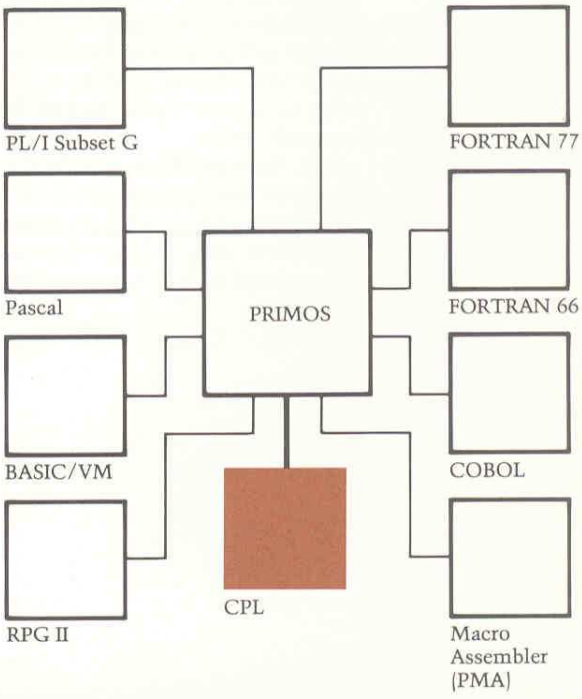
Product Bulletin

PRIME

Command Procedure Language (CPL)

Features

- Comprehensive command language for the Prime 50 Series systems.
- Allows intelligence to be built into command procedure files.
- Uniform invocation of Command Procedure Language (CPL) files and programs.
- Transparent interface to allow CPL files to be invoked as background (phantom) or batch jobs.
- Argument substitution into CPL files at run-time (positional and position independent, with type checking and default assignments).
- Local and global variables.
- Comprehensive set of flow-of-control directives.
- Over 35 standard command functions for expression evaluation, string manipulation, file and terminal I/O, etc.
- Interface to the PRIMOS® operating system standard error and condition mechanisms.
- Debug facility.
- Echo control.
- Command expansion facility.



Description

Prime's Command Procedure Language (CPL) is a powerful programming tool available as a standard feature with the PRIMOS® operating system. It is an extremely flexible high-level language with PRIMOS operating system commands as its basic statements. CPL allows sequences of operating system commands and CPL directives to be stored in a command procedure file that can be executed by specifying the file name. CPL directives provide for the passing of arguments into command procedure files, for control of the statement execution order within the files, and for error handling. CPL is made up of two parts, the language and the interpreter. The language allows users to write CPL procedures including PRIMOS commands, CPL directives, command functions and variable expressions. The interpreter evaluates variables and function calls and replaces them with their correct values. It then interprets and acts upon CPL directives. In addition, the interpreter passes any resulting command to the operating system for execution.

Interactive Program Development

Prime's family of interactive systems are well suited to the program development environment. A comprehensive set of high-level languages sharing a common call interface and an interactive Source-Level-Debugger is available. CPL provides an additional tool to further enhance a programmer's productivity. Often used sequences of commands such as compile, link and run sequences, and application control sequences can be stored in CPL files and executed with a single command. CPL's unique features, such as its comprehensive flow-of-control directives and the large set of command functions, allow CPL to be used as a programming language to write simple general-purpose tools useful to program developers. These tools can be used within other CPL procedures to provide more powerful tools.

CPL files can be submitted without change to be run in background mode or by Prime's batch subsystem. They are very amenable to interactive program development with background or batch production runs.

Application Control

Often it is desirable to store commands in a file for later execution. For example, one might wish to store a set of commands for control of an application to be run by a novice user. Instead of the user typing a long list of complex commands, CPL would allow a user to type a simple command to start the application running. Control of the application and its program would be the responsibility of CPL rather than the user. The logic in the CPL file would make intelligent decisions about which programs or commands to run based on the results of previous programs or simple arguments specified by the user when the file was run. Another use of CPL would be for the control of batch jobs where execution is essentially unattended and the user is not present to make job control decisions.

Compatibility

CPL runs on all Prime 50 Series systems, ensuring complete upward and downward compatibility among all central processors. Program migration is fully bidirectional. Users can develop CPL procedures on any other system supported by the PRIMOS operating system and run them on any other system supported by the PRIMOS operating system.

Capabilities

Invocation

CPL procedures are invoked exactly like other user-program run files. A file naming convention allows the PRIMOS operating system to distinguish between program run files and CPL procedures and take the appropriate action. CPL procedure files can be installed in the command directory and invoked simply by typing their name. In this way they can be made to look like any other command. CPL files can be submitted, without modification, to be run in background mode or by Prime's batch subsystem.

Variables

CPL provides for both local and global variables. Local variables exist only for the invocation of the CPL procedure in which they are defined. Global variables are associated with a user and exist until specifically deleted. (They even survive logout.) Global variables can be read and set by user programs as well as by CPL procedures, thus providing a communication mechanism between command procedures and programs. In CPL, variables are referenced by enclosing their name in '%' delimeters, eg: % this_is_a_variable %.

Argument Substitution

Arguments to be substituted into CPL procedures at run-time can be either positional or position independent. CPL provides a mechanism to assign default values to missing arguments. Type checking allows CPL to verify that a supplied argument is of the correct type. The 'args' directive is used to specify arguments (see Example 1). (Note: all CPL directives are preceded by a '&'.)

Example 1:
&args file: tree = mydir; number: dec = 1.
This statement defines two arguments: 'file' and 'number'. 'File' is a treename with a default value of 'mydir'; 'number' is a decimal number with a default value of 1.

'Rest' and 'unclaimed' data types are also available when the user wants CPL to interpret some arguments and take whatever else is on the command line as is.

Flow of Control

CPL flow of control statements allow a user to alter the normal sequential execution of a CPL procedure. A 'label' directive identifies a statement, and a 'goto' directive transfers control to a labelled statement. Conditional execution is provided by 'if...then...else' and 'select' statements. Many types of 'do' statements are available for statement grouping, iteration and list processing. 'Call' and 'return' directives allow transfer of control to an internal CPL routine, i.e. a routine in the same CPL procedure file. External CPL procedures can be invoked from within CPL procedures. Recursion is permitted.
(See Examples 2, 3 and 4)

Example 2:
&if (expression) &then &goto label
&else &do
....
....
&end

Example 3:
&do &while (expr.)
&do &until (expr.)
&do a: = 5 & to 10 &by 2
&do a: = 6 &repeat %a% *2 &while (expr.)

Example 4:
&do a &list %list _ of _ names%
This form sets 'a' in turn to each of the names from %list _ of _ names%.
This is useful for list processing.

Command Functions

Functions are procedures which return results as string values. These string values are substituted by CPL for the function call in the original statement. A method is available for users to define their own functions. Over 35 standard functions are available with CPL.
For example:

- The function 'calc' evaluates expressions containing the logical, arithmetic and relational operators. For convenience, CPL will allow a user to omit an explicit call on 'calc' if it is clear from the context that a 'calc' call is intended (see Example 5). Other arithmetic functions provide modulo and base conversion.

Example 5:
&if %variable% > 5 &then &return.

- The following string functions are available for the manipulation of character strings: length, substr, index, before, after, null, translate, trim, quote, unquote, subst (see Example 6).

Example 6:
If %filename% is equal to the file SOURCE. PL1, then the function [before %filename% .] will return the result SOURCE and [length %filename%] will return the result 10.

- File system functions are available to determine if a file exists; to manipulate treenames; to implement wildcard facilities; and to open, read and write files (see Example 7).

Example 7:
&if [exists %filename%]
&then [open _ file %filename% -mode R status].

If %filename% exists, then it will be opened for reading. The variable 'status' will indicate the success or otherwise of the open.

- Many miscellaneous functions are also available. Examples include: date generation and query and response (from terminal) functions.

Error Handling

A standard error severity mechanism is supported by the PRIMOS operating system. Commands return a severity code indicating 'error', 'warning' or 'no error'. This can be explicitly tested with an 'if' directive (see Example 8).

Example 8:
```
&if %severity$% = 1 &then &call error_routine.
```

Alternatively, a 'severity' statement allows the error-testing to be done automatically be CPL after every PRIMOS operating system command. Whenever CPL detects the specified severity level, it performs the specified action. Such action can be to ignore the error, to fail, or to invoke a CPL routine to handle the error (see Example 9).

Example 9:
```
&severity &error &fail.
```

A 'check' directive instructs CPL to automatically test the value of an expression after every PRIMOS operating system command and invoke a specified CPL routine whenever the expression evaluates true (see Example 10).

Example 10:
```
&check %A% > %B% &routine name
```

Condition Mechanism

The notion of a 'condition' comes from the PL/I language. It is an unusual or unexpected event such as terminal break key depressed, or arithmetic exception. The PRIMOS operating system Condition Mechanism provides a generalized method for user programs to define and trap conditions. CPL provides access to the Condition Mechanism in PRIMOS. CPL directives are available to define an on-unit (CPL routine to be invoked when a specified condition occurs) and to signal the occurence of a condition. Users are allowed to define their own conditions.

Miscellaneous Facilities

Subsystems and programs run from within a CPL procedure can take input either from the user's terminal or from data within the CPL file. A debug facility is available for checking CPL syntax (while suppressing execution of PRIMOS operating system commands), for checking the value of selected variables during execution, and for control of statement echoing. An expand facility controls statement expansion. This enables all PRIMOS operating system commands to be passed to a pre-processor for expansion before execution. This facility allows a user to tailor his command environment to his individual taste.

PRIME®

Prime Computer, Inc.
Prime Park
Natick, Massachusetts 01760

2/83

PB1438-002