”), one sealed-envelope record is generated (by the campnas6.rexx script):

1. Sort key (“2”)
2. Password type
3. Envelope type (“Campus Mail”)
4. Netid
5. Magic4 (first four characters of starter password)
6. Name of user

- 7. Address line 1 (“c/o Patrick Carroll”)
- 8. Address line 2 (“Princeton University Press”)
- 9. Address line 3 (“41 William Street”)
- 10. Address line 4 (null)
- 11. Address line 5 (null)

- For undergraduate students (hireoffice is “UG”), graduate students (hireoffice is “GR”), and special students (hireoffice is “SL”), no sealed-envelope record is generated currently in the overnight runs. It is expected that new students being added on an individual basis will visit the Help Desk to have a password reset done. In bulk adds of new students, the processing is done as part of the annual preparation of the Student Computing Initiative mailing in June. At that time, sealed envelopes are printed for the incoming students with their home address. In addition, incoming graduate students are sent the same information to their email address.

The sealed-envelope records are sorted on the sort key, which is then removed, and the records are processed by the formps.rexx script, which produces a PostScript file suitable to be printed on the sealed-envelope stock. The letters come in four formats:

1. The user already has a password.
2. The starter password uses the birthday.
3. The starter password uses the SSN.
4. The user needs to come to the Help Desk to get a password.

Each one-page letter is customized with the user’s name, netid, magic4, and address, as well as the “envelope type” field. The Postscript file is sent to the printer nslaser2\_t4s using the lpr command, so that OIT Production Control can print, fold, seal, and distribute the envelopes.

- The departments must also be notified when netids are assigned for new faculty and staff. The processing:
  - Records for non-students and non-GFDL personnel with department numbers other than 999 are extracted from campnadd.addeduid and compared (by department number) with the file cardepts.file (a manually maintained file), which lists department chairs, managers, and CARPROC contacts:

Columns	Field name	Field Value
1- 3	Department number	
5- 96	Department chair	universityid, name, address, phone
98-189	Department manager	universityid, name, address, phone
191-282	Associate chair	universityid, name, address, phone
284-375	CARPROC contact	universityid, name, address, phone
377-468	CARPROC contact cc1	universityid, name, address, phone
470-561	CARPROC contact cc2	universityid, name, address, phone

The format of each of these 92-byte fields is:

Columns	Field name
1- 9	Universityid
11- 46	Name
48- 71	Office address
73- 92	Office phone

The CARPROC contact is selected, if there is one for the department; otherwise, the department manager is selected. In some cases, there are also one or two CARPROC “cc contacts”; they are also selected if they exist.

- The departmental contacts’ email addresses are looked up (by universityid) in the file email.filecurr (created by allupdat.exec).
- Email listing the name, netid, and status (including “Visiting” for faculty, if applicable) of the new employees is sent to the department contact(s), with the subject line set to “NetID **netid** assigned for **name**”. The body of the email varies slightly depending on whether the new employee is faculty, University staff, or PPL staff. See the files carproc.memodf, carproc.memohr, and carproc.memopl for current text.
- The new employee is also sent email with the subject “Your Princeton University NetID”. The body of the mail begins “Your Princeton University NetID **netid** has been established. Your”. That is followed by text from carproc.usermemo introducing OIT computing facilities and passwords.
- The meta-data files, recent.userids, uid.file, and email.file, are updated with the newly-assigned netids:
  - The non-GFDL records from campnadd.addeduid are reformatted and merged with recent.userids. If there is already a record with the same netid and version code in recent.userids, the record is replaced with the new record, but the “fiscal year assigned” field is copied from the original record.
  - The non-GFDL records from campnadd.addeduid are reformatted and merged with uid.file. Existing records with the same universityid are replaced.

- The non-GFDL records from `campnadd.addeduid` are reformatted and merged with `email.file`. Existing records with the same `universityid` are replaced.
- The `campnete.exec` and `netesend.exec` scripts are invoked to send a file of new email addresses to Campus Community:
  - The file of all University people, `campcomm.file`, is read and each entry is looked up (by `universityid`) in `uid.file` and `email.file` to retrieve the known `netids` and email addresses.
  - The resulting records are compared (by `universityid`) with the file `ldiupdad.xaliases` (the list of all Xalias LDAP entries created by `ldiupdad.exec`). The people who have Xalias entries are not processed further.
  - The remaining records are reformatted to the format used in `netemail.file`:

Columns	Length	Field Name	Field Value
1-9	9	Universityid	Numeric
10-57	48	Netid	NetID (today will be only 8 characters for people, but will be longer in the future; already longer for non-people). Value of DELETE means remove any existing NetID.
58-128	71	Email address	Full Email address (e.g., <code>scoletti@Princeton.EDU</code> ). Value of DELETE means remove any existing Email address.

All records are written to the new version of `netemail.file`.

- The records are compared (by `universityid`) to the old version of `netemail.file` and new or changed entries are written to `netemail.txt`, which is FTP'd (mode 644) to `dmsdrop` as `/dmsdrop/campcomd/prod_files/netemail.txt`.

#### : **IV. NIGHTLY PROCESSING (`proclddi.exec`)**

- : The driver script `proclddi.exec` is run each morning at 00:15. It repeats the processing done by `proctxtg.exec` up through `delodat1.exec`. This results in adding additional LDAP attributes for
- : people who got added overnight. (Some of the enhanced directory information is not available
- : until this time.)

: **V. NIGHTLY PROCESSING (procfacexec)**

: The driver script procfacexec is run at 07:30 each morning to read in the new Campus  
: Community facebook files built earlier in the morning during the Campus Community overnight  
: processing. (The facebook\_new.txt file is built by about 07:15.)

: **facebgetexec**

: The file /dmsdrop/campcomd/prod\_files/facebook\_new.txt is FTP'ed (in binary mode) as  
: facebook.bin. The script facebge4.exec is run to deblock the file and translate it from codepage  
: 819 to codepage 1047 (from ASCII to EBCDIC). Any records that contain accented characters  
: are written to the file facebacc.file. The accented characters are then translated to the unaccented  
: equivalent and written (with the other records) to facebook.file.

: A facebook.file record contains the following tab-delimited fields:

- : • Emplid
- : • Facebook Flag
- : • Faculty Flag
- : • Academic Officer Flag
- : • Chair Flag
- : • Emeritus Type (F=faculty; A=administrator; L=librarian; N=not emeritus)
- : • Business Unit
- : • Position Nbr
- : • Jobcode
- : • Deptid
- : • Dept Descr
- : • Dept Descrshort
- : • Fb Business Unit
- : • Fb Position Nbr
- : • Fb Jobcode
- : • Fb Deptid
- : • Fb Dept Descr
- : • Fb Dept Descrshort
- : • Joint Deptid
- : • Joint Dept Descr
- : • Joint Dept Descrshort
- : • Sort Last Name
- : • Sort First Name
- : • Sort Middle Name
- : • Primary Last Name
- : • Primary First Name
- : • Primary Middle Name
- : • Primary Name Suffix
- : • Preferred Name Flag
- : • Preferred Last Name
- : • Preferred First Name
- : • Preferred Middle Name
- : • Preferred Name Suffix
- : • Degrees
- : • Titles

- : • Office Locations
- : • Email Addr

**facebook.exec**

: The script facebook.exec reads the facebook.file and processes the records in several ways:

- : 1. Three extracts of the facebook file records are built containing the three-digit department number and 100-character department descriptions (there are three sets in each record, “Deptid” and “Dept Descr”, “Fb Deptid” and “Fb Dept Descr”, and “Joint Deptid” and “Joint Dept Descr”). The three extracts are sorted together and duplicate records are discarded. Any records with the same department number but different descriptions are written to an error file. All records are compared on the department number with the records with three-digit department numbers in the file deptall.filecu72. Records with the same department number but a different description are written to facebook.fixdepts. Records with department numbers that are not found in deptall.filecu72 are written to facebook.adddepts. The file facebook.filecu72 is written as a copy of deptall.filecu72 with any new departments added and any changed descriptions replaced.
- : 2. Records from facebook.file with empty email address fields are compared on universityid with the file uid.file to determine the netid. If no match is found and the emeritus type is not “F”, “A”, or “L”, the record is written to an error file. If a match is found in uid.file, the record is compared on universityid with email.file to determine the email address. If no match is found, the record is written to an error file. If a match is found, a record containing the universityid, the netid, and the email address (in the format for netemail.file) is written to the file netemail.facetxt.
- : 3. The file titles.ccfile is built by extracting the universityid and title from the records in facebook.file.
- : 4. The file degrees.ccfile is built by extracting the universityid and the degrees from the records in facebook.file.
- | 5. The records from facebook.file with numeric department numbers are compared on universityid with the records from homedepartment.file. If no match is found in homedepartment.file, the record from facebook.file is written to an error file. If a match is found but the department number is different, the matching record pair is written to the file facebook.fixhome, a list of department numbers that are wrong in LDAP.

**rescoll.exec**

: The script rescoll.exec builds the file rescoll.ccfile, which contains residential college information for active freshmen and sophomores. The processing:

- : 1. campcomm.file is read and the records for active undergraduates are selected (“ST\_\_UGA” in columns 94-101).
- : 2. Freshmen and sophomores are selected (by the class year in columns 122-123).

- : 3. Records containing the universityid and the name of the residential college are written to  
: rescoll.ccfile. The residential college code in columns 127-129 is translated as follows:
- : • BUT: Butler College
  - : • FOR: Forbes College
  - : • MTY: Mathey College
  - : • ROC: Rockefeller College
  - : • WIL: Wilson College
- : Records that do not contain one of these college codes are ignored.

: **enodupdt.exec**

: The script enodupdt.exec builds the LDIF updates for updating titles and degrees for employees  
: and residential colleges for freshmen and sophomores in the LDAP directory, based on  
: information from the Campus Community titles, degrees, and rescoll ccfiles. (Temporarily, titles  
: are being updated only for ED (visiting faculty), EF (faculty), EL (professional library), EM  
: (monthly staff), ER (professional research), ES (professional systems), ET (professional  
: technical), and EV (visiting fellows), pending a decision about whether hourly staff titles are  
: ready to be displayed.)

: The processing:

- : 1. The file ldiupdad.accounts (which lists all people who have LDAP entries) is compared on  
: universityid with campcomm.file. For each universityid found in both files, a “DELETE”  
: record is written to the files titles.deletes, degrees.deletes, and rescoll.deletes. The purpose of  
: these three files is to delete LDAP information for all people in Campus Community unless an  
: override is found in another file (see below).
- : 2. The file titles.ccfile is read and records with blank title fields are discarded. The remaining  
: records are compared on universityid with the current-SSN employee records from  
: tesspers.file (temporarily not including the employee categories, EA, EB, EC, EE, EH, EO,  
: EW, EX, EY, and EZ). Records for which no match is found are written to an error file.
- : 3. Records for which a match is found in tesspers.file are compared on universityid with the  
: titles.deletes file built earlier to find universityids that occur in titles.deletes that do not occur  
: in the records derived from titles.ccfile. Those records are added to the records from  
: titles.ccfile.
- : 4. The records passed from the previous step are compared on universityid with the file  
: ldiupdad.accounts (all active LDAP entries). Records that have a match in ldiupdad.accounts  
: are passed to the next step.
- : 5. These records are compared on universityid with the file titles.file, which contains the titles  
: currently in LDAP. The passed records that have titles that are different from those in  
: titles.file are written to enodupdt.titlrepl with the old title appended to the record. The passed  
: records that match a record from titles.file but specify “DELETE” are written to  
: enodupdt.titledel with the old title appended to the record. The passed records that do not  
: match a record from titles.file and that have a title other than “DELETE” are written to  
: enodupdt.titladd.

- 
- : 6. The records from degrees.ccfile are processed in the same way (comparing against  
: pudegree.file from the current LDAP directory) to produce the files enodupdt.degrepl,  
: enodupdt.degradd, and enodupdt.degrdel.
  
  - : 7. The records from rescoll.ccfile are processed in the same way (comparing against  
: purescol.file) to produce the files enodupdt.rescrepl, enodupdt.rescadd, and enodupdt.rescdel.
  
  - : 8. The add, delete, and replace files created in the previous steps are compared on universityid  
: with ldiupdad.accounts to build enodupdt.dns, a list of all distinguished names that need  
: updating.
  
  - : 9. The file of LDIF updates is built:
    - : a. The file enodupdt.dns is read and reformatted so that each output record consists of the  
: universityid, the numeral "0", a centsign, the DN, a centsign, and "changetype: modify".
  
    - : b. The file enodupdt.titladd is read and reformatted so that each output record consists of the  
: universityid, the numeral "1", "add: title", a centsign, "title: ", the new title, a centsign, and  
: a hyphen.
  
    - : c. The file enodupdt.titldel is read and reformatted so that each output record consists of the  
: universityid, the numeral "1", "delete: title", a centsign, "title: ", the old title, a centsign,  
: and a hyphen.
  
    - : d. The file enodupdt.titlrepl is read and reformatted so that each output record consists of the  
: universityid, the numeral "1", "delete: title", a centsign, "title: ", the old title, a centsign, a  
: hyphen, "add: title", a centsign, "title: ", the new title, a centsign, and a hyphen.
  
    - : e. The files enodupdt.degradd, enodupdt.degrdel, and enodupdt.degrepl are similarly  
: processed to modify the pudegrees attribute.
  
    - : f. The files enodupdt.rescadd, enodupdt.rescdel, and enodupdt.rescrepl are similarly  
: processed to modify the purescollege attribute.
  
  - : 10. All of the records are collected together and sorted on the universityid and type code (in  
: column 10). The universityid and type code are removed and the records are split at the  
: centsigns. (At this point, the DN record for each user is before the other records for that user.)  
: The last hyphen record for each user is deleted.
  
  - : 11. The records are written to the file ldif.update. A linefeed character is appended to each  
: record and the records are translated to from codepage 1047 to codepage 819 (from EBCDIC  
: to ASCII). The resulting file is FTP'd (in binary mode) to directory.Princeton.EDU as  
: /tmp/update.ldif.

## VI. WEEKLY PROCESSING OF ALUMNI DATA

The alumni LDAP directory is updated automatically once a week. The alumni directory contains entries for all alumni/ae and alumni organizations recognized by the University. Currently enrolled undergraduate students are included in the alumni LDAP directory. They are marked as current students using the “pualumcurrstu: yes” attribute, however, so that various alumni services (such as email service) may be denied to them until they are actually alumni.

The weekly update of the alumni LDAP directory is driven by a file from Advance that is produced every Thursday evening. The script `procalmg.exec` runs at 21:15 on Thursdays. The file `/dmsdrop/advora/advp/alumldap.dat` is retrieved from `dmsdrop` using FTP and is stored as `alumdat.file`. `alumdat.file` contains directory information for all alumni/ae and alumni organizations that should be included in the alumni directory. The following scripts are used to update the existing alumni directory from `alumdat.file`:

### **alumget.exec**

`/var/local/ns-home/slapd41/slapd-alumni-ldap1/ldif/dumper.ldif`, the LDIF dump of the alumni LDAP directory, is FTP'd from `alumni-ldap1.Princeton.EDU` as `alumldif.dump`.

### **almupdad.exec**

The `almupdad.exec` script expands the dump of the alumni LDAP directory into several files and generates dummy `pualumids` for any new distinguished names added directly into the alumni LDAP directory.

The records for each entity in `alumldif.dump` are joined to form a single record and each record is processed:

- If an entry has a `pualumid` attribute, the `pualumid` is prefixed to the record containing all attributes. If an entry has no `pualumid` attribute, it may be necessary to assign a dummy `pualumid`. The file `almupdad.almidall` is searched, matching on the distinguished name:
  1. If a match is found, the previously assigned dummy `pualumid` is prefixed to the record.
  2. If no match is found, the entry must be one added by the alumni LDAP server itself; the script `almupda0.rexx` is invoked to assign a dummy `pualumid` (in the range beginning `0040000000`), which is prefixed to the record. (Note that `pualumids` are 10 digits.)

(`almupdad.almidall` is an accumulation of the records generated in step 2. above.)

- The entries are sorted into `pualumid` order and passed on to the next step.
- The full records are reformatted so that there is one record per attribute containing just the `pualumid`, the attribute name, and the attribute value. These records are written to the file `almupdad.file`.
- Entries with `pualumids` starting “00400” are not processed further.
- The records are reformatted so that there is one record per attribute containing just the `pualumid` and the attribute value for only the attributes of interest. These records are processed as follows:

- cn: The records are written to the file alumcn.file.
- dn: The distinguished names are written to one of three files, depending on the entry type:
  1. almupdad.xaliases (dn contains “ou=Xaliases”);
  2. almupdad.lists (dn contains “ou=Lists”);
  3. almupdad.accounts (all others).
- givenname: The records are written to the file alumgn.file.
- pualumcurrstu: The records are written to the file alumcstu.file.
- pualumpwhint: The records are written to the file alumbhint.file.
- pualumregpin: The records are written to the file alumpins.file.
- sn: The records are written to the file alumsn.file.
- uid: The records are written to the file alumuid.file.
- universityid: The records are written to the file alumuaid.file.
- userpassword: The records are written to the file alumpass.file.

**almupdal.exec**

The script almupdal.exec builds an LDIF file from the alumdat.file written by the Advance alumni overnight process.

The format of the records in alumdat.file:

Columns	Length	Field Name	Field Value
1-10	10	id_number	Numeric, Advance Alumni ID number
11-35	25	first_name	First name
36-60	25	last_name	Last name
61-62	2	primary_affil	Primary affiliation (A=undergraduate, G or GS=graduate student, D=honorary, S=spouse, W=widow)
63-66	4	pref_class_year	Preferred class year
67-126	60	formatted_name	Full name, with prefix and any generational suffix
127-150	24	inst_suffix	Institutional suffix (symbols, e.g., '46 h52)
151-160	10	pin	PIN number
161	1	person or org	Person or Organization indicator ("P" or "O")

The steps in almudal's processing:

- The files alumdat.file and alumdat.testuser are read and processed together. (The alumdat.testuser file contains records for a few test users needed for BlackBoard testing.)
- : Records for any alumni/ae who have requested not to be listed in the alumni directory are removed. (The only one currently is 0000449366.)
- The formatted\_name field is reformatted to reduce multiple blanks to a single blank.
- : • The records are compared (on pualumid) with the hand-maintained file alumni.tesspers, which lists alumni who also have universityids. If a match is found, the universityid is appended to the record.

- The records are reformatted to build the LDIF for an alumni LDAP entry with the following attributes:

```

dn: pualumid=pualumid,dc=alumni,dc=princeton,dc=edu
givenname: first_name
objectclass: inetOrgPerson
objectclass: organizationalPerson
objectclass: person
objectclass: puPerson
objectclass: top
palumregpin: pin
cn: formatted_name inst_suffix
universityid: universityid                (optional)
palumcurrstu: yes                          (current undergraduates only)
sn: last_name                             (entries for people)
sn: formatted_name                        (entries for organizations)

```

The records are written to the file `almupdal.file`. (Each record is preceded by the 10-digit `pualumid`; no “`pualumid:`” line is generated, because that is built automatically from the distinguished name when the LDIF is loaded.)

### **almupdat.exec**

The script `almupdat.exec` compares `alumdal.file`, the LDIF file built from the Advance file in the preceding step, with the LDIF dump of the current alumni LDAP directory and generates updates to be applied to the alumni LDAP directory. This update process is authoritative for all alumni (people and organizations), so it will add or delete those entries as needed. However, the LDAP directory is authoritative on a number of server and LDAP administrative entries which must be left unmodified by this process. In addition, the LDAP server is authoritative on some fields in entries for alumni people and organizations, so those fields must also be left unmodified by this update process. The only attributes modified in this process are those formatted in the previous step.

The processing:

- The old LDIF dump is reformatted to match the new LDIF file. (That is, entries for dummy `pualumids` are removed, all but the selected attributes are removed, and the remaining records are prefixed by the `pualumid`.)
- Both the old and new files are split into two subfiles, one containing the distinguished names and the other containing all other attributes of interest.
- The two files of distinguished names are compared (on `pualumid`):
  - *A `pualumid` occurs in the new file but not in the old file:* The record is written to the file of entries that need to be added, `almupdat.dntoadd`.
  - *A `pualumid` occurs in the old file but not in the new file:* The record is written to the file of entries that need to be deleted, `almupdat.todelete`.

- 
- *The pualumid occurs in both files:* A single record from each pair is written to the file of entries that need to be retained, `almupdat.dntokeep`.
  - The file of other attributes from the old dump is examined to discard records for LDAP entries listed in `almupdat.todelete` (which will be deleted in their entirety). The remaining records are passed to the next step.
  - The passed records are compared on `pualumid` with the file of other attributes from the new dump, and two output files are written, `almupdat.toadd` (which contains attributes to be added) and `almupdat.todelete` (which contains attributes to be deleted). These two files contain all attributes other than `dn` for new entries. They also contain the before and after values for attributes that are being changed in existing entries.
  - The LDIF updates file, `alumldif.updates`, is generated as follows:
    - *Records from `almupdat.dntodel`:* For each record, three records are written to the output file:
      1. A null record
      2. The distinguished name
      3. “changetype: delete”
    - *Records from `almupdat.dntoadd`:* For each new distinguished name, these records are written to the output file:
      1. A null record
      2. The distinguished name
      3. “changetype: add”
      4. The attribute records from `almupdat.toadd` for this `pualumid`; *i.e.*, the attribute records formatted by `almupdal.exec`
    - *Records from `almupdat.dntokeep`:* The file of distinguished names that are being retained is compared by `pualumid` with the files of attributes being added or deleted. If any changed attributes are found, the following records are written to the output file:
      1. A null record
      2. The distinguished name
      3. “changetype: modify”
      4. “delete:” `attributename`
      5. `attributename: oldattributevalue`
      6. “-”
      7. “add:” `attributename`
      8. `attributename: newattributevalue`
- Additions or deletions of attributes are handled similarly; more than one attribute may be changed.

### **alumsend.exec**

The alumsend.exec script sends the alumldif.updates file to the alumni LDAP server for updating the alumni LDAP database. The processing:

- The alumldif.updates file is read; the leading null line is removed; trailing blanks are removed from the records; and carriage return and linefeed characters are suffixed to the records, which are then translated from codepage 1047 to codepage 819 (from EBCDIC to ASCII) and written to the file alumsend.ascii.
- alumsend.ascii is FTP'd (in binary mode) to alumni-ldap1.Princeton.EDU as /tmp/update.ldif. A daemon on alumni-ldap1 checks once a minute for the existence of this file. When the file is found, it is moved aside and then loaded into the alumni LDAP directory.

## **VII. MAINTAINING AND VALIDATING THE META-DATA**

: Procedures to update a number of the meta-data files are run every morning at 03:30. (The driver  
: scripts are proctcc.exec and procdaly.exec.) The following scripts are run to update the files and  
: to note errors and discrepancies:

### **getfiles.exec**

The script getfiles.exec is invoked with the arguments “submit noacc” and “receive noacc” to run several MVS batch jobs and receive the output:

- pundicta.jcl produces allcurr.dictouta, a flattened copy of the \$File.
- gradsreg.jcl2000 runs against the Student Records database to extract gradsreg.file, which lists all currently registered graduate students. (This file is used primarily as a reference.)
- gsaddrph.jcl2000 and ugaddrph.jcl2000 run against the Student Records database to produce gsaddrph.file and ugaddrph.file, which list the addresses and telephone numbers of enrolled students from the Registrar’s point of view. (These two files are used primarily as references; they can be useful when there is confusion in other data.)
- studtr.jcl2000 runs against the Student Records database to extract studwork.file, which is then passed through the script studdept.exec to build studtr.file:
  - The Registrar’s two-digit department number in columns 66-67 of studwork.file is looked up in studtr.deptfile (a manually-maintained file), and the Controller’s 3-digit department number and 3-character department code are inserted into the record before the General Exam date.
  - If no department was specified in studwork.file, the residential college code in column 60 is looked up in studtr.collfile (a manually-maintained file), and the 3-digit residential college department number and the 3-character college code are inserted into the record.

The records in studtr.file are in the form:

```

1- 1 Status
    " " Active
    "A" Absent but degree candidate (grad)
    "C" Completed work (ug)
    "D" Deceased
    "E" Expelled
    "F" Failed to qualify (ug)
    "G" Graduated
    "H" Degree candidacy continues
    "L" Leave of absence
    "P" Pre-professional off-year (ug)
    "R" Required withdrawal (ug)
    "S" Suspended
    "T" Withdrawal
    "V" Voluntary withdrawal (ug)
    "W" Enrollment terminated (grad)
    "Z" Enrollment revoked
    "5" Five-year program off-year
    "x" Do not keep accounts active

2- 10 SSN
11- 19 Universityid
20- 21 Undergraduate class year (nn) or graduate year (Gn)
      or special student code (SP)
22- 40 Last name
41- 55 First name
56- 56 Middle initial
57- 57 Gender
58- 59 Undergraduate class year at matriculation
60- 60 Residential college code, additional status
61- 63 Highest graduate degree awarded
64- 65 Graduate cohort year (e.g. 04 for graduate students
      who matriculated between September 2003 and June 2004)
66- 67 Registrar's department number
68- 70 Controller's department number
71- 73 Department code
74- 74 unused
75- 80 General Exam date (MMDDYY)

```

The status fields listed above are those in the data from the Registrar, except that the category "x" is defined by CARPROC and maintained manually. It is used for special purposes when a student who would otherwise be entitled to computer accounts is not to be allowed computer access, as, for example, by the terms of his suspension or because the accounts of a student on leave have been compromised and we are unable to contact him.

The script phoncopy.exec is invoked to copy deptdir.direct and accent.direct from the MVS volume ADP204. These two files are maintained manually by the Telephone Office and Lee

Varian. The deptdir.direct is the department portion of the campus telephone directory. The accent.direct file is a list of people with accented names, expressed in TeX notation for use by the campus directory printing program.

The script tmspget.exec FTP's /dmsdrop/malltele/shared/tmsprd/DIRECTORY.LST, the Telephone Management System directory dump, from dmsdrop as tmsphone.direct.

#### **bldstudy.exec**

The PL/I program dirup1st is run to extract the file oldstudy.output from the \$File. oldstudy.output is another file used primarily as a reference. It lists all known default computing accounts with the user's name, status, flags indicating the state of the netid, and the default subaccount.

: **campmult.exec**

: The script campmult.exec examines campcomm.file for entries that have the same universityid,  
: SSN, and PeopleSoft status (columns 1-18 and 94-100) and writes the duplicate records to an  
: error file.

#### **currchek.exec**

: The script currchek.exec compares the uaidcurr.file and the campcomm.file to find any  
: universityids that are marked old in uaidcurr but current in campcomm or any universityids that  
: are marked current in uaidcurr but are not found in campcomm. It then does the same two  
: comparisons on the SSNs in campcomm.file and the old and current SSNs in sscurr.file. Any  
: records that meet these tests indicate a serious data problem.

#### **campretc.exec**

The script campretc.exec checks the campretc.file, which is maintained by hand. campretc.file lists Campus Community entries for retirees who are continuing to work as Casual Hourlies. It contains pairs of employee records, one Number 0 (retiree) and one Number 1 (Casual Hourly). The Number 1 records are looked up in campcomm.file to verify that the person is still an active Casual Hourly. The records for those who aren't are written to an error file. (Employee Records Number 1 are not seen in the LDAP transactions from Campus Community.)

: **campgone.exec**

: The script campgone.exec checks on the current status in campcomm.file of people who are still  
: showing as active in Campus Community, but who have actually gone. They may be being paid  
: for a while in a termination situation and thus still show as active in Campus Community, but  
: their netids should remain inactive.

: campgone.exec checks the campgone.file, which is maintained by hand. campgone.file contains  
: pairs of employee records, one with active status and one with terminated status. The active  
: records are looked up in campcomm.file to verify that the person is still marked active. The  
: records for those who aren't are written to an error file.

### **campaffs.exec**

The script `campaffs.exec` sets the “department number” in `campcomm.file` for any affiliates who are recognized for inclusion in LDAP:

- The `campcomm.file` is read and compared on the status fields (columns 94-101 vs. columns 1-8) with the affiliate entries (non-blank in columns 17-19) in `campless.tessstat`. If no match is found, the original record is copied to the output file, the revised `campcomm.file`.
- If a match is found:
  - *If the department number in the campcomm record is blank:* The department number is copied from `campless.tessstat` and the record is written to the output file.
  - *If the department numbers in campcomm and campless.tessstat match:* The record is copied unchanged to the output file.
  - *If the department numbers do not match but the campless.tessstat department number is “999”:* The record is written unchanged to the output file.
  - *All others:* The record is written to an error file to indicate that the department number needs to be determined manually, and the record is copied unchanged to the revised `campcomm.file`.

### **campstud.exec**

The script `campstud.exec` updates the `studtr.file` from the `campcomm.file` to fill in any missing names or universityids that have recently been assigned in Campus Community:

- The `studtr.file` is read, and records that already have universityids are copied directly to the output, the revised `studtr.file`.
- Records with blank universityid fields are compared (by SSN) with the `campcomm.file`. Records that are not found in `campcomm.file` are written to an error file, `studtr.misscomm`, to be examined.
- If a matching `campcomm` record is found, the person’s name is copied to the `studtr.file` record if that has a blank name field.
- The universityid is copied from the `campcomm` record to the `studtr` record, which is written to the revised `studtr.file`.

### **studtrfx.exec**

The script `studtrfx.exec` repairs the `studtr.file` from three manually-maintained files, `studtr.fixadd`, `studtr.fixdel`, and `studtr.fixrep`, and checks that `studtr.fixadd` records still reflect student entries in the `campcomm.file`. The processing:

- The `studtr.fixadd` file (which contains both test users and corrections for out-of-sync databases) is read and compared (on universityid) with student and special student records

- 
- : from the campcomm.file. Records that are not matched are compared with the alumni records
  - : from campcomm.file. Records that are still not matched in campcomm.file are written to an error file.
  - The matching records for students and special students are compared to check that both are active or both are inactive (active studtr.fixadd records have a blank in column 1; active campcomm records have “A” in column 101). The matching records for alumni are compared
  - : to check that the studtr.fixadd records are for graduated students (“G” in column 1). Mismatched records are written to an error file.
  - studtr.fixadd is compared (on SSN) with studtr.file. Any entries in studtr.fixadd that exist in studtr.file are written to an error file. Records from studtr.fixadd that do not match with studtr.file on SSN are compared with studtr.file on the universityid. Any matching entries are written to an error file. The non-matched entries and all of the records from studtr.file are passed to the next step.
  - The “before and after” pairs of records from studtr.fixrep are combined into single records and compared (on the before portion) with the passed records. If no matching record is found among the passed records, the fixrep record is written to an error file. If a match is found, the after portion of the fixrep record is passed to the next step along with the non-matched records from studtr.file.
  - The passed records are compared (on the full record) with the file studtr.fixdel. Records from the fixdel file that do not match one of the passed records are written to an error file. The passed records that match a fixdel record are deleted.
  - If any of the remaining records have blank universityid fields, they are written to an error file.
  - : • If any records have dummy SSNs starting “99SP”, they are copied to an error file and then
  - : corrected by changing them to start “9900”. All records are passed to the next step.
  - If any of the remaining records have the same SSN, they are written to an error file.
  - The remaining records are written to the output file, the corrected studtr.file.

### **studsyntax.exec**

The script studsyntax.exec synchronizes the universityids and SSNs in studtr.file with those in campcomm.file:

- The studtr.file is read and the universityid in each record is looked up in the “old universityid” field of uaidcurr.file. If a match is found, the “new universityid” is substituted in the studtr.file record.
- The SSNs are then looked up in the “old SSN” field of ssncurr.file. If a match is found, the “new SSN” is substituted in the studtr.file record.
- The records are written to the revised studtr.file.
- The studtr.file records are compared with campcomm.file:

— *Both the universityid and the SSN match:* No further processing is required.

— *The universityid alone matches:*

- If Campus Community has a real SSN or if both Campus Community and Student Records have dummy SSNs (“999nnnnnn”), an ssncurr-format record is built with the Student Records SSN as the old SSN and the Campus Community SSN as the new SSN.
- If Campus Community has a dummy SSN and Student Records has a real SSN, an ssncur-format record is built with the Campus Community SSN as the old SSN and the Student Records SSN as the new SSN.
- The records are written to ssncurr.newrecs.

— *The SSN alone matches:* A uaidcurr-format record is built with the Student Records universityid as the old universityid and the Campus Community universityid as the new one. The records are written to uaidcurr.newrecs.

— *Neither the universityid nor the SSN match:* The unmatched studtr.file records are written to studsync.missing. In addition, the records for special students are reformatted to be campcomm records and are written to campcomm.newrecs. The records for undergraduate and graduate students are reformatted to be Campus Community transaction records and are written to ldaptxt.newrecs.

### **gsstatus.exec**

The script gsstatus.exec examines the graduate student records in the studtr.file and flags those who are still considered as actively pursuing a degree and who should still be receiving computer accounts (according to Dean Redman of the Graduate School). It also flags those undergraduate and graduate students who have graduated during the year, but who will not formally receive their degrees until next June. June 7 is used as the end of the graduate year (as it should always be after graduation).

For CARPROC’s purposes, the Registrar’s status codes can be divided into three categories:

1. *Terminal* (student is permanently gone): Status “D”, “E”, “G”, “T”, “W”, ”x”, or ”Z”.
2. *Inactive* (student may return but does not get campus privileges for now): Status “C”, “F”, “R”, “S”, or ”V”. Graduate students who have continued in active status beyond G10 are changed to inactive status. Undergraduate students in inactive status are listed in the campus directory while their cohort is at Princeton, with the indication “Not currently enrolled”; this maintains the validity of their “@Princeton.EDU” address for forwarding purposes.
3. *Active* (student does get campus privileges even though he/she may not be enrolled or physically on campus at present): Status “\_”, “A”, “H”, “L”, “P”, or “5”. Students in active status are listed in the campus directory.

Netids and Dormnet IP addresses are kept reserved for students in active or inactive status but are released for terminal status. Computer accounts continue for students who are in active or inactive status.

The studtr.file is read and the records are processed according to the student’s status:

- *Graduate Students:* If the Generals date is in May, it is changed to June 7, to guarantee that it is after graduation. The records are processed further based on the studtr.file status field:
  - Blank (active), “A” (absent), “H” (degree candidacy continues), or “L” (on leave):
    - If the record is missing the graduate student cohort date, it is written to an error file.
    - Remaining records are passed through the script gsstatu0.rexx, which determines whether the graduate student is within five years of matriculation (if Generals are still pending) or within five years of having passed Generals. The current date is compared with an ending date that is either June 8 of the year that is four greater than the cohort year or five greater than the Generals date, if there is a Generals date:
      - If the ending date has passed and the status in column 1 is either blank or “H”, the record is written to the file studtr.degreeerr, which lists graduate students who are still shown as enrolled but who would otherwise be classed as no longer actively pursuing a degree. The records are also marked with an “a” in column 60, indicating that they are actively pursuing a degree, and are written to the output file, the revised studtr.file.
      - If the ending date has passed and the status is other than blank or “H”, the records are copied unchanged to the revised studtr.file.
      - If the ending date has not passed, the records are marked with an “a” in column 60 and are written to the revised studtr.file.
  - “G” (graduated): The records are compared (on universityid) with a studtr.file from after last June’s graduations. If the graduated student was not graduated as of then, the “recently graduated” flag (“g”) is set in column 60 of the record. All records are written to the revised studtr.file.
  - Other statuses: The record is copied unchanged to the revised studtr.file.
- *Special Students:* All records are copied unchanged to the revised studtr.file.
- *Undergraduates:* The records are compared (on universityid) with a studtr.file from after last June’s graduations. If the graduated student was not graduated as of then, the “recently graduated” flag (“g”) is set in column 60 of the record. All records are written to the revised studtr.file.

### **campmess.exec**

The campmess.exec builds the tesspers.file from the campcomm.file and the ssncurr.file. campmess.exec also uses the studtr.file to update a working copy of the campcomm.file with status information, in particular to distinguish between temporary and permanent completion of studies, prior to building the new tesspers.file. In the course of its processing, the script examines the data for many kinds of errors. The processing steps:

- : 1. The records from campcomm.file are compared (on the universityid, SSN, and name) with the “before” records from campcomm.fixname (a manually-maintained file) to find preferred names for people who have requested this. (These preferred names may or may not be reflected as Campus Community preferred names.) If a match is found, the name from the “after” record from campcomm.fixname is substituted in the record from campcomm.file. If an entry in campcomm.fixname is not matched, it is written to an error file.

2. The campcomm.file is examined for duplicate records:

- The records are sorted on the combination of universityid and SSN. If there are duplicates, the second and subsequent records of each set are passed unchanged to the next step.
- The remaining records are sorted on SSN. Records for SSNs that occur only once are passed unchanged to the output file.
- Records for SSNs that occur more than once are written to an error file. Records that are likely to be in error are written to an error file and not processed further. (These are records with any of the following values in columns 94-100: “MS\_\_L1”, “MS\_\_MS”, “MS\_\_FR”, “KN\_\_MS”, “MS\_\_US”, “MS\_\_CO”, “MS\_\_CP”, or “MS\_\_IT”.)
- If, after that, a single record for the SSN remains, it is passed unchanged to the next step. If multiple records remain, they are written to an error file and passed unchanged to the next step.

3. The campcomm.file is validated against the studtr.file:

- The records from studtr.file are prefixed with “G”, “S”, or “U”, depending on the type of student. The first two characters are then looked up in campmess.studstat (a manually-maintained file), to determine the Campus Community status equivalent to the Student Records status. The 8-character status (corresponding to campcomm.file columns 94-101) is appended to the studtr record. A sort key is also appended, consisting of the universityid and 7 characters of the affiliation and the affiliation group.
- An equivalent sort key is appended to the campcomm records passed from the previous step. If there is more than one record with the same key, the second and subsequent records are passed unchanged to the next step.
- The first (or only) record for each sort key is compared (on the 16-character sort key) with the records derived from the studtr.file:
  - *Matching universityid and (7-character) status:* If the 8-character student status in the matching records is the same, the original campcomm record is passed unchanged to the next step. If the status differs, the campcomm record is updated with the campcomm-style status that was earlier appended to the studtr record, and the updated campcomm record is passed to the next step.
  - *Non-matches:* The unmatched campcomm records are sorted on the universityid. If there are any duplicate universityids, the second and subsequent records for a universityid are written to an error file and passed unchanged to the next step. The first (or only) unmatched campcomm record is compared (on the universityid alone) with the unmatched studtr records:
    - *Matching universityid but non-matching status:* These records represent cases in which the same universityid occurs in Campus Community and Student Records, but the status of the person is different in the two files. These need to be analyzed further:
      - If the studtr status is “SV”, the record is for a withdrawn special student who has some other University status. It is passed unchanged to the next step.

- 
- If the campcomm record has an affiliation of “ST”, the record is for a person known in Campus Community as a student, but who needs to have the affiliation modified to “graduated”. The 8-character status that was appended to the studtr record is copied to columns 94-101 of the campcomm record, which is then passed to the next step.
  - If the campcomm record has a status of “MS\_\_SL”, the record is for a special student who needs a status update. The 8-character status that was appended to the studtr record is copied to columns 94-101 of the campcomm record, which is then passed to the next step.
  - All other records are written to an error file and then processed further. In each case, the original campcomm record is passed to the next step. In addition, a new record is created with the new campcomm-style status that was earlier appended to the studtr record. These records are also passed to the next step.
- *Unmatched studtr records*: Any studtr records that are still unmatched (representing students who have a record in Student Records but not in Campus Community) are written to an error file.
  - *Unmatched campcomm records*: Any campcomm records that are still unmatched in Student Records are written to an error file and passed unchanged to the next step.
4. Another pass is made to validate the campcomm records passed from the previous step against the studtr.file. The processing depends upon the student status:
- *Undergraduates*: The passed campcomm records with status “ST\_\_UGA” are compared on universityid with the undergraduate records from studtr.file:
    - *Studtr record matches campcomm record*: If the status field in column 1 of the studtr record is a blank, the campcomm record is passed unchanged to the next step. Otherwise, the record is written to an error file. Its status field (column 101) is changed to “I” (for “inactive”) and it is passed to the next step.
    - *No studtr record matches campcomm record*: The unmatched campcomm record is written to an error file. Its status field (column 101) is changed to “N” (for “no-show”) and it is passed to the next step.
  - *Graduate Students*: The passed campcomm records with status “ST\_\_GRA” are compared on universityid with the graduate student records from studtr.file:
    - *Studtr record matches campcomm record*: If the status field in column 1 of the studtr record is a blank, the campcomm record is passed unchanged to the next step. Otherwise, the record is written to an error file. Its status field (column 101) is changed to “I” and it is passed to the next step.
    - *No studtr record matches campcomm record*: The unmatched campcomm record is written to an error file and is passed unchanged to the next step. (Thus, Campus Community is treated as authoritative on newly added or visiting graduate students.) In addition, the alphabetic department code is looked up in studtr.deptfile to retrieve the two-digit department code used by Student Records. A studtr-format record is built and written (with an appropriate comment record) to studtr.gstoadd, which lists “no-show”

graduate students, who may in fact be newly-accepted graduate students who should be added to studtr.file.

- *Special Students:* The passed campcomm records with status “MS\_\_SLA” are compared on universityid with the special student records from studtr.file:

— *Studtr record matches campcomm record:* If the status field in column 1 of the studtr record is a blank, the campcomm record is passed unchanged to the next step. Otherwise, the record is written to an error file. Its status field (column 101) is changed to “I” and it is passed to the next step.

:  
:  
— *No studtr record matches campcomm record:* The unmatched campcomm record is written to an error file and it is passed unchanged to the next step. (Thus, Campus Community is treated as authoritative for new special students.) In addition, a studtr-format record is built and written (with an appropriate comment record) to studtr.sptoadd, which lists missing special students, who may in fact be newly-accepted special students who should be added to studtr.file.

- *Non-students:* All other records are passed unchanged to the next step.

5. The passed records are now ready to be reformatted to build the tesspers.file:

- The 8-character status field in columns 94-101 is then looked up in campess.tessstat to determine the equivalent status for formatting a tesspers-style record, as well as the precedence of the status.
- The records are rearranged into tesspers format, with periods removed from the names.
- The records are sorted on universityid and the precedence of the status. If more than one record is found for a given universityid, those with the higher (worse) precedence are written to an error file.
- The records are compared on universityid with the sscurr.file. If any matches are found, an additional tesspers record is written for each of the “old” SSNs.
- Any records with all-zeros SSNs (starting in either column 10 or column 100) are written to an error file.
- The remaining records are sorted on the combination of the universityid and the SSN, and the first record for each sort key is written to tesspers.file. Any additional records are written to an error file.
- The records that were written to tesspers.file are also sorted on the SSN in column 10. If there are any duplicates, the records are written to an error file.

**tesleave.exec**

The script tesleave.exec builds the current tesspers.leavedat file of non-faculty employees on leave and then uses that file to update the leave dates in tesspers.file and to set the status to “S” (separated) in tesspers.file for employees who have been on leave for longer than two years (*i.e.*, long-term leave). The processing:

- The existing tesspers.leavedat file is compared (on the universityid) with the “current SSN” entries from tesspers.file:

- *No match in tesspers.file:* The record is copied to an error file, but it is also passed to the next step.

- *Match found in tesspers.file:*

- The SSN is copied (to both the old and new SSN fields) from the tesspers.file record.
- The tesspers.file is read afresh and is extracted to narrow it down to current SSN records for non-faculty employees who are on leave (columns 10-18 the same as columns 100-108, “E” in column 79, not “F” in column 80, and “L” in column 81). Today’s date (YYMMDD) is stored (provisionally) as the leave date (column 90).
- This extract from tesspers.file is compared (on universityid) with the tesspers.leavedat records that matched tesspers.file:
  - *Person is in both files (continues on leave):* The original leave date is copied from column 90 of the leavedat record to column 90 of the record from tesspers.file, and the revised record is passed to the next step.
  - *Person is not in old tesspers.leavedat (has just gone on leave):* The record is written to tesleave.newleave and passed to the next step.
  - *Person is not marked on leave in tesspers.file (has just come off of leave):* The record is written to tesleave.oldleave and not processed further.

- The passed records are written to the output file, the revised tesspers.leavedat.

- The records from the new tesspers.leavedat are processed further to determine whether the person has been on leave for more than a year. The leave date is converted to days since the epoch and is compared to today’s date in days since the epoch. If the difference is not positive, the record is written to an error file, but is also passed to the next step. If the difference is greater than 731 days, the record is updated to mark the person separated (“S” in column 81); the record is written to tesspers.leavesep and is also passed to the next step. If the difference is not greater than 731 days, the record is passed unchanged to the next step.

- The tesspers.file is read afresh and is compared on universityid with the records passed from the previous step:

- *tesspers entry found in passed records:* The status field (column 80) and the leave date (columns 90-95) are copied from the passed record to the tesspers record, which is written to the output file, the revised tesspers.file.

- *tesspers entry not found in passed records*: The tesspers record is copied unchanged to the revised tesspers.file.
- *Passed record not found in tesspers*: The records is written to an error file.

### **tesfixup.exec**

The script tesfixup.exec corrects the tesspers.file from the manually-maintained files tesspers.fixadd (which lists records that need to be added), tesspers.fixdel (which lists records that need to be deleted), tesspers.fixrep (which lists records that need to be replaced), tesspers.fixnames (which lists records with names that need to be fixed), tesspers.regfxeme (which lists emeritus faculty whose records need to be fixed), and tesspers.regfxnoe (which lists retired faculty who are not emeriti but whose records need to be fixed). The processing:

- The tesspers.file is read and is compared on the entire record with the records from tesspers.fixdel. Any matched records are deleted. Unmatched record from tesspers.fixdel are written to an error file.
- The tesspers.file records that did not match tesspers.fixdel are matched on the entire record with the first of each record pair from tesspers.fixrep, tesspers.fixname, tesspers.regfxeme, and tesspers.regfxnoe. The matched records are replaced with the second record of the corresponding record pair from the correction file. Unmatched records from the correction files are written to an error file.
- The replacement records and the tesspers.file records that did not match one of the correction files are compared on SSN with the tesspers.fixadd file. Any records found in both are written to an error file. Unmatched records from either source are processed further.
- Any records that have all-zero current SSNs (column 100) are written to an error file.
- If multiple records have the same SSN (column 10), they are written to an error file.
- The remaining records are written to the output file, the revised tesspers.file.
- The records from the new tesspers.file are also processed further to verify that for each universityid there is only a single current SSN (column 100). Records that violate this rule are written to an error file.

### **tesupdat.exec**

- : The script tesupdt.exec builds the person.file and uaid.file from the tesspers.file. The processing:
- : 1. The tesspers.file is read and verified. Records are discarded if the current SSN is all zeros or non-numeric or if the SSN or universityid is non-numeric. If the SSN field is blank, it is filled in from the current SSN field.
  - : 2. The remaining records are sorted and two derivatives are made:
    - : a. **person.file**: Records for which the SSN matches the current SSN are selected. Only the first occurrence of each universityid is retained. The records are reformatted, sorted on SSN and universityid, and written to person.file. The format of the person.file records is:

PERSON FILE (ADP.PERSON.FLAT) layout:

Columns	Field name	Field Values
1 - 20	Last name	
21 - 40	First name	
41 - 60	Middle name	
61 - 69	Universityid	
73 - 81	SSN	
85	Constituency	E(mployee), A(student/alumni), G(graduate student/alumni), K (kin, might also be a casual employee), S (continuing education/special student -- added by LCV in SYNCSSNS), M (miscellaneous person).
89	Constituency group	Employee staff (ie. faculty) Blank (student), * (graduate student), A (Bi-weekly A), B (Bi-weekly B), D (visiting faculty), F (faculty), L (professional library), M (monthly), R (research staff), T (technical staff), 6 (funny student status?), H (casual hourly employee).
93	Status	A(ctive), D(ead), I(nactive), G(raduated), S(eparated) blank (funny student status?), L (employee leave), R (retired), C (casual employee), K (kin, might also be a casual employee), M (miscellaneous person).
97 -102	Hire date	for employees
105 -110	Separation date	for employees
113 -119	Department account	for employees (may be blanks or only first three characters may be non- blank)
121 -122	Class year	for students

- : b. **uaid.file:** The records are reformatted and written to the uaid.file, which has the following  
 : format:

ID	CHAR ( 9 )	KEY TO FILE
SSN	CHAR ( 9 )	ALTERNATE INDEX
LASTNAME	CHAR ( 20 )	
FIRSTNAME	CHAR ( 15 )	
MID INIT	CHAR ( 1 )	
BIRTHDAY	YYMMDD	
COHORT	CHAR ( 2 )	
DEPT	CHAR ( 3 )	
CLASS	CHAR ( 2 )	
MELLON COHORT	CHAR ( 2 )	
BLANK	CHAR ( 11 )	

### ldapupdt.exec

: The script ldapupdt.exec builds the LDIF updates for updating addresses, telephone numbers, fax  
 : numbers, voicemailbox numbers, and PAC numbers (phone access codes) in LDAP based on the  
 : information from the Campus Community derivatives address.ccfile and phone.ccfile. The  
 : processing:

- : • The file studtr.active is built to list active students whose address and phone information  
 : should be updated from Campus Community:
  - : 1. An extract of tesspers.file is made that contains only the current SSN records (columns  
 : 10-18 equal columns 100-108) for non-students (records that don't have "A" or "G" in  
 : column 79).
  - : 2. The active records from studtr.file are selected by applying the following rules:
    - : a. If the SSN indicates a dummy student (the number begins "99999"), the record is  
 : ignored.
    - : b. If the status is "\_" (active), "A" (absent), "L" (on leave), "R" (required leave), or "V"  
 : (voluntary leave) and the class year is one of the current four, the record is selected.
    - : c. If the status is "F" (failed to qualify), "L", "R", or "V" and the class year is one of the  
 : previous two, the record is selected.
    - : d. If the "class year" begins with "G" (indicating a graduate student) and the status code in  
 : column 60 is "a" (indicating an active graduate student), the record is selected.
  - : 3. The two extracts are compared on universityid and only the studtr records not matching a  
 : tesspers record are written to the file studtr.active. (That is, the tesspers extract contains  
 : only non-students, so matching entries from studtr should be for non-students.)
- : • The hand-maintained file buckley.list, which contains records for people who have requested  
 | FERPA protection, is processed to build updates to provide the requested protection. Note  
 | that this mechanism is used both for students who have formally asked to have their

information withheld in accordance with the Buckley Amendment (FERPA) and for employees who need similar protection (typically set up at the request of Public Safety or of OIT's Policy Advisor). The non-comment records in buckley.list are in the format:

Columns	Field name	Field Values
1 - 9	SSN	
11 - 19	Universityid	
21 - 23	Status	
25 - 47	Name	
48 - 50	Protection Category (see below)	
52 - 59	Date Entered	
62 - 69	Netid of requestor	
71 - 72	Flag	++ indicates a student who did not apply early enough to meet the formal requirements of the Buckley Act, but whom we will treat as having Buckley protection against publication of their information. == indicates a student who is currently not enrolled and who wishes not to be visible in the Directory. We will treat this as a case of Buckley protection against publication of their information. %% indicates a student who has requested Buckley protection and who should be contacting the Registrar's Office for formal protection.

Comment records start with an asterisk and can be ignored.

The categories of protection include:

- ALL:** All directory information (Name, Domicile address, Domicile telephone number, Email address, Frist mailbox address, Voicemailbox number, dates of enrollment, college concentration, degree, honors) are withheld.<sup>1</sup>

<sup>1</sup> Terminology (from a conversation with Anthony Broh and Joseph Greenberg on 11/01/1994):

- Domicile - The dormitory or residence where the student sleeps when studying at Princeton.
- Permanent home - The residence where the student lives when not living or studying in Princeton. For Undergraduates this is usually the location to which grades are sent.
- Office - An on-campus office location (and possibly telephone) provided by a department for the use of its students.

- : 2. **EO:** Only the email address is visible in LDAP and there is no entry in the printed Campus Directory.
  - : 3. **EN:** Only the email address is visible in LDAP and only the name and email address are shown in the printed Campus Directory.
  - : 4. **DO:** Only the Domicile address, Domicile telephone number, Frist mailbox number, and Voicemailbox number are withheld
  - : 5. **CL:** Only the class year is withheld.
  - : 6. **AO:** Only the domicile address is withheld.
  - : 7. **OO:** Only the Office address, Office telephone number, and Voicemailbox number are withheld.
  - : 8. **HO:** Only the Permanent home address and Permanent home telephone number are withheld.
- : Two files containing DELETE commands, buckley.address and buckley.phone, are built from buckley.file:
- : 1. Records specifying ALL, EO, or EN are processed to build “HOME DELETE”, “CAMP DELETE”, “OL1 DELETE”, and “OL2 DELETE” records for the buckley.address file and “ASTR DELETE”, “FAX DELETE”, “HOME DELETE”, “OL1 DELETE”, “OL2 DELETE”, and “VCML DELETE” records for the buckley.phone file.
  - : 2. Records specifying DO are processed to build “HOME DELETE” records for buckley.address and “HOME DELETE” and “VCML DELETE” records for buckley.phone.
  - : 3. Records specifying AO are processed to build “HOME DELETE” records for buckley.address.
  - : 4. Records specifying OO are processed to build “CAMP DELETE”, “OL1 DELETE”, and “OL2 DELETE” records for buckley.address and “ASTR DELETE”, “FAX DELETE”, “OL1 DELETE”, “OL2 DELETE”, and “VCML DELETE” records for buckely.phone.
- : • Files built by ldiupdad.exec from the nightly LDIF dump are merged to build files of current LDAP addresses and telephone numbers:
    - : 1. Records from puintoff.file, puofloc1.file, puofloc2.file, and pustuloc.file are written to ldapupdt.address and marked as CAMP, OL1, OL2, or HOME, respectively. (Note that the latter is the dormitory address, despite the name.)
    - : 2. Records from puoftel1.file, puoftel2.file, pustutel.file, fax.file, pts42.file, and voicebox.file are written to ldapupdt.phone and marked as OL1, OL2, HOME, FAX, PAC, and VCML, respectively.
  - : • The file ldiupdad.accounts, which lists all regular entries in the most recent LDIF dump, is compared on universityid with campcomm.file, which lists the people in Campus Community. Two files are built for all matched entries:
    - : 1. The file address.deletes contains “CAMP DELETE”, “HOME DELETE”, “OL1 DELETE”, and “OL2 DELETE” records for each of the matched entries.
    - : 2. The file phone.deletes contains “ASTR DELETE”, “FAX DELETE”, “HOME DELETE”, “OL1 DELETE”, “OL2 DELETE”, and “PAC DELETE” records for each of the matched entries. (Temporarily, it does not contain a “VCML DELETE” record, because the voicemailboxes are not yet all listed in phone.ccfile.)

- 
- : The purpose of these two files is to delete LDAP information for all people in Campus  
 : Community unless an override is found in another file (see below).
- : • Files of address updates to be applied to LDAP are built:
- : 1. The cumulative Campus Community address file, address.ccfile, is read and the file  
 : buckley.address is appended to it. The combined file is sorted (the sort is stable), and only  
 : the last entry for each universityid and address type is retained. (This selects the records  
 : from the buckley.address file in preference to records from address.ccfile.) Any records  
 : with an empty address field are written to an error file.
  - : 2. Records with address type CAMP are compared with the current SSN (columns 10-18  
 : equal columns 100-108) records from tesspers.file for employees (“E” in column 79); only  
 : those that are found in this extract of tesspers and that do not have an address of  
 : “Unspecified Dept” are selected for further processing. If the records contain the comment  
 : “\*\*\*\*\*Departmental Pickup\*\*\*\*\*”, that is blanked out. If the string “PPPL” is found in  
 : columns 15-69, the contents of columns 15-124 are replaced with the string “Forrestal  
 : Campus, PPPL”.
  - : Records with address type OL1 or OL2 and records with address type HOME (dormitory  
 : addresses) that specify “DELETE” are selected for further processing. Other records with  
 : type HOME are selected only if a record with a matching universityid is found in  
 : studtr.active.
  - : 3. These records are compared (on universityid and address type) against the address.deletes  
 : file (the list of all addresses in LDAP converted to “DELETE” requests). Any records  
 : from address.deletes that do not match one of the passed ccfile records are passed to the  
 : next step along with the ccfile records.
  - : 4. These records are compared on universityid with the records from ldiupdad.accounts (a list  
 : of the distinguished names from LDAP). Any for which a DN is not found are written to  
 : an error file.
  - : 5. The addresses are then reformatted:
    - : a. If the country field says “United States”, it is blanked out and the address is examined  
 : to determine whether it is local. If the zipcode is 08540, 08542, 08543, or 08544 or the  
 : address contains “GFDL” or “Plasma Physics” (any case), the city, state, and zipcode  
 : are also blanked out.
    - : b. For all countries, an address field is built off the end of the record. It consists of three  
 : 55-character “street” addresses, each followed by a dollar sign. This is followed by the  
 : city name, province or state abbreviation (2 characters, U.S., Canada, and Mexico  
 : only), and the postal/zip code (if they’ve not been blanked out) and another dollar sign.  
 : This is followed by the country name (if it’s not been blanked out).
    - : c. Trailing blanks and dollar signs are removed from the appended address. Runs of  
 : dollar signs are reduced to a single dollar sign.
  - : 6. The resultant records are compared on universityid and address type with the file  
 : ldapupdt.address. If no match is found in ldapupdt.address and the non-matched record is  
 : not a “DELETE” record, it is written to the file ldapupdt.addradd, which lists addresses  
 : that need to be added to LDAP.

: If a match is found in ldapupdt.address, the matching record is appended to the ccfile  
 : record and the addresses are compared. If the address in the record derived from  
 : address.ccfile is different than the address in ldapupdt.address and it is not “DELETE”, the  
 : record is written to the file ldapupdt.addrrepl, which lists addresses that need to be changed  
 : in LDAP. If it is “DELETE”, the record is written to the file ldapupdt.addrdel, which lists  
 : addresses that need to be deleted from LDAP.

- : • Files of telephone updates to be applied to LDAP are built:
  - : 1. The cumulative Campus Community telephone number file, phone.ccfile, is read and the  
 : file buckley.phone is appended to it. The combined file is sorted (the sort is stable), and  
 : only the last entry for each universityid and phone number type is retained. (This selects  
 : the records from the buckley.phone file in preference to records from phone.ccfile.) Any  
 : records with an empty phone number field are written to an error file.
  - : 2. Records are selected and reformatted according to type:
    - : a. Records with phone type OL1, OL2, FAX, and records with phone type HOME that  
 : specify “DELETE” are selected for further processing. Other records with type HOME  
 : are selected only if a record with a matching universityid is found in studtr.active. The  
 : phone numbers are reformatted to include hyphens after the areacode and exchange and  
 : copied to the end of the record.
    - : b. Records with phone type PAC are selected only if they are for administrative phone  
 : authorization codes (columns 20-21 are blank). The script ldapupdt1.rexx is invoked to  
 : encrypt the PAC code and append it to the end of the record.
    - : c. Records with phone type VCML are selected and the voicemailbox number is copied  
 : unchanged to the end of the record.
  - : 3. These records are compared (on universityid and phone type) against the phone.deletes file  
 : (the list of all phone numbers in LDAP converted to “DELETE” requests). Any records  
 : from phone.deletes that do not match one of the passed ccfile records are passed to the next  
 : step along with the ccfile records.
  - : 4. These records are compared on universityid with the records from ldiupdad.accounts (the  
 : list of DNSs). Any for which a DN is not found are written to an error file.
  - : 5. The remaining records are compared on universityid and phone type with the file  
 : ldapupdt.phone. If no match is found in ldapupdt.phone and the non-matched record is not  
 : a “DELETE” record, it is written to the file ldapupdt.phonadd, which lists phone numbers  
 : that need to be added to LDAP.

: If a match is found in ldapupdt.phone, the matching record is appended to the ccfile record  
 : and the phone numbers are compared. If the phone number in the record derived from  
 : phone.ccfile is different than the phone number in ldapupdt.phone and it is not “DELETE”,  
 : the record is written to the file ldapupdt.phonrepl, which lists phone numbers that need to  
 : be changed in LDAP. If it is “DELETE”, the record is written to the file ldapupdt.phondel,  
 : which lists phone numbers that need to be deleted from LDAP.

- : • The six files built in the two previous steps are compared on universityid with the file  
 : ldiupdad.accounts to build the file ldapupdt.dns, which lists the distinguished names to be  
 : updated.

- : • The file of LDIF updates is built:
  - : 1. The file ldapupdt.dns is read and reformatted so that each output record consists of the  
: universityid, the numeral “0”, a cent sign, the DN, a cent sign, and “changetype: modify”.
  - : 2. The file ldapupdt.addradd is read and processed according to the address type:
    - : — **CAMP:** The output record consists of the universityid, the numeral “1”, a cent sign,  
: “add: punterofficeaddress”, a cent sign, “punterofficeaddress: ”, the reformatted  
: address from the end of the input record, a cent sign, and a hyphen.
    - : — **OL1:** The output record consists of the universityid, the numeral “1”, a cent sign, “add:  
: street”, a cent sign, “street: ”, the reformatted address from the end of the input record, a  
: cent sign, and a hyphen.
    - : — **OL2:** The output record consists of the universityid, the numeral “1”, a cent sign, “add:  
: puofficelocation2”, a cent sign, “puofficelocation2: ”, the reformatted address from the  
: end of the input record, a cent sign, and a hyphen.
    - : — **HOME:** The output record consists of the universityid, the numeral “1”, a cent sign,  
: “add: studentstreet”, a cent sign, “studentstreet: ”, the reformatted address from the end  
: of the input record, a cent sign, and a hyphen.
  - : 3. The file ldapupdt.addrdel is read and processed according to the address type:
    - : — **CAMP:** The output record consists of the universityid, the numeral “1”, a cent sign,  
: “delete: punterofficeaddress”, a cent sign, “punterofficeaddress: ”, the reformatted  
: address from the end of the input record, a cent sign, and a hyphen.
    - : — **OL1:** The output record consists of the universityid, the numeral “1”, a cent sign,  
: “delete: street”, a cent sign, “street: ”, the reformatted address from the end of the input  
: record, a cent sign, and a hyphen.
    - : — **OL2:** The output record consists of the universityid, the numeral “1”, a cent sign,  
: “delete: puofficelocation2”, a cent sign, “puofficelocation2: ”, the reformatted address  
: from the end of the input record, a cent sign, and a hyphen.
    - : — **HOME:** The output record consists of the universityid, the numeral “1”, a cent sign,  
: “delete: studentstreet”, a cent sign, “studentstreet: ”, the reformatted address from the  
: end of the input record, a cent sign, and a hyphen.
  - : 4. The file ldapupdt.addrrepl is read and processed according to the address type:
    - : — **CAMP:** The output record consists of the universityid, the numeral “1”, a cent sign,  
: “delete: punterofficeaddress”, a cent sign, “punterofficeaddress: ”, the old address from  
: the end of the input record, a cent sign, a hyphen, “add: punterofficeaddress“, a cent  
: sign, “punterofficeaddress: ”, the reformatted address from the middle of the input  
: record, a cent sign, and a hyphen.
    - : — **OL1:** The output record consists of the universityid, the numeral “1”, a cent sign,  
: “delete: street”, a cent sign, “street: ”, the old address from the end of the input record, a  
: cent sign, a hyphen, “add: street“, a cent sign, “street: ”, the reformatted address from  
: the middle of the input record, a cent sign, and a hyphen.

- : — **OL2:** The output record consists of the universityid, the numeral “1”, a cent sign, “delete: puofficelocation2”, a cent sign, “puofficelocation2: ”, the old address from the end of the input record, a cent sign, a hyphen, “add: puofficelocation2“, a cent sign, “puofficelocation2: ”, the reformatted address from the middle of the input record, a cent sign, and a hyphen.
  - : — **HOME:** The output record consists of the universityid, the numeral “1”, a cent sign, “delete: studentstreet”, a cent sign, “studentstreet: ”, the old address from the end of the input record, a cent sign, a hyphen, “add: studentstreet“, a cent sign, “studentstreet: ”, the reformatted address from the middle of the input record, a cent sign, and a hyphen.
5. The file ldapupd.phonadd is read and processed according to the phone type:
- : — **OL1:** The output record consists of the universityid, the numeral “1”, a cent sign, “add: telephonenumber”, a cent sign, “telephonenumber: ”, the reformatted phone number from the end of the input record, a cent sign, and a hyphen.
  - : — **OL2:** The output record consists of the universityid, the numeral “1”, a cent sign, “add: puofficetelephone2”, a cent sign, “puofficetelephone2: ”, the reformatted phone number from the end of the input record, a cent sign, and a hyphen.
  - : — **HOME:** The output record consists of the universityid, the numeral “1”, a cent sign, “add: studenttelephonenumber”, a cent sign, “studenttelephonenumber: ”, the reformatted phone number from the end of the input record, a cent sign, and a hyphen.
  - : — **FAX:** The output record consists of the universityid, the numeral “1”, a cent sign, “add: facsimiletelephonenumber”, a cent sign, “facsimiletelephonenumber: ”, the reformatted phone number from the end of the input record, a cent sign, and a hyphen.
  - : — **PAC:** The output record consists of the universityid, the numeral “1”, a cent sign, “add: pts42”, a cent sign, “pts42: ”, the encrypted PAC from the end of the input record, a cent sign, and a hyphen.
  - : — **VCML:** The output record consists of the universityid, the numeral “1”, a cent sign, “add: voicemail”, a cent sign, “voicemail: ”, the voicemail phone number from the end of the input record, a cent sign, and a hyphen.
6. The file ldapupd.phondel is read and processed according to the phone type:
- : — **OL1:** The output record consists of the universityid, the numeral “1”, a cent sign, “delete: telephonenumber”, a cent sign, “telephonenumber: ”, the old phone number from the end of the input record, a cent sign, and a hyphen.
  - : — **OL2:** The output record consists of the universityid, the numeral “1”, a cent sign, “delete: puofficetelephone2”, a cent sign, “puofficetelephone2: ”, the old phone number from the end of the input record, a cent sign, and a hyphen.
  - : — **HOME:** The output record consists of the universityid, the numeral “1”, a cent sign, “delete: studenttelephonenumber”, a cent sign, “studenttelephonenumber: ”, the old phone number from the end of the input record, a cent sign, and a hyphen.

- : — **FAX:** The output record consists of the universityid, the numeral “1”, a centsign, “delete: facsimiletelephonenumber”, a centsign, “facsimiletelephonenumber: ”, the old phone number from the end of the input record, a centsign, and a hyphen.
  - : — **PAC:** The output record consists of the universityid, the numeral “1”, a centsign, “delete: pts42”, a centsign, “pts42: ”, the old PAC from the end of the input record, a centsign, and a hyphen.
  - : — **VCML:** The output record consists of the universityid, the numeral “1”, a centsign, “delete: voicemailbox”, a centsign, “voicemailbox: ”, the old voicemail number from the end of the input record, a centsign, and a hyphen.
7. The file ldapupdt.phonrepl is read and processed according to the phone type:
- : — **OL1:** The output record consists of the universityid, the numeral “1”, a centsign, “delete: telephonenumber”, a centsign, “telephonenumber: ”, the old phone number from the end of the input record, a centsign, a hyphen, “add: telephonenumber”, a centsign, “telephonenumber: ”, the reformatted phone number from the middle of the input record, a centsign, and a hyphen.
  - : — **OL2:** The output record consists of the universityid, the numeral “1”, a centsign, “delete: puofficetelephone2”, a centsign, “puofficetelephone2: ”, the old phone number from the end of the input record, a centsign, a hyphen, “add: puofficetelephone2”, a centsign, “puofficetelephone2: ”, the reformatted phone number from the middle of the input record, a centsign, and a hyphen.
  - : — **HOME:** The output record consists of the universityid, the numeral “1”, a centsign, “delete: studenttelephonenumber”, a centsign, “studenttelephonenumber: ”, the old phone number from the end of the input record, a centsign, a hyphen, “add: studenttelephonenumber”, a centsign, “studenttelephonenumber: ”, the reformatted phone number from the middle of the input record, a centsign, and a hyphen.
  - : — **FAX:** The output record consists of the universityid, the numeral “1”, a centsign, “delete: facsimiletelephonenumber”, a centsign, “facsimiletelephonenumber: ”, the old phone number from the end of the input record, a centsign, a hyphen, “add: facsimiletelephonenumber”, a centsign, “facsimiletelephonenumber: ”, the reformatted phone number from the middle of the input record, a centsign, and a hyphen.
  - : — **PAC:** The output record consists of the universityid, the numeral “1”, a centsign, “delete: pts42”, a centsign, “pts42: ”, the old PAC from the end of the input record, a centsign, a hyphen, “add: pts42”, a centsign, “pts42: ”, the encrypted PAC from the middle of the input record, a centsign, and a hyphen.
  - : — **VCML:** The output record consists of the universityid, the numeral “1”, a centsign, “delete: voicemailbox”, a centsign, “voicemailbox: ”, the old voicemail number from the end of the input record, a centsign, a hyphen, “add: voicemailbox”, a centsign, “voicemailbox: ”, the voicemail phone number from the middle of the input record, a centsign, and a hyphen.
8. All of the records are collected together and sorted on the universityid and type code (in column 10). The universityid and type code are then removed and the records are split at the centsigns. (At this point, the DN record for each user is before the other records for that user.) The last hyphen record for each user is deleted.

- : 9. The records are written to the file ldif.updates. A linefeed character is appended to each
- : record and the records are translated from codepage 1047 to codepage 819 (from EBCDIC
- : to ASCII). The resulting file is FTP'd (in binary mode) to directory.Princeton.EDU as
- : /tmp/update.ldif.