CPL is a First Aid Kit

by Dr AA Grainger

University of Tasmania Computing Centre

Abstract

Do you realise how useful and powerful CPL programs can be? CPL is a sort of **First Aid Kit** for System Administrators. CPL lets you place a 'real program' into the PRIMOS environment, combine two or more system utilities into a new tool, or patch over nasty or treacherous holes in a program or a PRIMOS interface. A survey of more than 100 CPL programs written over several years shows both the good and bad sides of CPL in the PRIMOS environment and highlights the high rate of change in PRIMOS itself.

What are some of the 'tricks' for creating powerful CPL programs? One of the most useful approaches is to have a more general CPL create another which is specifically tailored to do the task at hand. This paper explains a number of techniques which allow a CPL to examine the PRIMOS environment and generate tailored CPL commands based on what it finds.

From our experience with CPLs, we suggest a number of 'wish list' improvements to the PRIMOS environment.

Biography

Dr Tony Grainger is the Systems Programmer at the University of Tasmania Computing Centre. He has been responsible for Prime 9955-II, 750 and 2755 machines since 1985

Prior to 1985, Dr Grainger was involved for several years with the development of the Centre's data communications networks. He has held a number of other computing-related and research posts within the University, as well as acting as a consultant to industry.

Dr Grainger received a PhD in Electrical Engineering from the University of Tasmania in 1978 and a B.Eng. in 1968.

Introduction

This paper shows you the power of CPL — the broad range of ways CPL can simplify the PRIMOS operational environment. CPL programming is not always straightforward, and some of the most powerful commands aren't even documented. Several of the more useful CPL programming 'tricks' and general approaches are explained below. But, why have to use special tricks at all? If PRIMOS were to incorporate a number of reasonable enhancements, the complexity of powerful CPLs would be greatly reduced.

The appendices include further CPL reference information and an index to a set of contributed CPL example programs which are available on tape.

Appreciating the Power of CPL

Making Informed Decisions

Why write all these CPL's?
CPLs are suited for jobs which are tedious or difficult, subject to interruption or requiring special training. The idea is to automate as much of the job as possible — to let the program make the sorts of decisions that you are making.

The 'LogOut User' (LOU) cpl is an example. *(See Figure 1.1 and 1.2)* The operator is presented with information on the system console and has to make a decision: "what needs to be done in order to get all those processes off the system?" You don't want to have to think about that, you just want to press a button and make it happen. That is what most CPLs do. They simplify a complex situation into a simpler button-pushing exercise.

The crux of writing programs which take situations in context is to provide them with enough input to define the real situation. CPL is an excellent ***muscle*** but its ***sensory*** input is limited by what it can *(a)* glean from the FUNCTIONS available to it and any information it can *(b)* ***parse*** from the system utilities intended for the human eye.

Figure 1.1

What LIST_USERS provides:

```
[LIST_USERS Rev 19.4_2 SSP]

 3 users:

  3  U112004
 10  DAVISM
 19  MCGLASHAN

 7 phantoms:

 78  WSI300_MANAGER
 79  NM_SERVER
 80  WSIFTP_SERVER56
 81  WSIFTP_USER0
 82  BATCH_SERVICE
 83  YBTSMAN
 85  ANNEXE.B

 9 servers:

  1  SYSTEM
 67  NTS_SERVER
 68  TIMER_PROCESS
 69  LOGIN_SERVER
 72  DSMSR
 73  DSMASR
 74  SYSTEM_MANAGER
 76  NETMAN
 77  RT_SERVER
```

Figure 1.2

What we actually want to do:

```
/* PROCESSES_STOP.cpl  log out
&severity &error &ignore
LO   -3 /*  U112004
LO  -10 /*  DAVISM
LO  -19 /*  MCGLASHAN
LO  -83 /*  YBTSMAN
BATCH -STOP
PROP ANNEXE.B -STOP
&data STOP_NTS
yes
&end
&data STOP_WSI
yes
&end
&data STOP_NET
yes
&end
STOP_DSM
&return
```

if a user runs this command he only manages to log himself off since he doesn't have enough privileges to affect anyone else.

Techniques for Using CPL

How do you create CPL's that can act correctly in a complex situation? A number of general strategies are described below, along with specific details on how to implement each technique. Once you understand the basic concepts, it will be easy to copy or adapt these strategies and programming 'tricks' to suit your own CPL programs.

The First Option

The existing CPL FUNCTIONS provide information about most aspects of the file system. However, detailed information about the users on the system, the partitions of the file system and the actual configuration of the CPU (including the system name), are not available. The wish list below suggests the sort of details you need to be able to obtain from FUNCTIONS.

Alternatives

The only alternative is to use a como file to trap data from a utility and then to parse that file to create appropriate CPL statements. You can parse either with CPL's file interface or more rapidly using ED.

Using CPL to Parse Data

CPL itself has poor parsing ability. Its file input interface is slow and parsing is all done by FUNCTIONS. This is slow and inefficient. The only reasonable parser is the &ARGS statement. Even &ARGS is limited, since it can only parse command lines, not variables. To utilise this parser, it is necessary to call another CPL program which exports the results of the parse to global variables and then returns. (My second wish is for a parser like &ARGS for parsing a variable.)

Despite the slowness and poor parsing ability of the file interface, it is possible to write small programs which have small amounts of input to actually parse file input directly. For instance EDPAC.CPL, which parses only the access control list of an MFD, executes in a reasonable time while DISK_GVAR.CPL takes five minutes to parse a twenty line file .

Using PRIMOS ED for parsing data

The following are typical steps which a CPL must take to examine the PRIMOS environment and create a second customised CPL to carry out a specific task.

1. Use a como file to trap data from any utility.
2. Use an *&data* script for ED to transform the data to a CPL program or under rare circumstances write a special purpose utility to scan comos specifically to massage the data into an editable form.
3. Execute the CPL file.

Advantages
- Is nearly always feasible.
- Is not grossly inefficient, is much more efficient than reading and parsing the como in CPL.
- Is usually straight forward to achieve.

Disadvantages
- The editor script is virtually unreadable.
- Since the editor commands are not designed as a programming language there are deficiencies in making decisions based on the data in the file being processed.
- You must be careful not to use intermediate file names likely to be used by others.

Summary

Most CPLs in this paper use one of the last two techniques to obtain the data needed to sensitively apply the muscles provided by CPL. It is not the job of a CPL user to write functions to extract information provided by PRIMOS subroutines. This is a job for PRIME.

Examples

Before discussing some specific utilities used to make life easier, we will look at the techniques as they apply to LOU.cpl. *(See Figure 1.3).*

LOU has the overall structure discussed above:
- It has a capture segment, a como around a call to the LIST_USERS utility.
- It has an NSED script which allows it to make logical decisions about the data that was captured. At the end it executes the CPL it has carefully made to take into account all the problems of the situation.

In order to appreciate the complexity of the decisions LOU makes, look at the tables it sees on the terminal screen *(Figure 1.1)* and the orders it incorporates into the program to stop all the processes on the system *(Figure 1.2).*

The ordinary users have their entries in the table converted to logouts in the program. Some of the phantoms are treated in the same way.

The special purpose servers, which require much more complicated orders to stop them, have become ***&data*** scripts with the answer 'yes' included.

The printer phantom entries in the table have been carefully changed to PROP –STOP orders. The utilities which require name translations are also dealt with appropriately, STOP_DSM being the most obvious example.

Editor Script Programming

"How can an editor script make such complicated decisions? It's not a programming language!" Most people make that sort of response when they hear about all the things that can be done with an editor script. Look at the following section of editor script to see how it makes these decisions. It all depends on being able to detect the absence or presence of a cue. A typical line says:

LOU.cpl

Figure 1.3

```
/* LOU.cpl      Tidily stops all processes of specific kind.
/*               'Log Out Users'
&args kind:uncl
        como processes.como -ntty
        lu -sglcol -nw %kind%
        como -e
        &data NSED processes.como
             /* Remove lines from como we won't need.
             f No ; d
             t; n; d
             1 :; d; *
             /* Log out every_body.
             t
             n; g i|LO |nc i|-|c i| /+|f; c|+|*|; *
             /* Remove blank entries.
             t
             1 - /; d; *
             /* Do special logouts for these users.
             t; 1 BATCH; d; b; i BATCH -STOP
             &do printer &list [wild spool*>@@.ENV]
               &s printer := [before %printer% .ENV]
               t; 1 %printer%; d; b; i PROP %printer% -STOP
             &end
             /* Servers for the ETHERNET LAN users
             t; 1 NTS_SERVER; d; b; i &data STOP_NTS
             t; 1 STOP_NTS; i &end
             t; 1 STOP_NTS; i yes
             /* ETHERNET connect handler.
             t; 1 WSI300_MANAGER; d; b; i &data STOP_WSI
             t; 1 STOP_WSI; i &end
             t; 1 STOP_WSI; i yes
             t
             1 WSIFTP; d; *
             /* PRIMENET file transfer subsystem.
             t; 1 FTP; d; i FTOP -STOP_SRVR
             t; 1 YTSMAN; d; b; i FTOP -STOP_MNGR
             /* PRIMENET itself.
             t; 1 NETMAN; d; b; i &data STOP_NET
             t; 1 STOP_NET; i &end
             t; 1 STOP_NET; i yes
             /* DISTRIBUTED SYSTEM RESOURCE MANAGER.
             t; 1 SYSTEM_MANAGER; d; b; i STOP_DSM
             t; 1 DSMSR; d
             t; 1 DSMASR; d
             /* Don't log out these users.
             t; f LO   -1; d
             t; 1 TIMER_PROCESS; d
             t
             1 LHS_; d; *
             t
             1 _SERVER; d; *
             i &return
             t; i &severity &error &ignore
             file processes_stop.cpl
        &end
        delete processes.como
/*        como -tty
/*        klist processes_stop.cpl
/*&return
        sac processes_stop.cpl -like lou.cpl
        como -tty
        r processes_stop.cpl
 &return
```

technique for how to test this CPL

```
t; l BATCH; d; b; i BATCH -STOP
```

If the locate statement fails, the remainder of the commands on the line are skipped. So in this case, BATCH –STOP would only be included as an order if BATCH was 'located' successfully.

Look at how it deals with printers.

```
&do printer &list [wild spool*>@@.ENV]
    &s printer := [before %printer% .ENV]
    t; l %printer%; d; b; i PROP %printer% -STOP
&end
```

The ***&do*** statement in the CPL loop picks up the names of all printer's environment files from the SPOOL* directory. It then checks each one of these names to see whether it is referred to in the table of users. If a name is found, then it replaces that line with the PROP *name* –STOP order.

The Ethernet Connect handler is a good example of the third category.

```
/* ETHERNET connect handler.
t; l WSI300_MANAGER; d; b; i &data STOP_WSI
t; l STOP_WSI; i &end
t; l STOP_WSI; i yes
```

We need to make a data statement. The first part, making the ***&data*** STOP_WSI line, is easy. In order to make the other two parts you must use the same philosophy but instead of changing an existing line you have to create a new one. (Insert commands must be last on a script line so you need three lines of script).

These are specific examples of the power of the editor. To gain a full appreciation of how much the editor can do in this sort of situation, refer to the ***Editor Techniques*** help sheet in Appendix A.

Getting the Full Power of CPL

Make full use of parameter syntax
In order to fully appreciate how powerful CPL is you must understand one particular aspect of the language which is not normally explained. This is the kind of parameters you may use in particular CPL statements. One example is the ***&call*** statement. The name of the routine part of the ***&call*** statement can be any combination of literal and variable name. This is not explained in the CPL manual.

This example *(See Figure 2)* is called MATRIX.CPL because it takes two independent parameters, merges their names together to make a subroutine name, and thereby manages to call six or so subroutines based on the three possible values of one parameter and two possible values of the second.

If you look at the CPL summary sheet in Appendix B, you will see that each kind of CPL statement parameter is represented by different fonts. This enables you to quickly find out what you can put in each CPL statement.

You can see that not only is the ***&call*** statement capable of using FUNCTIONS, variables and literal strings in any mixture, but the ***&goto*** statement has the same power. Thus, computed *gotos* in CPL are particularly powerful.

Matrixed Addressing of Subroutines

Figure 2

If you have two (or more) independent variables and you need to deal with each possible pair (or triplet etc) by a unique subroutine then CPL will allow you to manufacture the name of the subroutine and then call it. The following example shows the principle:

```
/* MATRIX.cpl  demonstrates matrixed addressing of routines.
/*          operation has legal values of add, change, delete.
/*          object has legal values of tree, acl
&args operation; object; parameter1; parameter2

            &call %operation%_%object%
      &return

      &routine add_tree
            create %parameter1%
      &return

      &routine change_tree
            cname %parameter1% %parameter2%
      &return

      &routine delete_tree
            delete %parameter1%
      &return

      &routine add_acl
            sac %parameter1% %parameter2%
      &return

      &routine change_acl
            edac %parameter1% %parameter2%
      &return

      &routine delete_acl
            sac %parameter1%
      &end
```

Figure 3

/* RECURSE.cpl

```
/* Demonstrate do loop in recursive calls
&args path
      &if [null %path%] &then ~
            &s path := [dir [pathname x]]
      &else &s path := [pathname %path%]
      &s pad :=
      type %path% contains:
      &call directory
&return

&routine directory
  &s pad := %pad%....
  &do entry &list [wild %path%>@@ ]
    type %pad%%entry%
    &if [attrib %path%>%entry% -type] = UFD~
    &then &do
                &s path := %path%>%entry%
                &call directory
                &s path := [dir %path%]
    &end
  &end
  &s pad := [after %pad% ....]
&return
```

Figure 4

Recursive Use of *&do*

```
<USERB1>UCC>AAG>DOCUMENTS contains:
....PRIMOS_AREAS
........COMMAND_LINE_ENVIRONMENT
........FUNCTIONS
....NEWSLETTER
........RELATIVE_EFFICIENCIES
........C_USERS_UNDER_PRIMOS
....HELP
........HELP.DIR
........HELP_INDEX.HELP
........PDEV.HELP
....PACKAGE_DATA
........STANDARDS
........PACKAGE_LIST
....BACKUP
........DECISION_TREE
........OVERALL_ASPECTS
....NON_PRIME_STUFF
........EXTENSIONS
........IBM_SCREEN
```

Make full use of recursion
A feature of CPL hidden within the iteration part of the ***&do*** statement enables recursive procedure calls to contain ***&do*** statements. When an ***&do*** statement is entered, any ***&to***, ***&by*** or ***&list*** parameters are evaluated and stored. If the ***&do*** is within a procedure called recursively, then each recursion has its own separate store. This feature of CPL is very useful. The RECURSE.CPL example *(See figures 3 and 4)* demonstrates one use. (Please note that [WILD . . .] has a limit of 1024 characters.)

Utilities to make life easier:

Each new release of PRIMOS cures some problems and introduces new ones. The turnover of problems is a good indicator of the progress of PRIMOS over the years. CPL programs which patched the problems of the past and present provide these indicators. A number of CPLs are included below to show the span of problems faced by the Systems Administrator. PRIMOS doesn't provide any other way to accomplish these objectives.

We will review the following areas:

- spoolers
- the batch subsystem
- process control commands
- commands associated with partitions
- the shared segments list
- the system administrator's data base
- envelopes for utilities
- the aberrations of Rev.21

Of the eight areas listed above, PRIME have made significant improvements in two, and useful improvements in a third.

Spoolers

Prior to Rev.21 we maintained PRINT.CPL which decided from a user's project where to spool his output. It ensured that 500 students had output distributed among the 15 printers other than PR0 and thus made the operators' lives bearable. The Rev.21 spooler has rendered PRINT.CPL almost redundant. A small change (see the wish list for details) would make it completely so. *This is an example of PRIME getting it right!*

Batch

Prior to recent revs of PRIMOS we maintained two CPL programs, WAIT_UNTIL_-DARK.CPL and BATCH_IDLE.CPL. The latter ensured that BATCH jobs ran at IDLE priority. Recent changes to the BATCH subsystem have rendered this CPL totally redundant. *PRIME have it right again!*

The WAIT_UNTIL_DARK.CPL clogged the NIGHT BATCH queue so that no jobs could run from that queue during working hours from Monday until Friday, but did not clog the queue at weekends. Recent changes have given users the ability to apply a batch window which is much neater than clogging the queue. But, we still need to change that window at weekends. So, now we have BATCH_WINDOW.CPL. *Prime nearly did it!*

Process Control Commands

The discussion of LOU.CPL earlier showed how difficult it is to make appropriate actions when groups of users need to be logged off, sent messages, or be chapped. LOU.CPL and CHAPU.CPL allow any group of users, selected by LIST_USERS options, to be dealt with appropriately. We have a special version of LOU.CPL which shuts down all network servers and phantoms. There is no obvious way for PRIME to make these CPLs redundant, but I hope they do.

Commands Associated With Partitions

EDPAC.CPL is to priority access rights what EDAC is to normal access rights. Where two or more processes are controlling access to the same partition, EDPAC is essential; so essential that we converted our CPL to an EPF long ago. PRIME could provide EDPAC as a standard utility — why don't they?

Physical DEVice numbers (octal numbers) are used by PRIME to ADD, SHUTDN and FIX disk partitions. Our MOUNT, UMOUNT and CHECK_DISK CPL programs allow us to do the same things by partition name. We do this by creating global variables for every partition and removable pack drive from a disk definition file. Could PRIME achieve the same functionality? I wish they would.

The Shared Segments List

One task often neglected until the last minute is to update the shared segments list whenever a change is made to any utility which uses shared segments. MAKE_SHARE_LIST bypasses the need to do this by creating a shared segments list from the PRIMOS.COMI (STARTUP.CPL) and the individual SYSTEM>@@.SHARE files. It uses ED scripts to find the actual share commands and creates an annotated file of the segment numbers, the access mode and the package name. It cannot get the shared segments for INFORMATION because the share statement uses a CPL variable rather than a literal argument. I wish Prime maintained this list with the actual share command. (See the wish list below.)

The System Administrator's Database

EDIT_PROFILE is one system utility where it is almost impossible to effectively utilise the editor to transform como files. Output related to one user, one user project, or one project is spread over many lines of the como. For this particular case a general purpose extractor was written; it generates reports with all details of a particular subject on one line. Such a file is easily edited to make cpl programs which call other CPLs for SAD and file system maintenance. (See wish list.)

We have many CPLs which utilise these SAD dumps to tidy the file system, create user and project lists, and to tidy the SAD itself — for example, by removing users with no projects or no file trees.

A problem in the SAD: The sheer size of the data base brings problems not anticipated by the designers. For example at the end of each teaching year we must de-register some 500 student entries. This may take up to 6 hours. The slow part is the delete of the global user information. Even if the user is not in any project the entire project data base is searched to ensure this is true. This phase takes a long time. (See the 'wish list' below.)

Envelopes for Prime Utilities

Many PRIME utilities and packages require CPL envelopes to tame them for the use of ordinary mortals, particularly so for novice users. TED.CPL is a typical example.

ED is the basic editor on a Prime. To be useful as a production editor it needs many of it's default modes reset. It also needs to be protected against forced logouts. In addition tab stops need to be set appropriately for the source language of the file edited. TED.CPL is an envelope for ED which provides all these functions. I don't expect Prime to ever make this CPL redundant.

The Aberrations of Rev21

Each new PRIMOS rev has its teething problems. A typical conversion problem is described here. CPL provided a quick solution, even though every user in the SAD had to be checked.

At REV21 the OWNER key was added to the access rights keys. Information users need this right to some files which the automatic conversion done by remaking the file system did not give them. To correct this deficiency all information users needed to be found and their access rights upgraded. GIVE_OWNER_GROUP.cpl used the SAD dump files to find users with the .INFO group and then used their IAP to locate and fix the access problem.

Each new rev of PRIMOS has its own 'features', some of which have uncomfortable consequences. Rev21 has per process search lists. When these become 'circularised' PRIMOS re-initialises your command environment (ICEs you). In our login sequence all users have abbreviation files and global variables initialised: ICE disables them again. We use THAW to re-establish abbreviations and global variables. In the course of time PRIME will make this CPL redundant (I hope).

Wish List of PRIMOS Enhancements

Many of the CPLs and techniques mentioned above could be made simpler or would perhaps be unnecessary if Prime provided the necessary 'hooks' into the system. Other operating systems which we use, such as Unix and IBM S/38, are structured in such a way as to easily provide virtually all of information the systems programmer requires to accomplish the task at hand.

Experience demonstrates that PRIMOS continues to change to meet user's needs. The following enhancements are suggested as logical extensions of current PRIMOS commands and functions. Their availability would increase the control user's could exert over their environment. No less importantly, they would vastly simplify the implementation of the kind of powerful CPLs this paper has described. One would not have to be a 'systems guru', using arcane techniques to achieve an objective.

FUNCTION *Extra file system functions:*
The [ATTRIB ...] options need augmenting. Consider the extras:
- LAC List of access rights;
- LPAC List of priority access rights;
- LQ The actual quota;
- LSIZE The actual size in records;

FUNCTION *Extra user data functions:*
[WILD_U user_name ...] where user name is wild yields a list of process numbers for which the options below are valid:
- TERM Only terminal users
- PH Only phantom users
- PRIMIX Primix process
- PHX Primix phantom
- REMOTE Network user
- SLAVE Process accessing file system from remote system. with support from:

[ATTRIB_U process_number ...] yields specific details:
- U_NAME The name of the user.
- TYPE The process type.
- PROJ The project associated with process.
- GROUPS The list of groups available to the process.
- COMO The pathname of any como file open by the user.
- IAP The initial attach point
- HOM The current attach point
- CPU The CPU seconds used by process since startup.
- IO The IO seconds used by process since startup.
- CON The number of minutes since process startup.
- SSEGS The list of static segments in use by process.
- DSEGS The list of dynamic segments in use by process.
- ASSIGN The list of devices assigned to the process.

FUNCTION *Extra partition information:*
[WILD_D partition_name ...] where partition name is wild and yields a list of partition numbers satisfying options:
- SYS The name of the system partition.
- REMOTE The list of remotely mounted partitions.
- LOCAL The list of locally mounted partitions. with support from:

[ATTRIB_D partition_number ...] yields specific details:
- PDEV The physical device number.
- FREE The free space on the partition.
- USED The space used by the file system on the partition.
- NAME The name of the partition.

FUNCTION *Extra network information:*
[WILD_N net_name ...] where net_name is wild and yields a list of node names satisfying the options:
- PDN Public data network.
- RING Local ring network.
- FDN Full duplex network with support from:

[ATTRIB_N node_name ...] yielding specific details:
- TYPE The types of network as above.
- STATE The current state of the node interface.

FUNCTION E*xtra PRIMOS information:*
[ATTRIB_S ...] System information:
- NAME The name of the system.
- PMEM The memory size of the hardware in records.
- MEM The memory actually in use in records.
- WIRED The wired memory at configuration time.
- NSEGS The maximum number of segments.
- SEGS The segments in use.
- CPU The cpu seconds usefully used since boot.
- IO The io seconds since boot.
- USERS The maximum number of process slots.
- START The seconds since boot.

In fact there is a good argument for a CPL function to access any data which is available from a system call. This should be a target for any good operating system.

CPL *Parsing a variable as a text line:*
CPL provides a means of parsing the command line arguments which is both effective and powerful. It does not however provide the same power to parse and split a line of text read from a file. This prevents effective use of CPL to interactively deal with lines from a file. It reduces it's potential power to split a string variable into words, check their syntax, and distribute them word by word to other variables. CPL would have greatly enhanced power if a new statement was introduced to the language with the form:

&PARSE %fred% &TO <normal &ARGS syntax>

BATCH *The batch window control could include day of the week:*
The CAP open and CLOSE times at present cater for time of day. It would be useful if two pairs of open and close times be given: one effective on week days the other if present over-riding at week ends.

PDEVS *Physical device numbers should be invisible:*

The form and rules for manipulating PDEVS are archaic. They make CPLs and COMIs unreadable especially when more than one removable drive is available and disks may be moved between machines. If functions were available as suggested under extra partition information this would help but what is really needed is a small data base holding particulars of every disk. This should be updated from a disk as it is mounted on a drive or for fixed disks as they are mounted.

EDPAC *Priority access needs to be edited:*

In a multiprocess situation SPAC and RPAC are too crude for administering priority access rights to partitions. EDPAC is needed in the same way as EDAC is needed for ordinary access rights. (My epf seems to work but perhaps there is a null access period during the change over to new rights and this stops Prime from providing this essential tool.)

LOG *shared segments list:*

When the share command is issued from a cpl it would be easy for the share details and the name of the cpl to be logged as a one line entry in a shared segments log file. This could be done by making the share command a normal program which does the logging and calls the system as it does now. I have simulated this by enabling my abbreviations in the share cpls just before the share statement and thus replace the share command by a call to a logging cpl. While this approach is effective it is very non standard. The first option is far better.

SAD *Dumps of SAD contents:*

I wish the sad could be asked to create a list of all global user data, or all user project data, or all project profile data with one line containing all the details about one user, one user project, or one project respectively. It would be much more efficient than scanning a como and extracting the details.

SAD *User count of project entries:*

The design of the SAD needs modifying by adding a field to the global data for a user. This is a count of the projects for which the user is registered. This would speed user deletions enormously in our case since we delete users after we delete all their project entries. It would also speed deletes of any user by a factor of two on average since there is no need to scan further projects when the count reaches zero.

SAD *Free format store for USER and PROJECT entries:*

The SAD would become a true user data base if it could hold a small amount of extra data for each global user entry and each project entry. This field should allow a free format line of text so that user details such as their full name may be stored. The full name of a project is also useful in the same way.

SAD *ACL and QUOTA data for each IAP:*

The SAD could be much more helpful by storing and setting some file system attributes associated with the IAP it presently stores for user project entries and for project entries. The ability to create the entry part of an IAP is needed first. An ACL which was used to EDAC or SAC the IAP when requested by the administrator would be very useful. A QUOTA which could be applied by edit_profile would complete the tasks presently done by our CPLs.

PRIMOS SPOOLER

Prior to the REV21 spooler we used a PRINT cpl which used the user's project name to decide which printer to send output to. This was essential since we support widely dispersed printers and terminals. Our projects are distributed among some sixty departments of the University with up to seven or so projects per department. For this reason the first three characters of a project are defined by the department name. Thus the Science faculty department of Information Science projects are:

SIS210	A second year undergraduate project,
SIS310	A third year undergraduate project,
SIS.SUPPORT	A project for support programmers,
SIS.RESEARCH	A project for staff doing general research.

When the REV21 spooler was released we could dispense with our CPL and add one file per project to the SPOOL*>ATTRIBUTES directory. This sounded like heaven. On our smaller machine we did it. On the larger machine which supports 164 projects we did not. The physical effort did not seem worth the result: so our users still incur the CPL overhead and we must still maintain the CPL to cater for any changes in the legal syntax of SPOOL requests.

Alternative

If wild characters and fields were allowed in the names of the SPOOL*>ATTRIBUTES file names and they had their usual meanings, then we could restrict Information Science printers with a single file called SIS@@. A more useful alternative would be to have three files called SIS2@@ SIS3@@ and SIS.@@ so that student access to special printers could be inhibited. If there was a NOT wild character, this could be reduced to SIS.@@ and SIS^.@@ which would be ideal.

Appendix A: Editor Techniques

Conditional Edit Commands

If TARGET then <edit if found> else <edit if not found>:

1. To transform the file if any TARGET is found in the file you do:

   ```
   t; l TARGET; <editor orders to change file>
   ```

   the editor orders are not executed if the TARGET is not found.

2. To transform the file if no TARGET is found in the file you do:

   ```
   t; l TARGET; t
   n-1; <editor orders to change file>
   ```

If the TARGET is found the pointer will be placed at the top of the file: the second pointer movement then fails and the editor orders are not executed. When the TARGET is not found the pointer is at the bottom of the file: the second pointer move then succeeds and the associated editor commands are executed.

3. Do editing commands both if target is present and if target is absent by merging these two.

   ```
   t; l TARGET; <edit if found>; t
   n-1; <edit if not found>
   ```

Using File Content to Edit File Content

To make a macro from the current line to change other lines in the file, take these steps:

1. Reduce the line to the PART(s) you need to make the edit order, e.g. use:

   ```
   g e5m8          to get the 6th to 13th characters
   g nd d nd c     to get the second word.
   ```

2. Make the PART(s) into the edit order you need to achieve your goal. Common goals are listed below:

   (a) To use the PART as the TARGET of a conditional edit command. e.g. use

   ```
   g i|t; l |fi|; <edit commands if TARGET found>|
   ```

   will locate all lines containing PART and edit them.

   ```
   g i|t; f(7) |fi|; <edit commands if found>|
   ```

   will find all lines with PART starting in column 7 and edit them.

   (b) To make the PART into an edit command with explicit search or address to set the pointer before the edit. e.g.

   ```
   g i|n-1; g a  |f  will append PART to previous line.
   g i|t; l TARG; c/TARG/|fi|/; *|
   ```

   will replace every TARG with PART.

   (c) To use one PART of the line for a TARGET and a second (or the same) part as a modification to a line which is found.

   ```
   g i|t; f |c i|; c//|nd c i| /; *|
   ```

   will use first word to find the lines to be edited and will insert the second word at the start of the line.

3. Put the command into a buffer and delete it from the file with:

   ```
   mov STR.1 inlin; d
   ```

4. Execute the macro with:

   ```
   x STR.1
   ```

CPL: Command Procedure Language

General Remarks

- Put only one statement per line of CPL program.
- Indicate a statement is continued on the next line by appending a ~.
- Text between a /* string and line end is treated as a comment.
- The only legal statements are PRIMOS commands or CPL statements.
- Reserved words are words beginning with an ampersand (&) character.
- Ignore case for all except literal strings.
- Ignore case for label, routine and variable names and for reserved words.
- A PRIMOS command's return status is assigned to variable SEVERITY$.
- Any text enclosed by single quotes becomes a LITERAL string.

Variables

Strings only. Maximum 1024 characters. Global scope.

%var_name% Read as "replace me with text in local variable var_name".

%.var_name% Read as "replace me with text in global variable .var_name".

Documentation Conventions

Abbreviations and kinds of parameters.

&OPTioNaL Write as &OPTNL, &OPTIONAL, &optnl or &optional. (Ignore case)

[can_omit] The item enclosed is optional.

{many_times} The item enclosed is optional and may be repeated many times.

The sub-strings allowed in a situation are shown by the text font and style:

literal strings only (no unquoted % [&] or space characters.);

variable strings and literal strings in any mixture;

`functions, variables, and literal strings in any mixture;`

expressions (CALC function syntax) or any string mixture.

Entry/Exit from CPL programs

&ARGS Assign parsed command line words to the specified variables.

&ARGS [var_spec {; var_spec}] {optional_var_list;}

where an optional_var_list is:

var_name:{**-option_key**} {var_spec}

and a var_spec is:

var_name:[parse_rule][=*default_value*]

and a parse_rule is one of: CHAR, CHARL, DEC, OCT, HEX, PTR, ENTRY, TREE, DATE, REST, UNCL

Optional_var_list variables are set if the option_key is found.

&RESULT Export expression when this CPL function finishes.

&RESULT *the_answer_to_be_returned_to_the_caller*

&STOP Exit the program and optionally return a severity and message.

&STOP [*severity_number*][&MESSAGE `message_for_user_terminal`]

Simple Statements

`primos_command` Invoke the primos command with its normal terminal input.

&EXPAND Do/Don't expand abbreviations as a first step of CPL parsing.

&EXPAND ON | OFF (default OFF, Scope: current procedure)

&Set_var Set variables in list to string formed by expression.

&S `var_name` {, `var_name` } := *assigned_value*

&DATA-&END Text formed by statements is filed, then program is started with terminal input from file, then file is deleted.

```
&DATA primos_command
{statement} /* Text formed here goes to a temporary file.
[&TTY]  /* Revert to terminal at end of file input.
&END
```

Debugging CPL programs

Scope: current subroutine

&DEBUG Enable or disable execution and/or echo all or primos commands and/or display variables when their value changes(default OFF)

&DEBUG [&OFF | &[No_]EXecute | &[[NO_]ECHO [ALL | COM | DIR]] | ~
[&[NO_]WATCH {*var_name*}]]

Execution Flow Control

&IF-&THEN-&ELSE When expression is true, execute statement following &THEN.
When expression is false, execute statement following &ELSE.

```
&IF expression &THEN statement
[&ELSE statement]
```

&SELECT-&END Find the first &WHEN expression matching the control expression and execute the next statement. If no WHEN expression matches, execute the statement following &OTHERWISE.

```
&SELECT expression
{&WHEN expression
statement}
[&OTHERWISE
statement]
&END
```

&DO-&END For each iteration execute the statements enclosed by &END.

```
&DO [iteration]
{statement}
&END
```

iteration may be omitted (statements are executed once only)

or [var_name := *initial_value*]~
[[&TO *final_value* [&BY *increment_value*]] |~
[&REPEAT *next_value*]]~
[[&WHILE *condition*] |~
[&UNTIL *condition*]]

or `var_name` &LIST `list_with_spaces_between_items`

or `var_name` &ITEMS *a_value* (usually [wild ....-SGL])

NB: *final_value* and *increment_value* are calculated only once.

&LABEL Name a place to which program flow may transfer with &GOTO.

&label **label_name**

&GOTO Transfer program flow to named label.

&goto `label_name`

Subroutines

&CALL Invoke the named subroutine.

&CALL `routine_name`

&ROUTINE Declare the following program text to be the subroutine named.

```
&ROUTINE routine_name
{statement}
&ROUTINE | &RETURN | "end of file"
```

&RETURN Exit the subroutine and optionally return a value or a message.

&RETURN [*severity_number*] [&MESSAGE `terminal_message`]

Special Conditions

Scope of &CHECK, &SEVERITY and &ON is current subroutine.

&CHECK Invoke named handler if expression is true after any PRIMOS command.

&CHECK *my_truth* &ROUTINE `handler_name`

&SEVERITY At specified level of severity$ (set by PRIMOS commands) stop the program, ignore the error or execute named routine.

&SEVERITY (&ERROR | &WARNING) (&FAIL | &IGNORE | &ROUTINE `name`)

&ON When the named condition arises execute the routine named.

&ON **condition** &ROUTINE `handler_name`

&REVERT Cancel the condition handler for condition. (Armed by &ON.)

&REVERT **condition**

&SIGNAL Set nominated condition. Will invoke a handler armed by &ON.

&SIGNAL **condition**

ANY$ and QUIT$ are the most useful of the 50 odd PRIMOS conditions.

Primos Command Line Functions

General Remarks

The PRIMOS command line parser (and the CPL language parser) interpret text enclosed within square braces, e.g. [date -full] as a function and parameters. The braces and the enclosed text are replaced by text produced by function execution, thus rebuilding the command before it too is executed. Each PRIMOS intrinsic function, its parameters, and returned text are summarised below.

Documentation Conventions

{item1 item2 item3} means select at most one of the enclosed items.

Environment

ABbrev	-EXPand text	"text" expanded from your abbreviations.		
	-STatus	Pathname of current abbreviations file.		
CMD_INFO	-name	Name of the condition on your stack,		
	-CONTinue_	SWitch Boolean value of the continue-to switch on the stack,		
	-RETurn_PerMiT	Boolean value from stacked return-permitted switch.		
DATE	-VFULL	21 Oct 81 13:24:48 Tuesday	-TAG	811021
	-FULL	81-10-21.13:24:48.Tue	-TIME	13:24:48
	-UFULL	10/21/81.13:24:48	-AMPM	1:24 PM
	-CAL	October 21, 1981	-DOW	Tuesday
	-FTAG	811021.132448	-DAY	21
	-VIS	21 Oct 81	-MONTH	October
	-USA	10/21/81	-YEAR	1981

GET_VAR expr — String in the variable named by expression or $UNDEFINED$.
GVPATH — Pathname of your active global variables file.

File System Information

ATTRIB path {-TYPE -DTA -DTB -DTC -DTM -LENGTH}
- -TYPE — Type of file. e.g. SAM DAM SEGSAM SEGDAM UFD ACAT etc.
- -DT{A B C M} — Date and time: Accessed, Backed_up, Created, Modified.
- -LENGTH — Length in words. (2 bytes per word.)

EXISTS path {-FILE -DIRectory, -SEGment_DIRectory, Access_CATegory}
TRUE if file of optional type exists else FALSE.

Expand_Search_Rules entry_pathname_in_key_directory {search_rule_name}
Full pathname of file system object located by rule.

PATHNAME rel_path — Pathname of "rel_path" even if entry part nonexistent.

WILD wildpath {wildentry(s)} {control} {-SINgle unit_var}
Entry name list, space separated, of all files in the directory of wildpath which satisfy the wildpath or wild entries and also satisfy the control options. Use HELP WILDCARDS to get wild and control details. -SINGLE checks variable "unit_var". If 0 it opens the wildpath directory and sets "unit_var" to file unit. One directory entry is returned for each call.

File Access

(status_var is set to PRIMOS ERROR CODE.)

OPEN_FILE pathname -MODE {r w rw} status_var
Decimal unit number of file unit opened.
Variable "status_var" is set: 0 if file opened
(To close file use a form of the PRIMOS command as in:
CLOSE -UNIT unit_number or
CLOSE pathname or
CLOSE [TO_OCTAL %unit_number%])

READ_FILE unit status_var {-BRIEF}
Quoted string containing one line of file.
Variable "status_var" is set, if read ok: 0, if EOF: 1.

WRITE_FILE unit text — 0 if unquoted text is written to file else PRIMOS ERROR CODE.

Interactive Input

QUERY text {default} — TRUE if response to "text" on screen is Yes or OK. FALSE if response is NO (or <RETURN> when no "default") "default" if given and RETURN is pressed.

RESPONSE text {default} — User's answer to "text" on screen unless answer is <RETURN> when "default" is returned.

Substrings of Strings

AFTER text target — The part of "text" after substring "target" else NULL.
BEFORE text target — The part of "text" before substring target else "text".
DIR path — The part of "path" before last ">" else "*".
ENTRYNAME path — The part of "path" after last ">" else "path".
SUBSTR text nth {num} — "text" from and including "nth" char (for "num" chars).
TRIM text {-LEFT -RIGHT -BOTH} {char}
Remove leading, trailing, or both "chars" (spaces).

Translating Arbitrary Strings

SUBST text old new — "text" with each occurence of "old" replaced by "new".
TRANSLATE text {new old} — "text" with each occurrence of a char in "old" swapped for the matching char in "new". If "old" is omitted the ASCII collating sequence is assumed. Upper case output results when both "old" and "new" are omitted.

Translating Number Strings

HEX hex_string — Decimal string for the number defined by hex_string.
OCTAL octal_string — Decimal string for the number defined by octal_string.
TO_HEX decimal_string — Hex string for the number defined by decimal_string.
TO_OCTAL decimal_string — Octal string for the number defined by decimal_string.

Boolean and Numeric String Data

NULL text — TRUE if no text or text is ' ' else FALSE
LENGTH text — Position in "text" of last character.
INDEX text target — First position in "text" of "target". (0 if not found)
SEARCH text targets — First position in "text" of any char in "targets". (0 if not found)
VERIFY text targets — First position in "text" of a char not in "targets". (0 if not found)

Manipulating Quotes

QUOTE list — 'list' with any enclosed ' translated to ' '.
UNQUOTE list — list with single 's deleted and ' ' replaced by '.
RESCAN text — Evaluation of the expression formed by [UNQUOTE text].

Numeric and Boolean Arithmetic

CALC expression — Decimal string or TRUE or FALSE resulting from the evaluation of "expression" containing the operators below.

()	Space each side of each operator is mandatory.
^ + -	Parentheses may be nested to five levels deep.
/ *	Unary "not" "plus" and "minus" used first.
+ -	Arithmetic operators "divide" and "multiply" are next.
= ^= < > <= >=	Binary "add" and "subtract" are used next.
&	Relational operators are next.
|	Logical "and" is next.
	Logical "or" is last to be used.

MOD d_A d_B — Decimal string for remainder(A / B) where A and B are the numbers defined by decimal strings d_A and d_B.

Appendix C: List of CPLs

Utility:

TITLE_EXTRACT.cpl	Extract the title line of every file in this directory.
MAKE_BOOK.cpl	Concatenate the files with a alphabetic index and long file.
ED.cpl	Prime's ED with cpl for wild file names and recovery at logout.
FTP.cpl	Call WSI_FTP when arg is either name or LAN address.

Partition:

MAKE_DISK.cpl	Make the named disk partition.
CHECK_DISK.cpl	Runs fix_disk to verify disk directories.
MOUNT.cpl	ADDs a disk given its partition name.
UMOUNT.cpl	SHUTDNs a disk given its partition name.
DISK_GVAR.cpl	Scans the DISK_DEFINITION file to create global vars.
CHECK.DISK.cpl	Runs fix_disk to verify disk directories.
EDPAC.cpl	Adds or deletes access rights from existing ones.

New Primos Rev:

MAKE_SHARE_LIST.cpl	A program to extract shared segment numbers from a cpl.
NEW_ENV.cpl	Call from old spool directory to convert all old E. files.
INSTALL.cpl	Install new PRIMOS REV from distribution partition(s).
CONVERT.cpl	Converts new comi share files to cpl files.

Process control:

LOU.cpl	Stops all processes of specific kind.
TIDY_NET.cpl	Stops and restarts the network.
CHAPU.cpl	Chap using arg1 all users selected by lu arg2 arg3 ..
CHECK_USER.cpl	Use LU options and count users selected.

File system:

SEARCH.cpl	Finds target file(s) anywhere in tree below current ufd.
TREE.cpl	Just walks over the tree starting from current at point.
TMP_TIDY.cpl	Delete file(s) if more than %back% work days old
DEFACC.cpl	Set default access to directories, files, and segdirs in tree.
AGE_SIZE_TREE.cpl	Reports file sizes in reverse dtm order for a tree.

University of Tasmania Prime configuration

'PrimeA' 750:
- 339 users
- 34 projects
- 4 departments

'PrimeB' 9955II:
- 728 users
- 164 projects
- 65 departments

'Admin' 2755:
- 186 users
- 13 projects

A B