HelpDesk Management System
Administrator's Guide

Related HLI Programs

This section of the Administrator's Guide gives a quick description of what HLI offers (syntaxes and instructions on programming are left to the UNIFY manuals) and then goes on to discuss the HLI programs that are included as part of the HelpDesk Managemtn System package.

What is HLI?

Depending on the version of UNIFY that you purchased, you will have either Direct Host Language Interface or Relational Host Language Interface. These are two separate packages with different names, but they have the same basic goal. That goal is to allow the local database development people to write C programs that read from, write to, and delete from the database as well as making changes to the design itself. DHLI and RHLI both provide sets of library routines as well as a UNIFY preprocessor, compiler, and linker. If you are so inclined, you may use the HLI tools to go as far as bypassing the ACCELL/Integrated Development System and writing your own application in C. However, at Delmarva, we chose to use the ACCELL/IDS to develop the application and use the HLI tools to build programs outside of the application to do very specific things that are either impossible to do with ACCELL, or are just easier and quicker to do at a command line versus entering the application.

HLI Programs Included With HDMS

The sub-sections below describe the function of each of the HLI programs included in the HDMS package.

`cronmail` `cronmail` is an automatic mail delivery system designed for the purpose of reminding people, via electronic mail, of events such as LAN server backups, monthly team meetings, etc. The origin of `cronmail`'s name is the UNIX *clock daemon* called cron that serves as a job scheduler. An example of a use for cron is our electronic mail link. We have cron set up to execute a program every hour during the day, and once during the night. This program dials out on a modem to Virginia where we take electronic mail meant for our site, and give mail for sites outside of Delmarva Power. cron is very flexible in that it can be set up to execute certain programs at ranges of dates of times. `cronmail` is based on the same idea except that all it does is send mail. `cronmail` comes in two parts. The first part is the database part where records are stored for requests to send mail to certain users on certain dates. The second part is an HLI program that runs every night, looking at each record in the `cronmail` Table and either sends mail or does nothing. When a record is entered in the `cronmail` database, you must enter the electronic mail address of the person that you want mail sent to. This e-mail address must be either a login name on the machine that HDMS runs

on, or an alias defined with a mail system such as mmdf or sendmail. In other words, it must be a name that the local machine will recognize.

`prob` `prob` is the most important program associated with the Problem Management and Trouble Ticketing portion of HDMS. This HLI program lies at the heart of HDMS's ability to promote a problem's priority. It is also the determining factor for when and where a Trouble Ticket is printed. At Delmarva Power, because of the fact that the SystemPro was almost completely dedicated to HDMS, we decided to simply start `prob` when the machine was booted. The way we did this was by adding the following lines to `/etc/rc`:

```
DBPATH=/usr/db/ACL4/db
UNIFY=/usr/db/ACL4/lib
export DBPATH UNIFY
su root -c "/usr/db/ACL4/src/prob/prob &"
echo "prob loaded."
```

The first three lines set up the environment of the process by setting certain environment variables. Note that the `setenv` shell is *not* used because this file is executed by `root`, who runs `sh`; `setenv` is a `csh` command. This is needed so that when `prob` is actually started (in the next line), it can find both the data and the application. Note that when the program is started, it is run in the background (due to to `&` sign following the command); this is necessary, because if it were not, `prob` would never finish (it is a *daemon*), and none of the remaining lines in `/etc/rc` would be executed, and the machine would never come up.

As stated before, `prob` is an important program that does a lot of things; as a result, it is difficult to describe briefly. Without going into a lot of detail, I will quickly outline how it works. When `prob` is first executed, it opens up the Problem Table in the database and searches for all open problems. It then looks at the date/time stamp on each record, and compares it to the current date and time. Depending on the difference, `prob` will either do nothing, promote the priority of the problem, and/or print the problem on the appropriate printer. It does this comparison for each open problem, and then `prob` goes to sleep for 60 seconds. When it wakes up, it starts over from the beginning by reopening the problem table and doing the search. In order to see how `prob` works, it would be best to either refer to the code itself, or to the flowchart contained in this package.

One note of caution about `prob` is that when the database is being reconfigured, `prob` will fail and the process will die. This happens because when `prob` wakes up *during* reconfiguration and starts to do its work, there are a lot of changes being made to the database. At some point, some of the pointers that `prob` is using will become invalid and there will be an error that causes the program to end. At this point you must either reboot the machine, or use the `nohup` program to bring back `prob`. We spent a little time at Delmarva Power trying to use `SIGUSR1` to suspend the process until reconfiguring was done, and then send

`SIGUSR1` again to restart the process, but we never got it to work. Also note that the way we have `prob` written currently, electronic mail is sent to a certain group of people telling them that `prob` has died. This is just a safeguard so that if `prob` dies at an unanticipated time, people will be alerted that there is a problem with the [crucial] program.

`statmon` `statmon` stands for Statistics Monitor and is simply an aid for quickly looking at the open problems in the system. The way this HLI program works is by opening up the problem table of the database and searching for open problems. It then pulls in certain fields from each record into an array and sorts that array based on priority and problem number. I chose to use bubble sort (an $O(n^2)$ sorting algorithm) to sort the array. This could definitely be improved upon, but for our purposes where the number of problems is usually under 20, bubble sort is sufficient. I did optimize this sorting a little because of the fact that it was a dual-key sort. The actual details are a little too low-level to explain here, but basically, I created one sort key from the priority and problem number fields, and then sorted by that one key. To see exactly how the sorting is done, you should refer to the code itself. This is a very good program to look at when you want to learn how to write an HLI program; it uses several of the UNIFY library functions for both database access and screen functions (like (X,Y) cursor positioning, screen clearing, *etc*.). This program has a man page written for it that explains how to use it.

`vg` `vg` stands for Vendor Graph, and it is a method we use to track the average time for vendors to repair problems with their equipment. The program will display the average number of days it took for the vendor to repair problems during the current month, plus the past 11 months. This program has a man page written for it that explains how to use it.

Note To Programmers and System Administrators

Just as user's environments need to be set up correctly with environment variables in order to use HDMS, processes running HLI programs need to have the same variables set so that they can find the data. This can be accomplished in one of at least two ways. First of all, you could use a shell script similar to `/usr/local/bin/hdms` (the script that actually starts HDMS) that contains lines to set the necessary environment variables. `statmon` is an example of an HLI program that is started this way. The other method we have used at Delmarva Power is putting the `putenv()` system call in the C program to manually set the necessary environment variables (`DBPATH` and `UNIFY`). `vg` is an example of an HLI program that uses this method. Both ways work perfectly fine, but it is probably cleaner to use `putenv()` because of the fact that it doesn't require an extra process for the shell script.