Useful Utilities for System Administrators

SY1397

Don Prezioso

Baldwin-Wallace College
275 Eastland Road
Berea, OH 44017
440-826-2316

E-mail: dprez@bw.edu
# Table of Contents

## Useful Utilities for System Administrators

<table>
<thead>
<tr>
<th>Useful Utilities for System Administrators</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disclaimer</td>
<td>1</td>
</tr>
</tbody>
</table>

## FTP Security & Login Control

<table>
<thead>
<tr>
<th>FTP Security &amp; Login Control</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/profile</td>
<td>2</td>
</tr>
<tr>
<td>datatel.login</td>
<td>3</td>
</tr>
<tr>
<td>bwc_utd.c</td>
<td>4</td>
</tr>
<tr>
<td>LOGIN.EXECUTE</td>
<td>7</td>
</tr>
<tr>
<td>set.access</td>
<td>8</td>
</tr>
<tr>
<td>X/MYHOME&amp;</td>
<td>9</td>
</tr>
<tr>
<td>lock.logins</td>
<td>10</td>
</tr>
<tr>
<td>unlock.logins</td>
<td>11</td>
</tr>
</tbody>
</table>

## User Accounts

<table>
<thead>
<tr>
<th>User Accounts</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>mkadvise</td>
<td>12</td>
</tr>
<tr>
<td>pwdupdt.c</td>
<td>14</td>
</tr>
<tr>
<td>MAKE.ADVISOR</td>
<td>16</td>
</tr>
</tbody>
</table>

## UniData Security

<table>
<thead>
<tr>
<th>UniData Security</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANGE.PASSWORD</td>
<td>17</td>
</tr>
<tr>
<td>STAFF.RESTRICTED</td>
<td>18</td>
</tr>
<tr>
<td>AE Security</td>
<td>19</td>
</tr>
<tr>
<td>PREPROG_AE</td>
<td>19</td>
</tr>
<tr>
<td>POSTPROG_AE</td>
<td>22</td>
</tr>
</tbody>
</table>

## System Stuff

<table>
<thead>
<tr>
<th>System Stuff</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>BATCH</td>
<td>25</td>
</tr>
<tr>
<td>CATALOG.CUSTOM.SOURCE</td>
<td>31</td>
</tr>
<tr>
<td>dmictl</td>
<td>32</td>
</tr>
<tr>
<td>weekly.analysis</td>
<td>36</td>
</tr>
<tr>
<td>MAKE.FILE.STATIC</td>
<td>38</td>
</tr>
<tr>
<td>clean.utmp.c</td>
<td>40</td>
</tr>
</tbody>
</table>

March 28, 2001
Useful Utilities for System Administrators

Over the last 15 years I have spent a great deal of time trying to make the task of system administration less tedious. I will spend hours to program my way around 30 minutes of data entry. Having changed operating systems, database systems, R13-R16, and the college I work for all in the last 5 years, I have needed to re-create a number of the tools I have developed to make my job less monotonous. This presentation is a collection of the programs, scripts, and subroutines that I have found most valuable in getting my job done. For the most part, they are examples of how a little bit of programming can greatly enhance my ability to accomplish the day-to-day tasks of system administration.

I have broken these examples down into four general areas.

- The first group of examples show how we have been able to give our users access to FTP and other methods for downloading and uploading files with our Colleague machine without giving them a dangerous level of access to our database files.
- The second group deals with how we set up new users so they have a minimal level of access, with a minimal effort on our part.
- The third area deals with things we have done to provide security at the UniData level.
- Finally the last segment of this session consists of a few examples of utilities that I have found greatly valuable, but don’t fit into the first two segments.

Disclaimer

While the examples given here have been tested and work properly on the Colleague system at Baldwin-Wallace College, we cannot guarantee that they will work properly on any other system. Differences in operating system, release level, security setup, and account structure in particular may cause these examples to work improperly.
FTP Security & Login Control

When users are utilizing Colleague, they need full (read and write) access to all (or at least most) of the database files. As long as they remain in UniData, there are protections that can be put in place to keep users from deleting or damaging your institution’s data files. The problem arises when those users need access to the system when they are NOT in UniData. The example we are using in this case is FTP, but it could easily be any time a user needs access to the system when they are not in the UniData environment.

To solve this problem, we use a combination of Unix scripts, a C program, and a Unibasic program to give users access to the Colleague database files ONLY while they are in Unidata. When they are NOT in Unidata (like during an FTP session) they will not have access to ANY of the database files with the exception of a few directories (like _HOLD_) that we want them to be able to access with FTP.

/etc/profile

To begin with, when users log in, they execute the system profile script. Of importance here is the section of the script that allows a limited group of people (generally I.T. staff) access to the operating system level. All other users then execute another script (datatel.login) using the exec command so that if they should somehow abort from that script, or from Unidata, they will be without a shell, and thus be logged out. Note that using the system profile script means that the user never executes their personal profile script from their home directory since they could modify that. Also note that we place /datatel/bin in the PATH environment variable for all users BEFORE the path to the Unidata executables. That way we can put our own version of the ‘udt’ command there and it will be executed instead of the real thing.

```
PATH=/usr/local/bin:/datatel/bin:/usr/ud41/bin: { rest of path removed }
. /usr/udms/udmsprofile

CASE $USER in
  root | datatel | adm | udmsmgr ) ;;
  dprez | mshadrak | ragnew | dkilbane ) ;;
  allofus | Reggie )
    exec /datatel/bin/datatel.login ;;
  *) stty > /dev/null
    if [ 0 -eq $? ] ; then
      exec /datatel/bin/datatel.login
      else
        cd /datatel/live/collive
        fi
    fi
esac

stty intr "^C";
stty quit "\"";
stty susp "^Z";
if [ 0 -ne $? ] ; then
  cd /datatel/live/collive
  fi
```
datatel.login

Most users will continue on to the datatel.login script. The first thing this script does is to check if the user in question is one of those allowed to access multiple accounts (like coldev or coltest). If they are not one of the users listed in the file menu.users (a text file with one username per line) they are immediately taken to the live main account where UniData is started. This is not the real ‘udt’ command remember, this is our own version. If they are allowed to access multiple accounts, then they are given a menu to choose from, which determines where they are taken, and again, our version of UniData is started.

```bash
#!/bin/ksh
TERM=bwvt420 ; export TERM

if [[ -z $( grep ^$USER /datatel/menu.users ) ]] ; then
cd collive ; exec udt ; exit
fi

while true; do
  clear
  cat <<end_of_menu
You are seeing this menu because you have access to several accounts. If you only wish to access the LIVE account, and do not want to see this menu anymore, send an E-mail message to dprez@bw.edu and I will remove this menu for you.

Which Datatel Account would you like to start?
   L) LIVE Account
   T) Test Account
   D) Development Account
   C) Enrollment Chill Account

   P) Change Password
   X) Exit

end_of_menu

print " Enter L, T, D, C, P, or X: \c"
read ANS

  case $ANS in
    E  ) cd coleduc16 ; exec udt ; exit ;;
    T  ) cd coltest ; exec udt ; exit ;;
    L  ) cd collive ; exec udt ; exit ;;
    D  ) cd coldev ; exec udt ; exit ;;
    C  ) cd chill ; exec udt ; exit ;;
    P  ) passwd ;;
    X  ) exit ;;
    * ) print "Invalid: Try again!" ;;
  esac
esac
sleep 2
done
exit
```
This is our ‘replacement’ for the standard udt command. We create a link to the compiled program from /datatel/bin/udt, and since /datatel/bin is in the search path before the standard udt command this is what gets run. This program gives the user the groups needed to access the database files, then starts the ‘real’ udt command.

Note that this program uses some AIX specific functions (like ‘getgroups’ and ‘setgroups’) that may be different on other operating system. For example, Linux also has getgroups and setgroups functions that do the same thing, but some of the constants may be different. Also, some older flavors of unix do not allow a user to belong to multiple groups at the same time. Because of this, if you are not using AIX, you will need to make some changes to the program, but the same concepts should apply.

/*
Written by Don Prezioso for Baldwin-Wallace College, Jan 4, 1999.
This program gives a user membership in some groups that are needed for the user to access Datatel's database programs and files, and enters The Unidata environment. The program also checks for the existence of two files, /datatel/login.txt, and /datatel/userlock.txt . If /datatel/login.txt exists, then it is printed out and the program waits for input to continue. If /datatel/userlock.txt exists, it is printed and the program waits 5 seconds, then logs the user out.

To decide which groups a user should belong to the program checks the group of the VOC file in the current directory. The groups given to the user are this one, and the group of the file $UDT BIN/udt. If either file cannot be found, then the program exits.

To use this program, it should be called after changing directory to the account the user will be using. For example:

   cd /datatel/live/collive
   exec /datatel/bin/bwc_udt

Running the program this way causes the current shell to be replaced by this program. That way, when the program exits (for any reason) the user will be logged out.

To re-compile the program do the following as root

   xlc -Oo /datatel/bin/bwc_udt /datatel/bin/bwc_udt.c
   chown root:security /datatel/bin/bwc_udt
   chmod u+s /datatel/bin/bwc_udt
*/

#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <grp.h>
#include <unistd.h>
#include <limits.h>
#define LOCK_FILE_NAME "/datatel/userlock.txt"
#define MESSAGE_FILE_NAME "/datatel/login.txt"
#define MAX_LINE_LEN 255
#define GROUP_ERROR UID_MAX

int add_file_group ( const char *file_path )
/*
   Get the group id of the file named by 'file_path' and add it to
   to the groups for this process.
*/
{
    int status = 0;
    int idx = 1;
    int gid_count;
    struct stat* file_info;
    struct stat fibuf;
    gid_t group_id;
    gid_t gid_list[NGROUPS_MAX];
    char temp_text[MAX_LINE_LEN];

    file_info = &fibuf;
    if ( stat( file_path, file_info ) )
    {
        sprintf( temp_text, "Error getting group for %s", file_path );
        perror ( temp_text );
        status = 1;
    }
    else
    {
        group_id = file_info -> st_gid;
        gid_count = getgroups( NGROUPS_MAX, gid_list );
        if ( gid_count == -1 )
        {
            perror( "Error getting group membership" );
            status = 1;
        }
        else
        {
            while ( ( idx < gid_count ) && ( gid_list[idx] != group_id ) ) idx++;
            if ( idx >= gid_count )
            {
                gid_list[gid_count] = group_id;
                if ( setgroups( ++gid_count, gid_list ) )
                {
                    perror( "Error setting group membership" );
                    status = 1;
                }
            }
        }
    }
    return( status );
}

int print_text_file( char *file_path )
/*
Check for the existence of the text file named in 'file_path'
and if it exists print it out to stdout. Also return true if
file existed so appropriate action can be taken. */
{
int status = 0;
char text_file_line[MAX_LINE_LEN];
FILE *text_file;

text_file = fopen(file_path, "r");
if ( text_file )
{
  fgets( text_file_line, MAX_LINE_LEN, text_file );
  while ( !feof( text_file ) )
  {
    printf( "%s", text_file_line );
    fgets( text_file_line, MAX_LINE_LEN, text_file );
  }
  fclose( text_file );
  status = 1;
}
return( status );
}

main ( int argc, char* argv[] )
{
char user_groups[MAX_LINE_LEN];
char temp_text[MAX_LINE_LEN];
char udt_path[MAX_LINE_LEN];
int errors = 0;

sprintf( udt_path, "%s/udt", getenv( "UDTBIN" ) );

if ( ! errors ) errors = ( print_text_file( LOCK_FILE_NAME ) );
if ( ! errors ) errors = add_file_group( udt_path );
if ( ! errors ) errors = add_file_group( "VOC" );
if ( ! errors )
{
  if ( print_text_file( MESSAGE_FILE_NAME ) )
  {
    printf( "\n\n%s", "Press RETURN to continue " );
    fgets( temp_text, MAX_LINE_LEN, stdin );
  }
}
seteuid( getuid() );
argv[0] = "udt";
if ( ! errors ) execv( udt_path, argv );
if ( errors ) sleep( 5 );
exit;
LOGIN.EXECUTE

This is a program that was originally taken from the Colleague Administrator learning guide, which we have modified so that even if a user has no xxx.LOGIN VOC entry they will still run the ST application. This program is called as a part of the LOGIN paragraph in all of our accounts so that we can insure that users will be forced into a menu, and that they will therefore be validated against the UT.OPERS records.

* DEW
* This program is executed upon user login. It searches the VOC file for a paragraph named USERID.LOGIN, where USERID is the UNIX userid of the given user. If the paragraph exists it is executed. This code appears in the Handouts section of the Colleague Administrator student guide.
* David Waldron. April, 1996.
*
* Modified 10/25/99 by Don Prezioso to enter the ST menu if there is no xxx.LOGIN for the user. That way we only need xxx.LOGIN records for those users who either go straight to the colon prompt or to the CF menu.
* OPEN ","VOC" TO F.VOC ELSE
  ABORT "CANNOT OPEN VOC"
END
*
VOC.KEY=@LOGNAME:"LOGIN"
*
READ PARA FROM F.VOC, VOC.KEY THEN
  EXECUTE VOC.KEY
END ELSE
  EXECUTE "ST"
END
*
END

Now, when we create new users at the operating system level, we DO NOT give them access to the database, or to UniData. The only way they get that access is by running the above programs. Since FTP access does not run the programs, then they do not have any access to the database files while using FTP. Also, if there is some other way that they might access our system, or if we wanted to allow them access to the operating system level, they would not be able to access the database.
set.access

At this point, users can only access files they create in directories they would normally be able to access. To allow them access to be able to download hold files, and upload files (like for Financial Aid imports) they will need some limited access to some specific directories. We open up access to the _HOLD_ file for reading only so users can download information, and we open access to a separate directory (FTP) completely for uploading.

Note that on the directories above those that are opened, we only give execute access to ‘others’ (chmod o+x directory). This means that they can ‘cd’ (or the ftp equivalent) through those directories, but they cannot see what files exist in them. That way if someone tries ‘cd /datatel/live/collive’ they can’t see anything, but if they ‘cd /datatel/live/collive/_HOLD_’ then they will be able to see the files in the hold directory. In other words, people trying to use FTP have to know where they are going; they are not able to ‘snoop’ around.

```
#!/bin/ksh
#
# ASH:DVP Reset permissions on datatel directories.
#
echo "Changing mode for upper level directories"
chmod 775 /datatel
chmod 775 /datatel/work
chmod 775 /datatel/live

echo "Changing owner, group and mode for release directories"
chown -R datatel:clive /datatel/release
chmod -R 770 /datatel/release
chown -R datatel:users /datatel/release/COLL16.AppServer
chmod -R 750 /datatel/release/COLL16.AppServer
chown -R datatel:cdev /datatel/development
chmod -R 770 /datatel/development

echo "Changing owner, group and mode for collive files"
chown -R datatel:clive /datatel/live/collive
chmod -R 770 /datatel/live/collive

echo "Changing mode for live _HOLD_ and FTP directories"
chmod o+x /datatel
find /datatel/live/* -type d -prune -exec chmod o+x {} 
find /datatel/live/*/_HOLD_ -type d -prune -exec chmod o+rx {} 
find /datatel/live/*/_HOLD_ -type d -prune -exec chmod g+s {} 
find /datatel/live/*/SAVEDLISTS -type d -prune -exec chmod g+s {} 
find /datatel/live/*/*_FTP_ -exec chmod o+rwx {} 
find /datatel/live/*/*DATA -type d -prune -exec chmod o+x {} 
find /datatel/live/*/*DATA_S -type d -prune -exec chmod o+x {} 
find /datatel/live/*/*DATA_S/SCHED25.DIRECTORY -exec chmod o+rwx {} 
find /datatel/remotes/live/*/*_HOLD_ -type d -prune -exec chmod o+rx {} 
find /datatel/remotes/live/*/*_FTP_ -exec chmod o+rwx {} 

echo "Changing owner, group, and mode for coltest files"
chown -R datatel:cdev /datatel/work/coltest
chmod -R 770 /datatel/work/coltest
```
Useful Utilities for System Administrators  Baldwin-Wallace College

```bash
chmod o+x /datatel/work/coltest
find /datatel/work/* -type d -prune -exec chmod o+x {} \;
find /datatel/work/*/_HOLD_ -type d -prune -exec chmod o+rx {} \;
find /datatel/work/*/_HOLD_ -type d -prune -exec chmod g+s {} \;
find /datatel/work/*/SAVEDLISTS -type d -prune -exec chmod g+s {} \;
find /datatel/work/*/_FTP_ -exec chmod o+rw {} \;
find /datatel/work/*/DATA -type d -prune -exec chmod o+x {} \;
find /datatel/work/*/DATA/DATA_S -type d -prune -exec chmod o+x {} \;
find /datatel/work/*/DATA/DATA_S/SCHED25.DIRECTORY -exec chmod o+rw {} \;

echo "Changing owner, group, and mode for coldev files"
chown -R datatel:cdev /datatel/work/coldev
chmod -R 770 /datatel/work/coldev
chmod o+x /datatel/work/coldev

echo "Changing owner, group, and mode for chill files"
chown -R datatel:cdev /datatel/work/chill
chmod -R 770 /datatel/work/chill
chmod o+x /datatel/work/chill
```

&MYHOME&

Also, we have created a file pointer to the users home directory called &MYHOME& so most downloads can be done there, where they normally have access. Additionally, the user’s home directory is actually the home directory on our campus e-mail system (remotely mounted via NFS) which is also available to networked PCs (via SAMBA) as the user’s U:/ directory.

---

Top of "&MYHOME&" in "VOC", 3 lines, 18 characters.

*---: P
001: DIR
002: @HOME
003: D__HOLD_
Bottom.
*---:
lock.logins

Now that we have a way to give people the access they need while in UniData, we will have times that we want to keep them out, for instance when loading patches, or new releases. One of the things built in to our ‘replacement’ udt command is a check for the existence of the file /datatel/userlock.txt. If this file exists, then the contents of that file are displayed for a little while, and then the user is logged out, without ever entering UniData. You could create that file by hand whenever you needed it, but this script will create a rather generic message more easily. This script will also stop the DMI listener so WebAdvisor users will also be blocked from accessing the system.

#!/bin/ksh
# lock.logins
#
# This script creates a file that is checked at login time
# to see if normal users can access the Administrative Database. The
# file, if it exists, contains a message telling them they are not allowed
# access.
#
lockfile=/datatel/userlock.txt
echo "" >> $lockfile
echo "" >> $lockfile
echo "" >> $lockfile
echo "" >> $lockfile
echo "" >> $lockfile

# * * * NOTICE *** " >> $lockfile

echo "Sorry, but access to the Baldwin-Wallace Administrative
Database is currently restricted.
For more information, please call Administrative Computing at 2310."
chgrp staff $lockfile
chmod 775 $lockfile
/datatel/bin/dmictl stop

echo "Access for non-privileged users to unidata has been locked"

echo "To allow users back in, use run /datatel/unlock.logins"
unlock.logins

Of course it would also be nice to have a script to get rid of that file and restart the listener...

```
#!/bin/ksh
# unlock.logins
#
# This script deletes a file created by the lock.logins script, allowing
# access to the Administrative Database.
#
rm /datatel/userlock.txt
/datatel/bin/dmictl start
echo "Users can now access the Administrative Database"
```

***

These scripts and programs are only one method for handling FTP security and login control. There are a number of other methods that are also effective including using more secure FTP server daemons, restricting what users can use FTP and not using FTP at all, but using some other form of file transfer instead (Kermit for example). Each method has its advantages and drawbacks that may make them more, or less suitable at your site. The best advice is to know your setup very well, and “Be afraid... Be VERY afraid”.
User Accounts

Creating accounts for users is one system administration task that is greatly complicated on a Colleague system. While many systems might have a way to create large groups of users in a batch mode, those setups are often not what is needed for Colleague. Then once the accounts are created, there is still the task of creating OPERS records at least. We needed a way to set up a large number of faculty in a short time since they would be starting to do on-line advising. Additionally, since all faculty, staff and students have existing accounts on the e-mail system, we wanted the new accounts to be coordinated with those on the e-mail system. The result are the following script, C program and Unibasic program that work together to create advisor accounts, either one at a time, or in a batch.

mkadvise

The mkadvise script is used to get user information from our campus e-mail server and create accounts. It does this by taking information from the /etc/passwd file on the remote system to create new records for our passwd file. Then a script is created on-the-fly which changes the users login script for peek, runs an AIX utility (pwdck) to create proper user entries, and runs a C program (pwdupdt) to make their password the same as their e-mail password. Finally, another script is created on-the-fly to make UT.OPERS records for the users with very limited access. Much of the specifics of this script are very tailored to our site, and could not be used at most other schools. It is the techniques used, however, that will be of some interest.

#!/bin/ksh
# This script is used to create advising accounts for all faculty and staff
#
# userfile=$1
new_users=/tmp/new_users
cur_users=/tmp/current_users
bad_users=/datatel/bad.users
new_pwds=/tmp/new_passwd
peek_updt=/tmp/peek_update
oper_updt=/tmp/opers_update
#
# Find out if the argument is a username or a file, and create $new_users file.
#
if [[ $USER != "root" ]]; then
  echo "Error: mkadvise can only be run as root"
  echo
  userfile=""
fi
if [[ -z $userfile ]]; then
  echo "Usage: mkadvise File"
  echo " or: mkadvise Username"
else
  if [[ -z $userfile ]]; then
    awk '{ OFS="" ; print "$",$1,":" }' $userfile > $new_users
  else
    awk '{ OFS="" ; print "$",$1,":" }' /etc/passwd | sort > $cur_users
  fi
fi
# Get passwd entries from fileservice for new faculty and staff
#
echo "*** Getting fileservice:/etc/passwd entries for new users ***"
rsh fileservice ypcat passwd |
grep -f $new_users |
grep -E ":/home/staff/|:/home/fac/" |
grep -v "^LOCED" |
grep -v "^www" |
grep -v -f $cur_users |
grep -v -f $bad_users |
sort -t: +3 -n > $new_pwds
#
# Only do the rest of this if there were some valid passwd entries
# if [[ -s $new_pwds ]]; then
#
# Add these entries to the /etc/passwd file.
# cp /etc/passwd /etc/passwd.bak
cat $new_pwds >> /etc/passwd
#
# Change the default shell so they can be 'peeked' on
#
echo "*** Creating script to update users shells for peek ***"
echo "#!/bin/ksh" > $peek_updt
echo "# Script to update users shells for peek ***" >> $peek_updt
awk -F : '# print "chuser shell="/bin/pksh-ksh"$1 "' $new_pwds >> $peek_updt
echo "# Now update the passwd entries so they can log in" >> $peek_updt
echo "pwdck -y ALL" >> $peek_updt
awk -F : '# print "pwdupdt",$1,$2 "' $new_pwds >> $peek_updt
#
ksh $peek_updt
#
# Create UT.OPERS records so they can access the Course Availability screen.
#
echo "*** Creating script to update UT.OPERS records ***"
echo "#!/bin/ksh" > $oper_updt
echo "# Script to create advisor accounts for the users below" >> $oper_updt
echo "cd /datatel/live/collive" >> $oper_updt
echo "udt <<NO.MORE.OPERS" >> $oper_updt
awk -F : '# print "MAKE.ADVISOR",$1,$5 "' $new_pwds >> $oper_updt
echo "NO.MORE.OPERS" >> $oper_updt
#
ksh $oper_updt
#
fi
fi
echo "Done!"
pwdupdt.c

This program was written to allow us to change a user’s password so that it would be the same as the password on another system. This is not something to be recommended due to security issues. However, we found that this is precisely what faculty wanted regardless of the security implications. Also, since the access being granted was very limited, and not particularly sensitive, it was worth the risk for the added ease of use.

This program was written by Don Prezioso for Baldwin-Wallace College to update the records in /etc/security/passwd with a new version of the encrypted password, usually obtained from the passwd file of another system.

The syntax for the command is:

cmd [username [encrypt.password]]

Note: You must be logged in as root to run this command.

Compile using the command: xlc -l s -Oo pwdupdt pwdupdt.c

/*

#include <string.h>
#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#include <userpw.h>
#include <usersec.h>
#include <sys/types.h>

/**************************************************************************/

int set_password( char *user, char *pass )
{
    /* Change the encrypted password for 'user' to 'pass'. */

    struct userpw pwdata;
    int errors = 0;
    time_t time_val;

    /*
       Make sure /etc/passwd has a '!' in the password field, so that
    the /etc/security/passwd file will be used for this user.
    */

    errors = setuserdb( S_READ | S_WRITE );
    if (! errors ) errors = putuserattr( user, S_PWD, "!", SEC_CHAR );
    if (! errors ) errors = putuserattr( user, NULL, NULL, SEC_COMMIT );
    if (! errors ) errors = enduserdb();

    if (! errors ) errors = setpwdb( S_READ | S_WRITE );
    if (! errors )
    {
strncpy( pwdata.upw_name, user, PW_NAMELEN );
pwdata.upw_passwd = pass;
pwdata.upw_lastupdate = time( &time_val );
errors = putuserpw( &pwdata );
}
if ( ! errors ) errors = endpwdb();

return( errors );

/**************************************************************************/

int main( int argc, char* argv[] )
{

char *user_name, *password;
char user[PW_NAMELEN], pass[20];
int errors = 0;

if ( argc > 1 ) user_name = argv[1];
else { printf( "User name:"); user_name = gets( user ); }
if ( argc > 2 ) password = argv[2];
else { printf( "Password:" ); password = gets( pass ); }

if ( set_password( user_name, password ) )
{
    printf( "Error setting password for %s\n", user_name );
    errors = 1;
}

if ( ! errors ) printf( "User %s successfully changed!\n", user_name );
return( errors );
}
MAKE.ADVISOR

This Unibasic program essentially does the job that would be done by hand using the UT>SOD screen. Since this should only be used to create new users, we check to see if the record already exists, and if not, then create it.

```unibasic
PROGRAM MAKE.ADVISOR
COMMON /MKADVSR/ OPENED, F.UT.OPERS
IF NOT(OPENED) THEN
   OPEN ',UT.OPERS' TO F.UT.OPERS ELSE
   STOP "UNABLE TO OPEN UT.OPERS FILE - OPERATOR NOT CREATED"
END
OPENED = 1
END
COMMAND.LINE = @SENTENCE
COMMAND.LINE = TRIM(COMMAND.LINE)
USER.ID = FIELD(COMMAND.LINE," ",2,1)
USER.ID = OCONV(USER.ID,"MCU")
USER.NAME = FIELD(COMMAND.LINE," ",3,9)
PRINT "Creating advisor operator record for: ":USER.ID:" ":USER.NAME
READ R.UT.OPERS FROM F.UT.OPERS, USER.ID THEN
   STOP "Operator ":USER.ID:" already exists - not modified"
END ELSE
   R.UT.OPERS = ""
   R.UT.OPERS<1> = "XRG"
   R.UT.OPERS<3> = "3"
   R.UT.OPERS<4,1> = "UT.BASE"
   R.UT.OPERS<4,2> = "ADVISE"
   R.UT.OPERS<16> = USER.NAME
   WRITE R.UT.OPERS ON F.UT.OPERS, USER.ID
END
END

***

These scripts and programs are not something you will likely be able to use “as is” at your site. Rather, I have presented them to show you some of the things that can be done by combining scripts and programs, using each for their strong points to get a complex job done. In this case I use a script to create other scripts which run programs. I have also done the reverse and used programs to execute scripts or system commands. Again, the best advice is to know your system. If you think something might be possible (like adding records to /etc/passwd and letting the system turn them into proper user records) find some local unix expert and ask them. Better yet, find some time when the system is not being used and has been backed up, and give it a try. The time you save may well be your own...
UniData Security

We have seen how you can start securing your system at the operating system level, and Datatel provides for securing the system once users are in Colleague or Benefactor menus using Envision security. What is left is the need to provide security at the UniData level. This is partly a matter of securing UniData functions that might be dangerous. We have some users we want to be able to write their own queries, and who need access to the colon prompt. For those folks we want to reduce the possibility that they will do something damaging. This is accomplished through training and reducing their ability to do harm.

The majority of our users however, do not need to do their own queries, but occasionally need to do things that are not a standard part of the Colleague menus. Examples would be things like changing their login password, LISTUSER, Ipstat, or listing locks. For these kinds of activities we needed a way to run operating system commands, but from within the Colleague menus.

CHANGE.PASSWORD

Do you want users to be able to change their password from within a menu? This program can be called from a menu item so that users can change their login password even if they do not have colon prompt access. The technique of using PCPERFORM can be used for any OS level command you might want. For example, a similar program could execute the command “listuser | more”. For even more complex tasks, you could create a shell script that does what you want, and call it using PCPERFORM.

```
PROGRAM CHANGE.PASSWORD
* Written 4/30/98 by Don Prezioso for Baldwin-Wallace College, this program is
* used to allow users to change their login password.
CRT @(-1)
PCPERFORM 'passwd'
END
```
You want to have your users in your main account, and you want them (at least some of them) to have colon prompt access. What you DO NOT want is for someone to accidentally (or maliciously) use a command like DELETE PERSON. One answer is to restrict dangerous verbs to a select few technical folks who really need to be able to use those verbs. This can be done using remote VOC entries (see “Using Unidata - Chapter 4 - The Unidata VOC file” for information on creating remote VOC entries).

One of the features of remote VOC entries is that you can call a subroutine to evaluate if a user should be allowed to execute the command in question. This subroutine is what we use for most of our restricted verbs. It simply checks the current user against a list of ‘approved’ staff, which is hard-coded into the program.

```
SUBROUTINE STAFF.RESTRICTED(OK)
STAFF = ""
STAFF<1> = "jszloh"
STAFF<1> = "dkilbane"
STAFF<1> = "qjaamey"
STAFF<1> = "lartim"
STAFF<1> = "dprez"
STAFF<1> = "mshadrak"
STAFF<1> = "ragnew"
STAFF<1> = "srobinso"
STAFF<1> = "datatel"
STAFF<1> = "udmsmgr"
USERNAME = @LOGNAME
OK = 0
STAFF.COUNT = DCOUNT(STAFF,@FM)
FOR IDX = 1 TO STAFF.COUNT
  IF USERNAME = STAFF<IDX> THEN OK = 1
NEXT IDX
RETURN
END
```

Following is an example of a remote VOC entry that makes use of this subroutine. RESTRICTED.VERBS is a file that contains the original DELETE verb. We have set up similar remote VOC records for many other verbs such as BASIC, ELE, CATALOG, CLEAR.FILE etc…

```
Top of "DELETE" in "VOC", 4 lines, 58 characters.
*---: P
001: R BWC:DVP 12/7/98
002: RESTRICTED.VERBS
003: DELETE
004: STAFF.RESTRICTED
Bottom.
*---:
```
AE Security

Of course there is always a special case, and for UniData security, AE is that special case. Unlike verbs like DELETE or COPY there may be some end users who need to edit records. This may be simply creating and modifying their own paragraphs, or could even be files like GL.IMPORTS. However, if we give them full access to AE, then all the security above is useless since they can simply edit their way around any security given enough know-how. So what we need for AE is a way to limit what they can do with it. Fortunately enough, AE has some built-in security capabilities that can do quite a lot for you. There are two subroutines which, if globally cataloged, allow you to do a variety of checks both at the time the user starts up the editor (or loads a record in an editor session), and just after the user has exited the editor (or unloads a record). Complete (or as complete as it gets) documentation is available in $UDTHOME/sys/AE_SECURITY. The I_AE_SECURITY insert is especially helpful. Following are the two subroutines we have set up at Baldwin-Wallace College.

PREPROG_AE

```plaintext
SUBROUTINE PREPROG_AE(MAT SECURITY)
 *
* BWC:DPREZ December 21, 2000
* Written by Don Prezioso for Baldwin-Wallace College
* This subroutine is called when AE is started to check if the user
* is allowed to edit the file in question, and sets appropriate
* limits on loading/unloading records.
*
* The program checks what files a user is allowed to edit by looking
* at fields 30 and 31 of the UT.OPERS file. Field 30 holds a multi-valued
* list of the files the user may edit, while field 31 holds a list of
* the dictionaries they may edit.
*
* Documentation on AE security can be found in $UDTHOME/sys/AE_SECURITY
*
$INCLUDE @UDTHOME/sys/AE_SECURITY/I_AE_SECURITY
*
EQUATE AE.FILES TO OPERS.REC<30>; * Allowed files
EQUATE AE.DICTS TO OPERS.REC<31>; * Allowed dictionaries
*
* Get the user's UT.OPERS record to find out what files and dictionaries
* they are allowed to edit.
*
OPERS.KEY = OCONV(@LOGNAME,"MCU")
OPEN ",","UT.OPERS" TO OPERS.FILE ELSE
   PRINT "UNABLE TO OPEN UT.OPERS FILE"
END
READ OPERS.REC FROM OPERS.FILE, OPERS.KEY ELSE
   PRINT "UNABLE TO READ OPERATOR RECORD FOR ":OPERS.KEY
END
*
FILE.COUNT = DCOUNT(AE.FILES,@VM)
DICT.COUNT = DCOUNT(AE.DICTS,@VM)
```

March 28, 2001
IF SEC.SET = "" THEN SEC.SET = 1
IF SEC.CALL2.TYPE = "" THEN
    GOSUB PRIMARY.CALL
END ELSE
    GOSUB SECONDARY.CALL
END
RETURN
*
*************
PRIMARY.CALL:
*************
*
* Called before AE reads a record
* IF SEC.DICT.FLG EQ 1 THEN
    CHECK.LIST = AE.DICTS
END ELSE
    CHECK.LIST = AE.FILES
END
*
* If the user is allowed to access all files then give them full access
* but still do the post-edit checking if they are editing a VOCLIB file.
* IF CHECK.LIST = "*ANY*" THEN
    SEC.READ.FLG = 1
    SEC.WRITE.FLG = 1
    SEC.DELETE.FLG = 1
    SEC.LOAD.FLG = 1
    SEC.UNLOAD.FLG = 1
    SEC.XSEQ.FLG = 1
    SEC.XCOM.FLG = 1
    IF SEC.FN[6] = "VOCLIB" THEN
        SEC.POST.FLG = 2
    END
END
*
* If this file is in the list of allowed files for this user, allow
* them with some limitations on loading and unloading. Always do the
* post-edit checking.
* IF LOCATE SEC.FN IN CHECK.LIST<1,1> SETTING TEMP THEN
    SEC.READ.FLG = 1
    SEC.WRITE.FLG = 1
    SEC.DELETE.FLG = 1
    SEC.LOAD.FLG = 2
    SEC.UNLOAD.FLG = 2
    SEC.XSEQ.FLG = 0
    SEC.POST.FLG = 2
END
*
* Tell the user if they are not being allowed to edit the file.
* IF SEC.READ.FLG NE 1 THEN
    IF SEC.DICT.FLG EQ 1 THEN
        PRINT "You are not authorized to edit the ":SEC.FN:"
    END ELSE
        PRINT "You are not authorized to edit the ":SEC.FN:"
END
* Called before AE loads/unloads a record *
* IF SEC.DICT2.FLG EQ 1 THEN
   CHECK.LIST = AE.DICTS
END ELSE
   CHECK.LIST = AE.FILES
END
*
* If the user can edit any file, then they can also load and unload *
* on any file.
*
* IF CHECK.LIST = "*ANY*" THEN
   SEC.OK2.FLG = 1
END
*
* If the user can edit this file, then they are also allowed to load *
* and unload records in this file.
*
LOCATE SEC.FN2 IN CHECK.LIST<1,1> SETTING TEMP THEN
   SEC.OK2.FLG = 1
END
*
* IF SEC.OK2.FLG NE 1 THEN
   IF SEC.DICT2.FLG EQ 1 THEN
      PRINT "You are not authorized to load/unload the ":SEC.FN:" dictionary"
   END ELSE
      PRINT "You are not authorized to load/unload the ":SEC.FN:" file"
   END
END
*}
RETURN
SUBROUTINE POSTPROG_AE(MAT SECURITY)
 *
 * BWC:DPREZ December 21, 2000
 * Written by Don Prezioso for Baldwin-Wallace College
 * This subroutine is called when AE is done editing a record and has
 * just written the record to the file. So far the only thing that we
 * is to make sure records written to ...VOCLIB files are sentences or
 * paragraphs, and create appropriate remote VOC entries for them.
 *
 $INCLUDE @UDTHOME/sys/AE_SECURITY/I_AE_SECURITY
 *
 IF SEC.SET = "" THEN SEC.SET = 1
 *
 * Check ...VOCLIB records.
 *
 IF SEC.FN[6] = "VOCLIB" AND SEC.DICT.FLG NE 1 THEN
 AE.REC = ""
 VOC.REC = ""
 *
 * Make sure the record just written is either a paragraph or sentence.
 * If it is not, add a line at the top that makes it a paragraph. This
 * may cause the paragraph to be invalid (if it was a Verb or Subroutine
 * it will not make sense as a paragraph), but will leave the record where
 * the user can do something with it as opposed to simply deleting it.
 *
 OPEN "",SEC.FN TO AE.FILE THEN
 READ AE.REC FROM AE.FILE, SEC.ID THEN
  IF AE.REC[1,2] NE "PA" AND AE.REC[1,1] NE "S" THEN
   GOSUB FIX.PARAGRAPH
   WRITE AE.REC ON AE.FILE, SEC.ID
 END
 END
 *
 * Handle the creation (or deletion) of an associated remote VOC record.
 * This way we can let users create paragraphs and sentences in the VOCLIB
 * without letting them touch the VOC. Of course, we need to make sure
 * they can't create a remote VOC (or delete one) that would overwrite an
 * existing VOC entry that is not pointing to their paragraph.
 *
 OPEN "";"VOC" TO VOC.FILE THEN
 READ VOC.REC FROM VOC.FILE, SEC.ID THEN
 *
 * If we have created (or deleted) a VOCLIB entry where a VOC record
 * by the same name exists and does not point to the VOCLIB entry, then
 * just print an appropriate message. Do NOT overwrite or delete the
 * existing VOC entry.
 *
  IF VOC.REC<1>[1,1] NE "R" OR VOC.REC<2> NE SEC.FN |
   OR VOC.REC<3> NE SEC.ID THEN
    IF AE.REC = "" THEN
     PRINT "Could not remove VOC entry!"
     PRINT "Another VOC item existed with that name"
    END ELSE

PRINT "Could not create VOC entry!"
PRINT "Another VOC item already exists with that name."
END
END ELSE
*
* The VOC entry was pointing to the record that was being edited, but
* the user has deleted the record (using the FD or some similar command)
* so now we should clean up by deleting the remote VOC entry as well.
*
IF AE.REC = "" THEN
  PRINT "Deleting VOC entry for ":SEC.ID:"...
  DELETE VOC.FILE, SEC.ID
END
END
END ELSE
*
* The user has created a paragraph or sentence, and there is no VOC entry
* by the same name so we will create a remote pointer.
*
IF AE.REC NE "" THEN
  GOSUB MAKE.VOC.REC
  WRITE VOC.REC ON VOC.FILE, SEC.ID
END
END
END ELSE
  PRINT "Could not create VOC entry!"
  PRINT "Unable to open VOC file for writing!"
END
END
RETURN
*
**************
FIX.PARAGRAPH:
**************
*
* The user has attempted to create something other than a sentence or
* paragraph. Since we would not want them to be able to create Verbs
* or Catalog entries we need to alter the record they created so that
* it IS a paragraph.
* Of course, if they just forgot the first line this will take care of
* that too.
*
PRINT "You can only make 'S'entences or 'PA'ragraphs in a VOCLIB!
PRINT "Modifying record to be a 'PA'ragraph...
TEMP = "PA BWC:" :OCONV(@LOGNAME,"MCU"): " "
TEMP := OCONV(DATE(),"D4-"
TEMP<-1> = AE.REC
AE.REC = TEMP
RETURN
*
**************
MAKE.VOC.REC:
**************
*
* The user has created a new paragraph or sentence, so we need to create
* a remote VOC entry so they can use it.
PRINT "Creating VOC entry for ":SEC.ID:"...
TEMP = "R BWC:" :OCONV(@LOGNAME,"MCU"):" 
TEMP := OCONV(DATE(),"D4-")
TEMP := " Created by AE_POSTPROG security routine"
VOC.REC<1> = TEMP
VOC.REC<2> = SEC.FN
VOC.REC<3> = SEC.ID
RETURN
*
END

To make these work, you need to globally catalog the subroutines, and create environment variables (probably in /etc/profile) named PREPROG_AE_UDT and POSTPROG_AE_UDT that are set to the names of the subroutines you have cataloged.
System Stuff

BATCH

You want to do something like the PHANTOM command so you can run commands in the background, except you don't want them to run right now, you want it to wait until, say 6:00PM after all the data entry for the day is done. Here is a program that does just that. This program works like PHANTOM, but it actually uses the unix ‘at’ command to run at a later time. Now you can use a command like:

    BATCH 18:00 LIST STUDENTS WITH ... LPTR

But wait! There’s MORE!!! An almost identical subroutine by the name of S_UNIX_BATCH_PROC is provided by Datatel which can be modified for your system. Using this subroutine, any Envision procedure that presents you with the Phantom mode definition screen can also be delayed. Just use the ‘B’ option for phantom type instead of ‘P’ and you can enter the time you want the job to run.

One caveat to all of this is that while PHANTOM does not use up an additional Unidata license, BATCH does. This is not usually a problem however since the whole point is to schedule things to run when most users are off the system.

Here is the source code for BATCH:

```
PROGRAM BATCH
*
* Written by Don Prezioso for Baldwin-Wallace College 1/27/99
* This program was written to allow the use of the 'B' option
* on the Phantom mode definition screen. It can also be used from
* the colon prompt by using a command of the form:
* :BATCH start sentence . . .
* where 'start' is the start time desired in the form HH:MM and
* 'sentence . . .' is any normal sentence.
*
* Modified by Don Prezioso on 8/23/99 to allow the full capability of
* the 'at' command. In place of the HH:MM format of the start time
* (which is still supported) the user may enter any flags or arguments
* acceptable to the at command encapsulated in quotes (single or double).
*
* Examples:
*
* :BATCH 18:00 REG.ENROLL.STATS 1999FA
* This will run the REG.ENROLL.STATS paragraph for 1999FA at 6:00PM
*
* :BATCH "-t 08301800" REG.ENROLL.STATS 1999FA
* This will run the same paragraph at 6:00PM on 8/30
*
OPEN '','UFD' TO F.UFD ELSE STOP "Unable to open the UFD file"
*
* Parse the command line to get the start time and the sentence to
* be executed later.
```
* BATCH.COMMAND = @SENTENCE
BATCH.START.TIME = FIELD(BATCH.COMMAND," ",2,1)
BEGIN CASE
   CASE BATCH.START.TIME[1,1] = ""
      BATCH.START.TIME = FIELD(BATCH.COMMAND," ",2,1)
      BATCH.SENTENCE = TRIMF(FIELD(BATCH.COMMAND," ",3,999))
   CASE BATCH.START.TIME[1,1] = ""
      BATCH.START.TIME = FIELD(BATCH.COMMAND," ",2,1)
      BATCH.SENTENCE = TRIMF(FIELD(BATCH.COMMAND," ",3,999))
   CASE 1
      BATCH.SENTENCE = FIELD(BATCH.COMMAND," ",3,999)
   END CASE
BATCH.NAME = FIELD(BATCH.SENTENCE," ",1,1)
* IF BATCH.START.TIME MATCHES "2N:2N" THEN
* BATCH.START.TIME = ICONV(BATCH.START.TIME,"MT")
* BATCH.START.TIME = OCONV(BATCH.START.TIME,"MT")
* END
BATCH.START.TIME = OCONV(BATCH.START.TIME,"MCL")
* If the command is supposed to run now, just for grins, run it in the
* batch queue. This way we can set the batch queue to run at a lower
* priority.
* BEGIN CASE
   CASE BATCH.START.TIME = "now"
      BATCH.SCRIPT = "at -q b now <<eob"
   CASE 1
      BATCH.SCRIPT = "at ":BATCH.START.TIME:" <<eob"
   END CASE
* This is where we do the actual command that gets run by at.
* The COMO file created is not very unique, but that way it will be
* less of a problem keeping it cleaned up.
* BATCH.SCRIPT<-1> = "udt > /dev/null 2>&1 <<eoi"
BATCH.SCRIPT<-1> = "TERM ,0"
BATCH.SCRIPT<-1> = "COMO ON ":BATCH.NAME
BATCH.SCRIPT<-1> = "DATE"
BATCH.SCRIPT<-1> = BATCH.SENTENCE
BATCH.SCRIPT<-1> = "DATE"
BATCH.SCRIPT<-1> = "COMO OFF"
BATCH.SCRIPT<-1> = "QUIT"
BATCH.SCRIPT<-1> = "eoi"
* Now finish the batch submission process.
* BATCH.SCRIPT<-1> = "eob"
* Submit the job
* WRITE BATCH.SCRIPT ON F.UFD,BATCH.NAME
PCPERFORM "ksh ":BATCH.NAME
DELETE F.UFD,BATCH.NAME
END
And here is the code for S.UNIX.BATCH.PROC as it has been modified for our use at Baldwin-Wallace College. As you can see, the core of the subroutine is very similar to what is done in BATCH.

```plaintext
SUBROUTINE S.UNIX.BATCH.PROC(A.JOB.NAME, A.START.TIME, A.JOB.QUEUE, A.PRIORITY)
* Batch Process S_UNIX_BATCH_PROC generated by ENVISION Batch Generator -
* Date: 11:30:29 May 01 1998
* Logname: ALEX
*----------------------------------------------------------------------*
* The information contained herein is proprietary to and considered a *
* trade secret of Datatel Inc. and shall not be reproduced in whole or *
* in part without the written authorization of Datatel, Inc. *
*----------------------------------------------------------------------*
$INSERT I_COMMON FROM UT.INSERTS
$INSERT I_WVAR_EQU FROM UT.INSERTS
$INSERT I_COMMAND_EQU FROM UT.INSERTS
DIM AR.CWVAR(1,CWVAR.SIZE)
MAT AR.CWVAR = ""
MAX.FIELDS = 1; CRNT.FLD.INDX = 1
$INSERT I_PRCS.CTL FROM UT.SOURCE
$INSERT I_VALCODES FROM UT.SOURCE
CRNT.PROCESS = "S_UNIX_BATCH_PROC"
* Process Fields
EQUATE V.BATCH.QUEUE TO AR.FVAR(2)
DIM AR.FVAR(2)
AR.FVAR.SIZE = 2
MAT AR.FVAR = ""
MAT AR.VWVAR = ""
DIM AR.AEVAR(1); MAT AR.AEVAR = ""
DIM AR.DFVAR(1); MAT AR.DFVAR = ""
PROCESS.APPL.TREE = "UT"
$INSERT I_BATCH_INIT FROM UT.INSERTS
IF NOT(R.PHANTOM.QUEUES) THEN
   R.PHANTOM.QUEUES = ''; KV.VALCODES = "PHANTOM.QUEUES"
   CALL @MIO.READ.RECORD(MIO.READ,FV.VALCODES,KV.VALCODES,R.PHANTOM.QUEUES)
END
$INSERT I_BATCH_PRCSR FROM UT.INSERTS
* Batch Program Code from the INSERT File
USER.MAIN.CODE:
* S_UNIX_BATCH_PROC - Unix Batch Interface
*
* Arguments:
* A.JOB.NAME = Name of paragraph to run
* A.START.TIME = Time to start paragraph
* A.JOB.QUEUE = Batch queue to use (unimplemented)
* A.PRIORITY = O/S Priority level (unimplemented)
*
* $$ :UT.INSERTS T_LOCAL.DEBUG:
* $USING UT,PROCGEN
IF INDEX(UT.DEBUG.STRING,"PROCGEN",1) THEN
   LOCAL.DEBUG = 1
END ELSE
   LOCAL.DEBUG = 0
END
```

March 28, 2001
Note: This is the simplest interface to the Unix batch processor possible, and does not take advantage of any queue or priority capabilities of the Unix implementation. For purposes of demonstration, code has been included to validate the job queue against the PHANTOM.QUEUES valcode table.

\[
\begin{align*}
V.BATCH.QUEUE &= A.JOB.QUEUE \\
BATCH.QUEUE.ACTION1 &= ""; BATCH.QUEUE.ACTION2 = ""
\end{align*}
\]

IF V.BATCH.QUEUE NE "" THEN
   LOCATE V.BATCH.QUEUE IN R.PHANTOM.QUEUES<4,1> SETTING CODE.VALIDATED ELSE
   CODE.VALIDATED = ""
   IF CODE.VALIDATED THEN
      BATCH.QUEUE.ACTION1 = R.PHANTOM.QUEUES<3,CODE.VALIDATED>
      BATCH.QUEUE.ACTION2 = R.PHANTOM.QUEUES<5,CODE.VALIDATED>
   END ELSE
      ERROR.OCCURRED += 1
      MSG = INSERT(MSG,ERROR.OCCURRED,0,0, V.BATCH.QUEUE:" is not in the "PHANTOM.QUEUES:" table.")
   END
   END

X.BATCH.ACTION = BATCH.QUEUE.ACTION1
X.BATCH.OS.QUEUE = BATCH.QUEUE.ACTION2

BEGIN CASE
   CASE A.JOB.QUEUE = "B" ;* Use default queue
      X.JOB.QUEUE = ""
   CASE X.BATCH.OS.QUEUE NE "" ;* Use O/S queue name instead
      X.JOB.QUEUE = X.BATCH.OS.QUEUE
   CASE 1
      X.JOB.QUEUE = A.JOB.QUEUE
   END CASE
   IF INDEX(A.START.TIME, ":",1) THEN A.START.TIME = ICONV(A.START.TIME,"MT")
   JOB.START = OCONV(A.START.TIME,"MT")
   IF LEN(JOB.START) THEN
      JOB.SCRIPT = "at ":JOB.START:" <<eob"
   END ELSE
      JOB.SCRIPT = "at -q b now <<eob"
   END

* The preceding lines were strictly setup for submitting the batch.
* These lines are what will actually be run when the batch starts.
* The references to TERM are for eliminating screen paging, and are dependent upon the version of Unidata that is running. Prior to Unidata 2.3, "TERM ,0" doesn't work, so the next best solution is to set the page size as high as possible.
JOB.SCRIPT<-1> = "udt > /dev/null 2>&1 <<eoi"
IF SN.RELLEVEL LT "2.3" THEN
  JOB.SCRIPT<-1> = "TERM ,32767"
END ELSE
  JOB.SCRIPT<-1> = "TERM ,0"
END
JOB.SCRIPT<-1> = "COMO ON ":A.JOB.NAME
JOB.SCRIPT<-1> = "DATE"
JOB.SCRIPT<-1> = A.JOB.NAME
JOB.SCRIPT<-1> = "DATE"
JOB.SCRIPT<-1> = "COMO OFF"
JOB.SCRIPT<-1> = "QUIT"
JOB.SCRIPT<-1> = "eoi"
*  Now finish the batch submission process.
*  
JOB.SCRIPT<-1> = "eob"
*  
WRITE JOB.SCRIPT ON F.UFD,A.JOB.NAME
PCPERFORM "ksh ":A.JOB.NAME CAPTURING X.RESULTS
*  
*  Clean up
*  
IF LOCAL.DEBUG THEN
  CRT "Submitted ":A.JOB.NAME:
  IF LEN(A.START.TIME) THEN CRT " starting at ":JOB.START:
  CRT ""
  CRT X.RESULTS
END ELSE
  DELETE F.UFD,A.JOB.NAME
END
*  
RETURN
* END OF BATCH PROGRAM
*----------------------------------------------------------------------*
EXIT.PROCESS:
ABORT.PROCESS: ;* TOTAL ABORT OF PROCESS WITH NO RECOVERY
RETURN
*----------------------------------------------------------------------*
FILE.OPENS: ;* Open all of the Process files
RETURN
*----------------------------------------------------------------------*
PARSE.RECORDS: ;* Parse the Records
RETURN
*----------------------------------------------------------------------*
REBUILD.RECORDS: ;* Rebuild out All Records
RETURN
*----------------------------------------------------------------------*
$INSERT I_BCANCL_LOCK FROM UT.INSERTS
*----------------------------------------------------------------------*
I figured most of this out by looking at a similar routine by Bill Edgette (Our Lady of the Lake University) which goes into greater detail and allows for multiple batch queues that run at a variety of times and priorities. This can be found at: ftp://ftp.cedarville.edu/pub/unidata/ollusa/PHANTOM.TXT. Of course there are all kinds of things you could add in to this to do special logging, additional processing, etc.. This is just a very simple approach which does what we need it to.
CATALOG.CUSTOM.SOURCE

This paragraph is stored in CUSTOM.SOURCE with a remote VOC entry that points to it. Any time we create a custom program or subroutine it is added to this paragraph so that any time we load a release, we can re-compile and re-catalog all our custom programs. For routines that are not Datatel STANDARD.FORMS programs, we just need to compile and catalog them. For routines that originate in STANDARD.FORMS (or any other Datatel supplied source) we need to add the step of overwriting Datatel’s object code as well. This is done so that if VOC pointers are accidentally created which point to the Datatel object code instead of the global catalog space, it will still be our version.

```
PA BWC:DVP Paragraph to re-catalog custom programs and subroutines
*
COMO ON CATALOG.CUSTOM.SOURCE
*
BASIC CUSTOM.SOURCE LOGIN.EXECUTE
UDT.CAT CUSTOM.SOURCE LOGIN.EXECUTE
*
BASIC CUSTOM.SOURCE DFLTED
UDT.CAT CUSTOM.SOURCE DFLTED
COPY FROM CUSTOM.SOURCE TO UT.OBJ _DFLTED OVERWRITING
*
BASIC CUSTOM.SOURCE S_UNIX_BATCH_PROC
UDT.CAT CUSTOM.SOURCE S_UNIX_BATCH_PROC
COPY FROM CUSTOM.SOURCE TO UT.OBJ _S_UNIX_BATCH_PROC OVERWRITING
*
BASIC CUSTOM.SOURCE S.PRINT.BPO.BW
UDT.CAT CUSTOM.SOURCE S.PRINT.BPO.BW
*
BASIC CUSTOM.SOURCE S.PRINT.PO.BW
UDT.CAT CUSTOM.SOURCE S.PRINT.PO.BW
*
BASIC CUSTOM.SOURCE S.PRT.AP.CHK1.DEW
UDT.CAT CUSTOM.SOURCE S.PRT.AP.CHK1.DEW
COPY FROM CUSTOM.SOURCE TO FORM.IMAGES S.PRT.AP.CHK1.BW.IMAGE OVERWRITING
*
BASIC CUSTOM.SOURCE S.1099.FORM
UDT.CAT CUSTOM.SOURCE S.1099.FORM
COPY FROM CUSTOM.SOURCE TO CF.OBJ _S.1099.FORM OVERWRITING
*
BASIC CUSTOM.SOURCE S.PRINT.CASH.RCPT
UDT.CAT CUSTOM.SOURCE S.PRINT.CASH.RCPT
COPY FROM CUSTOM.SOURCE TO ST.OBJ _S.PRINT.CASH.RCPT OVERWRITING
*
*** You get the idea...
*
COMO OFF
*```

March 28, 2001
dmictl

You want to be able to stop the DMI listener (or multiple listeners), either to reboot, or to do some sort of file maintenance. Or you want to be able to start the listener when the system first boots, or when the earlier file maintenance is done. Or you want to be able to ‘bounce’ the listener because you have made some changes to your WebAdvisor setup. You want to be able to do these things without having to remember the proper paths and commands to do it all. That is what this script is for. The dmictl script is designed to be used either interactively, from a script (like lock.logins or unlock.logins), or from a cron or at job. The script is run like a command with at least one argument. The syntax is:

```
_dmictl action [account1 [account2 [accountN]]]
```

Action is one of; start, stop, or restart (up, down, or bounce if you prefer). Account is currently either ‘live’, ‘test’, or ‘all’. If no account is given, then all accounts are acted upon. The account names are hard coded into the main portion of the script; so adding another account is just a matter of adding a few lines there to let the script know where to find the `dmiserverd` command and the PATH that is needed. When running the `dmiserverd` command, I use the full path for the `dmiserverd` command is so that the script is able to tell the difference between my test and live listeners.

Since the script checks to see if a listener is running before it attempts to start or stop it, the command may not be used without knowing if a given listener is already running. For example, if you just want to stop all the listeners you can simply use the command:

```
dmictl stop
```

The script will attempt to stop each of the possible listeners, and if it is not running, the script will just display a message to that effect and go on to the next listener. The same is true when starting listeners. If you use the command to start all the listeners, it will first check if a given listener is running, and if so will just display a message to let you know and go on to the next one.

One thing that is sort of strange about this script is how I go about starting the `dmiserverd` process. Rather that just doing the command that Datatel uses in their documentation, I am submitting the command as an at job using a ‘here’ file (designated by `<<EOJ`). I ran up against a problem where if I started more than one listener from within a script, then later tried to stop only one of the listeners, both would stop. If I started them separately, no problem, except that I wanted to be able to start all of them at one time for things like rebooting or file maintenance. By using the at job technique, they were actually being run separately, but only one command was called. I would consider this a bug in the `dmiserverd` process, but since it is unlikely to be fixed real soon I found a way to avoid the problem.

A note here about creating scripts that can be run from a cron job. You have to be careful to set up any necessary environment variables, especially `UDTHOME`, `UDTBIN`, and `PATH`, since anything run by cron DOES NOT get any of the environment you may have set up in `/etc/profile`, or the `.profile` of the user involved. If you have a script that runs just fine when you run it interactively, and also works well as an at job, but seems to die for no apparent reason when run from a cron job, this is something you will want to check.
#!/bin/ksh
#
# BWC:DVP September 27, 2000 Don Prezioso
#
# This script was written by Don Prezioso for Baldwin-Wallace
# College to control the dmiserverd process. It is run with the
# following arguments:
#
#   stop   - Stop the currently running dmiserverd process
#   start  - Start the dmiserverd process
#   restart - Stop the current dmiserverd, and start a new one
#   bounce - Same as restart
#
function dmistart {

# This function starts a DMI listener. It uses an 'at' command to start the
# listener rather than just running the command here because of a bug in
# the listener. If more than one listener is started from one script, and
# you later try to kill one of the listeners, all of them will die. For
# some reason I haven't been able to determine, starting them in separate
# environments gets around the problem. This may very well be an AIX specific
# problem.
#
# After submitting the job, the function checks every so often until it
# sees that the listener has in fact started and displays a message. If
# it checks a number of times and still cannot see the listener, then it
# gives up and displays a failure message.
#
  dmi_running=$( ps -ef | grep "$dmicmd" | grep -v grep )
  if [[ -z $dmi_running ]]; then
    echo "Starting DMI listener ($dmiacct)...
    at now > /dev/null 2>&1 <<EOJ
    /bin/nohup $dmicmd > dmiserverd.log 2>&1 &
    sleep 15
    EOJ
    dmi_running=
    let check_count=0
    while [[ -z $dmi_running && $check_count -le 10 ]]; do
      sleep 2
      dmi_running=$( ps -ef | grep "$dmicmd" | grep -v grep )
      let check_count=check_count+1
    done
    if [[ -n $dmi_running ]]; then
      echo "DMI listener ($dmiacct) was already running."
      echo $dmi_running
    else
      echo "DMI listener ($dmiacct) failed to start! - Must be started manually!"
    fi
    else
      echo "DMI listener ($dmiacct) was already running."
  fi
}
#
function dmistop {
#
# This function stops a DMI listener.
#
# After sending the kill signal, the function checks every so often until
# it sees that the listener has in fact stopped and displays a message. If
# it checks a number of times and the listener is still running, then it
# gives up and displays a failure message.
#
dmi_running=$( ps -ef | grep "$dmicmd" | grep -v grep )
if [[ -n $dmi_running ]]; then
    echo "Stopping DMI listener ($dmiacct)"
    kill $( ps -ef | grep "$dmicmd" | grep -v grep | cut -c 9-15 )
    dmi_running=$( ps -ef | grep "$dmicmd" | grep -v grep )
    let check_count=0
    while [[ -n $dmi_running && $check_count -le 10 ]]; do
        sleep 2
        dmi_running=$( ps -ef | grep "$dmicmd" | grep -v grep )
        let check_count=check_count+1
    done
    if [[ -z $dmi_running ]]; then
        echo "DMI listener ($dmiacct) successfully stopped!"
    else
        echo "DMI listener ($dmiacct) failed to stop! - Must be stopped manually!"
    fi
else
    echo "DMI listener ($dmiacct) was not running."
fi
}
#
# Main Script
#
UDTBIN=/usr/udt/bin ; export UDTBIN
UDTHOME=/usr/udt ; export UDTHOME
BASEPATH=/datatel/bin:/usr/local/bin:$UDTBIN:/usr/bin:/etc:/usr/sbin:/usr/ucb:/sbin ; export BASEPATH
PATH=$BASEPATH ; export PATH
#
# Make sure we are running as 'datatel' and if not, use 'su'
# to re-run the command.
#
if [[ $USER != "datatel" ]]; then
    su - datatel -c /datatel/bin/dmictl $@
else
    cd /datatel
#
# Check which account should be acted on and allow for
# acting on all accounts.
#
    action_arg=$1
    shift
    account_args=$@
    if [[ $account_args = "all" || -z $account_args ]]; then
        account_args="live test"
    fi
# Now loop through each account and take the appropriate action.
#
for dmiacct in $account_args ; do
  case $dmiacct in
    live )
      dmicmd=/datatel/live/dmilive/dmiserverd
      PATH=/datatel/live/dmilive:$BASEPATH ; export PATH
      cd /datatel/live/dmilive
      ;;
    test )
      dmicmd=/datatel/work/dmitest/dmiserverd
      PATH=/datatel/work/dmitest:$BASEPATH ; export PATH
      cd /datatel/work/dmitest
      ;;
    * )
      echo "$dmiacct is not valid... Aborting..."
      dmiacct=abort
      ;;
  esac
#
# Now find out what we are doing, or if we are aborting that account.
#
if [[ $dmiacct != "abort" ]] ; then
  case $action_arg in
    stop | down ) dmistop ;;
    start | up ) dmistart ;;
    restart | bounce ) dmistop ; dmistart ;;
    * )
      action_arg=abort ;;
  esac
fi
#
if [[ $action_arg = "abort" ]] ; then
  echo "Usage: dmictl action account"
  echo " where action is one of the following:"
  echo " start or up - Stop a dmiserverd process"
  echo " stop or down - Start the dmiserverd process"
  echo " restart or bounce - Stop dmiserverd, and start a new one"
  echo " and account is one of the following:"
  echo " live - Colleague live account"
  echo " test - Colleague test account"
  echo " all - All accounts"
fi
done
fi
weekly.analysis

This is a very simple script that we use to run our weekly file analysis. We use FAST instead of WUFA, but the same principles apply. I have included it here because it is an example of how you can set up a script to be run as a cron job that runs a UniData processes. Note that as we are careful to define UDTBIN, UDTHOME, and PATH so the script will run properly as a cron job.

The first thing the script does is to stop any DMI listeners that are running. After running the command to stop the listeners, the process sits and waits for 6 minutes to let the UniData processes exit cleanly. For some reason, these processes (as reported by listuser) seem to hang around for a little while after the process (as reported by ps -ef) has died.

Then the script checks for regular and phantom UniData processes which can cause real trouble when resizing files. If everything checks out, then we run the file analysis using a ‘here’ file. A ‘here’ file is when you run a command from within a script, and rather than having the input for that command come from the terminal, we specify the input within the script. In this case, the important part of the script is the following:

```bash
$UDTBIN/udt >> $logfile 2>&1 <<EOF
FAST.BATCH WEEKLY.ANALYSIS (INIT)
QUIT
EOF
```

Everything between <<EOF and the line which contains only EOF is taken as input for the udt command. The only tricky thing here is that you must be able to blindly enter the correct commands to do what is needed without seeing any screen output. The easiest way to handle that is to put everything in a paragraph so you are less likely to leave something out.
#!/bin/ksh
#
# This script does regular file maintenance in the colleague accounts
# UDTBIN=/usr/udt/bin ; export UDTBIN
# UDTHOME=/usr/udt ; export UDTHOME
# PATH=/usr/local/bin:/datatel:$UDTBIN:$PATH ; export PATH
operators="dprez@bw.edu"
logfile=/datatel/weekly.analysis.log
>
# First stop the dmi listener since it will show up as a logged in
# process when listuser does it's check
# /datatel/bin/dmictl stop >> $logfile 2>&1
sleep 360
#
# Check to see if anyone is logged in. If so, do not run FAST.
# if [[ -z $(listuser | grep -E "udt|phant") ]] ; then
# # Run FAST WEEKLY.ANALYSIS
# cd /usr/local/fast >> $logfile 2>&1
# $UDTBIN/udt >> $logfile 2>&1 <<EOF
# FAST.BATCH WEEKLY.ANALYSIS (INIT)
# QUIT
# EOF
# # If someone was logged in, print a message, and show who.
# else
# print "Users logged in - FAST aborted!" >> $logfile 2>&1
# listuser >> $logfile 2>&1
#fi
#
# Restart the dmi listener
# /datatel/bin/dmictl start >> $logfile 2>&1
# # Send output to someone for checking
# mailx -s "GRANITE File Analysis Log" $operators < $logfile
MAKE.FILE.STATIC

After years of creating all my files as dynamic files, I decided that static files were the way to go (that is a story for a whole other session...). Unidata at 4.1.?? can change files from dynamic to static using the RESIZE command, but alas, that only works if the file is in the same directory as you are when you run Unidata (a local file). The manual method for changing a dynamic file into a static file involves moving the file to a temporary name, re-creating the file as a new static file, and copying all the records from the temporary file into the new file. Then you still need to delete the temporary file once you are sure everything went well. This program will convert most normal files from dynamic to static with a single command. It does depend on you having run WEEKLY.UDT.FILE.ANALYSIS first. By using the data from WUFA, you can create a paragraph that will convert almost all of your dynamic files to static all at once.

PROGRAM MAKE.FILE.STATIC
* 
* This program was written by Don Prezioso for Baldwin-Wallace College
* to change dynamic files into static files as a single command.
* 
* Get the name of the file to be converted.
* 
COMMAND.LINE = @SENTENCE
COMMAND.LINE = TRIM(COMMAND.LINE)
WORD.COUNT = DCOUNT(COMMAND.LINE,' ') 
IF WORD.COUNT = 2 THEN
   FILE.NAME = FIELD(COMMAND.LINE,' ',2,1)
END ELSE
   STOP "*** USAGE: MAKE.FILE.STATIC file.name"
END
* 
* Open files
* 
OPEN '"','VOC' TO F.VOC ELSE STOP "*** UNABLE TO OPEN VOC FILE'
OPEN '"','UDT_GUIDE' TO F.UDT_GUIDE ELSE STOP "*** UNABLE TO OPEN UDT_GUIDE FILE'
* 
* Read the VOC entry for this file and make sure it is nothing special
* like a synonym, or with the dictionary in a different directory from
* the data.
* 
READ R.VOC FROM F.VOC, FILE.NAME ELSE
   STOP "*** UNABLE TO READ VOC RECORD FOR ":FILE.NAME
END
FILE.PATH = R.VOC<2>
DICT.PATH = R.VOC<3>
PATH.LEVELS = DCOUNT(FILE.PATH,'/')
DIRECTORY.PATH = FIELD(FILE.PATH,'/','1', (PATH.LEVELS - 1)):'/
IF FILE.PATH NE DIRECTORY.PATH:FILE.NAME THEN
   STOP "*** FILE PATH IS NOT "':DIRECTORY.PATH:FILE.NAME:" - ABORTING"
END
IF DICT.PATH NE DIRECTORY.PATH:"D_"':FILE.NAME THEN
   STOP "*** DICT PATH IS NOT "':DIRECTORY.PATH:"D_"':FILE.NAME:" - ABORTING"
END
* 
* Read the UDT_GUIDE record for the data portion of the file and make sure
* we are dealing with a dynamic file.
* 
READ R.UDT_GUIDE FROM F.UDT_GUIDE, FILE.PATH ELSE
   STOP "*** UNABLE TO READ UDT_GUIDE RECORD FOR "':FILE.PATH:" - ABORTING"
END
OPEN '',FILE.NAME TO F.CHECK.FILE THEN IF FILEINFO(F.CHECK.FILE,3) NE 3 THEN STOP "*** ":FILE.NAME:" IS NOT A DYNAMIC FILE - ABORTING" END CLOSE F.CHECK.FILE END ELSE STOP "*** UNABLE TO OPEN ":FILE.NAME END

* Figure out a reasonable modulo for the static file.
* KEYDATA.SIZE = R.UDT_GUIDE<27> + R.UDT_GUIDE<31> + 9
FILE.SIZE = KEYDATA.SIZE * R.UDT_GUIDE<22>
NEW.MODULO = INT(FILE.SIZE / 4000) + 7
* Move the old file to a temporary name, create the new (static) version
* of the file, and copy all the records from the temporary file to the
* new file, then delete the temporary file.
*
EXECUTE "SELECT ":FILE.NAME
ORIGINAL.COUNT = @SYSTEM.RETURN.CODE CLEARSELECT TEMP.FILE.NAME = FILE.NAME":.TEMP"
TEMP.FILE.PATH = FILE.PATH":.TEMP"
TEMP.DICT.PATH = DICT.PATH":.TEMP"
PCPERFORM "mv ":FILE.PATH:" ":TEMP.FILE.PATH
PCPERFORM "mv ":DICT.PATH:" ":TEMP.DICT.PATH
R.VOC<2> = TEMP.FILE.PATH
R.VOC<3> = TEMP.DICT.PATH
WRITE R.VOC ON F.VOC, TEMP.FILE.NAME
EXECUTE "SELECT ":TEMP.FILE.NAME
TEMP.COUNT = @SYSTEM.RETURN.CODE
CLEARSELECT IF TEMP.COUNT NE ORIGINAL.COUNT THEN STOP "*** ERROR RENAMING ":FILE.NAME:" TO ":TEMP.FILE.NAME END
DELETE F.VOC, FILE.NAME
EXECUTE "CREATE.FILE ":FILE.PATH:" ":NEW.MODULO:" ,2"
EXECUTE "COPY FROM ":TEMP.FILE.NAME:" TO ":FILE.NAME:" ALL OVERWRITING"
EXECUTE "COPY FROM DICT ":TEMP.FILE.NAME:" TO DICT ":FILE.NAME:" ALL OVERWRITING"
EXECUTE "SELECT ":FILE.NAME
NEW.COUNT = @SYSTEM.RETURN.CODE
CLEARSELECT IF NEW.COUNT NE ORIGINAL.COUNT THEN STOP "*** ERROR COPYING DATA FROM ":TEMP.FILE.NAME:" TO ":FILE.NAME END
EXECUTE "DELETE.FILE ":TEMP.FILE.NAME
END
clean.utmp.c

If you ever had someone on an AIX system that showed up on a ‘finger’ listing, even though they were not logged in, this is the program for you. The problem is that, for one reason or another, when the user left the system, the utmp file (where AIX keeps track of logged in users) never got updated. In some cases, this is just a nuisance (you think someone is logged in, but they are not), but this can also cause some serious security problems. If one of these ‘ghosts’ is hanging around from, let’s say, one of YOUR previous sessions, and someone logs in and obtains the same port number, they may get YOUR access rights in Colleague. This small program cleans up any of these entries in the utmp file, removing the problem. Set this program as a cron job to run every 15 minutes or so and it will keep your system clean.

This script is VERY AIX specific! Use on other systems at your own risk!

```c
/*
Clean dead processes from UTMP file.
From _AIX RS/6000 System and Administration Guide_ by James W. DeRoest
*/
#include <sys/types.h>
#include <utmp.h>
#include <fcntl.h>
main()
{
    int fd;
    struct utmp utmp;
    if ((fd = open ("/etc/utmp", O_RDWR)) < 0)
        exit (1);
    while (read (fd, &utmp, sizeof utmp) == sizeof utmp)
    {
        if (utmp.ut_type == USER_PROCESS && kill(utmp.ut_pid,0) != 0)
        {
            lseek (fd, - (long) sizeof utmp, 1);
            utmp.ut_type = DEAD_PROCESS;
            write (fd, &utmp, sizeof utmp);
        }
    }
    close (fd);
    printf("UTMP clean complete\n");
    exit(0);
}
```
E-mail Reports as PDF Attachments

We wanted to be able to send reports from our system as e-mail to our users, but the normal method for sending reports as e-mail messages is lacking to say the least. Normally, Colleague just adds the text of the report to the body of the mail message. This makes it impossible to print well, both because it tends to wrap, and because it includes the mail header.

There were three things we needed to do:

1) Convert the text version of the report to a PDF file so it could be viewed and printed from the user’s PC similar to what they would get if we sent it on hard copy. To do this we use the free ascii2pdf utility which can be found at: http://bulldog.tzo.org/ascii2pdf/ascii2pdf.html.

2) Attach the PDF file to the e-mail message so it could be saved, viewed, or printed separate from the mail message itself. This is done using uuencode. It turns out that by simply appending the output from uuencode to the text of a mail message, it gets sent as an attachment.

3) Create an e-mail message with a subject and text the way we wanted it. We did this by creating a simple text file of the mail message, along with the From, To, and Subject fields, and any text message for the body, and using that file, along with the uuencoded PDF file to the sendmail program using the –t option. This sends the message and reads the header information from the text file.
BUDGET.EMAIL

This is an example of how it all happens. The text message is created, then the report is translated to a PDF file using ascii2pdf, then it is uuencoded and appended to the message, then the whole message is used as input to the sendmail –t command.

Note that the From username does not have to be yourself, or even any real user. Sendmail does not care.

We use a UniBasic program to do all this, but it could also be done from a unix script as well.

```
Budget.email:

From: Santa Clause <santa@bw.edu>
To: dprez@bw.edu
Subject: Christmas Budget

Attached please find your budget report.
ascii2pdf -l -p 8.5 -s 80 Budget.txt
uuencode Budget.pdf Budget.pdf >> Budget.email
cat Budget.email | sendmail -t
rm Budget.txt ...
```

Resources

One of the nice things about unix and Colleague is the ability to use relatively small amounts of programming to greatly enhance your ability to get your work done. Even more important however is the fact that, if there is something you would like to do, someone may well have already done it. [ftp://ftp.cedarville.edu/pub/unidata](ftp://ftp.cedarville.edu/pub/unidata) has a great wealth of programs, subroutines etc. If you don’t find what you are looking for there, by all means, post a question on info-datatel. Even if you don’t find exactly what you are looking for, you may find a starting point. I doubt that many of these programs can be put to use at your site as-is. My real hope is that they start you thinking about the kinds of things that can be done so that you will go home and make more utilities that I can use!