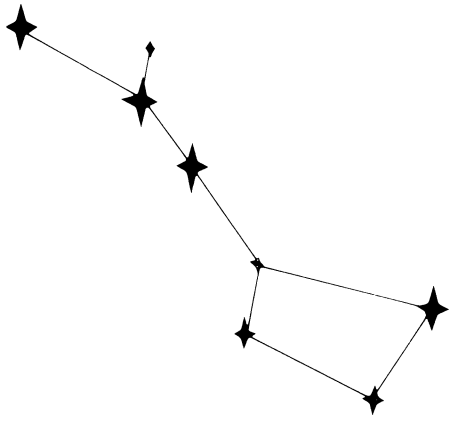


polar

software systems

POLAR SYSTEMS

POLAR TEST LANGUAGE

VERSION 1.3

1 CONTENTS

The following additional Options may be available at the end of this document :

PTL Driver Development Package
PTL Graphical User Interface
PTL Icon Editor Package
PTL Network Support

CHAPTER 2

HOW TO USE THIS MANUAL

2 HOW TO USE THIS MANUAL

This manual is divided into these major sections:

CHAPTER 2 How To Use This Manual
Describes the organization of the various manual chapters, syntax diagrams and notation conventions.

CHAPTER 3 Introduction
This section gives a global introduction to PTL with examples, phases in using PTL, the use of the different operating modes, the structure of PTL procedures and PTL filenames.

CHAPTER 4 System Generation
Describes how to generate a PTL system disk which contains all required instrument drivers.

CHAPTER 5 System Configuration
Describes how to use standard Fluke FDOS functions from within the PTL environment.

CHAPTER 6 System Functions
Explains in detail all configurations to be made to define a system.

CHAPTER 7 PTL Command Reference
Describes the individual PTL commands in detail.

CHAPTER 8 Editing a procedure
Describes the procedure writing process and the usage of the menu functions.

CHAPTER 9 Testing a procedure
Describes the procedure testing and compilation process.

CHAPTER 10 Operator Disk Generation
Explains how to create an easy to use operator disk.

CHAPTER 11 Advanced Programming
Gives a number of examples of PTL procedures, such that advanced users can make better use of the system.

CHAPTER 12 System Details
Gives several details on the PTL system, for more insight in the PTL system.

CHAPTER 13 Appendices
A collection of examples of output of the system.

2.1 HOW TO READ SYNTAX DIAGRAMS

Although not all of them are used in this manual, an overview of all symbols as used in the standard 1722A manuals has been provided for completeness.

A syntax diagram is a graphical representation of how to construct a valid command or statement in a programming language. It is a kind of "shorthand" way of writing down all the rules for using the elements of a language. Since they are used throughout this manual, learning how to read them can be a great time saver.

Words inside ovals must be entered exactly as they are shown.

Words inside boxes with round corners indicate a single key must be pressed, such as RETURN or ESC.

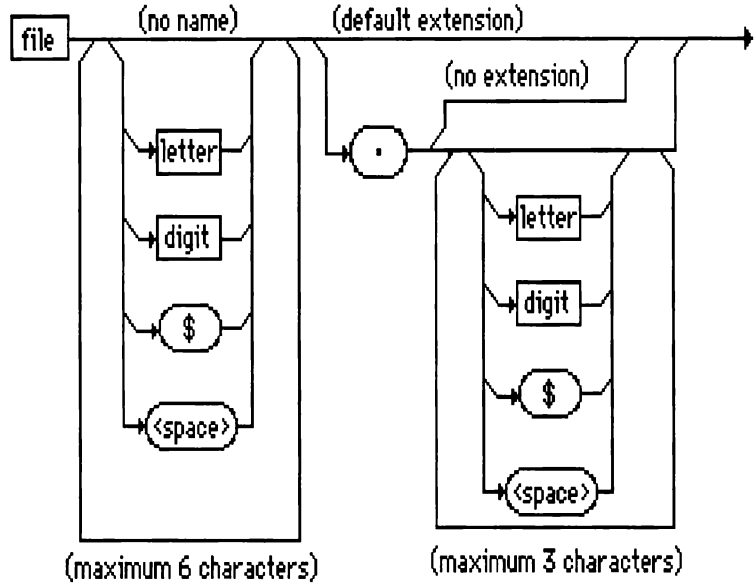
This indicates a space in the statement. (Press the space bar.)

To create a control character, hold down the control key (CTRL), then press the other key. This one is Control/C;
A box with lower-case words inside means that you supply some information. In this case, you would enter a filename.

Words in parentheses are explanations of some kind. They give added information about the nearest block or path.

From left to right, any path that goes in the direction of the arrows is a legitimate sequence for the parts of a statement. The following example shows the correct syntax for naming a file.

The translation is given below.



A line exits the top of this diagram with no keyboard input. This indicates that it is possible to NOT specify the filename or its extension. In this case, the file would have "no name" and the system would assign a "default extension".

Further down the diagram, you can see that there are other possibilities. They are explained by the remarks, "maximum of 6 characters" for the name and a "maximum of 3 characters" for the extension.

The filename can be any combination of letters, digits, the $ sign and spaces (up to six characters) and the extension can be up to three of those characters.

The filename and extension must be separated by a period, as shown in the oval block at the top centre.

The remark "no extension" means that it is not necessary if a name is specified.

Here are some examples of valid filenames according to the syntax illustrated in the diagram:

TESTIN.PTL DEMO.$3C $$$$$$.$$$

2.2 NOTATION CONVENTIONS

The conventions listed here are used for illustrating keyboard entries and to differentiate them from surrounding text. The braces {}, brackets [] and angle brackets < > are not part of the sequence. Do not type these symbols.

<xxx> Means "press the xxx key"
Example : <RETURN> indicates the RETURN key.

<xx>/y Means Hold down key xx and then press y
Example : <CTRL>/C means to hold down the key labeled CTRL and then press the key labeled C.

[xxx] Indicates an optional input.

Example : [input filename] means to type the name of the input filename if desired. If not, no entry is required and a default name will be used.

xxx Means to type the name of the input as shown.

Example : BASIC means to type the program name BASIC as shown.

{xxx} Indicates a required user-defined input.

Example : {device} means to type a device name of your choice, as in MF0: for floppy disk drive 0.

(xxx) This construction has two uses:

1. As a separate word, (xxx) means that xxx is printed by the program.
   Example: (date) means that the program prints today's date at this point.
2. Attached to a procedure or function name, (xxx) means that xxx is required input of your choice; the parentheses are required parts of the input.

   Example : TIME(parameter) means that a procedure specification is the literal name
   TIME followed by a parameter that must be enclosed in parentheses.

CHAPTER 3

INTRODUCTION

3 INTRODUCTION

The Polar Test Language has been developed for testing electrical and electronic equipment. It is the solution for those applications in which complex test procedures have to be created by technicians. PTL eliminates the need to control instruments in a system with low level commands.

Although originally designed for the Fluke 1722A System Controller, PTL has been meant not to be a computer oriented language, but a development tool for test engineers with a thorough knowledge of test problems, but without time to be invested in programming.

PTL is self documenting: it has been designed in such a way that a test procedure, developed with PTL, can be used without any change for documentation purposes as well.
Because PTL is so easy to use, time savings can be as much as 80% when a new procedure has to be created.

The language is powerful enough to develop and execute complete tests, but is not limited to its standard instruction set only. If desired, normal Fluke Extended Basic statements can be added anywhere in a Polar Test Language procedure.
If necessary, even Fortran or Assembler routines can be included in PTL procedures.

Operators, using procedures generated with PTL, are guided through all procedure steps without the use of the computer keyboard, decreasing the chance for wrong input. This approach makes it possible for even unskilled personnel to execute complicated test procedures.

3.1 EXAMPLE OF PTL

```
SUB Test(task)
On Error Subret
Procedure("Name='Amp test';Version='1.2'")
  Start_task("Name='Supply test'")
    Write("To=Printer;Type=Header")
    Setup("Instr=Power-supply;Voltage=12.6")
    Setup("Instr=Dmm;function=Vdc;Range=20")
    Apply("Instr=Power-supply;To=Amp input")
    Meas("Instr=Dmm;At=Testpoint 1;Min=5.24;" &
         +"Max=8.75,Units='Vdc'")
    Write("To=Printer;Type=Result")
  End_task("")
End_proc
SUBEND
```

3.2 SOFTWARE SUPPLIED

The Polar Test Language software is distributed on four or more disks in 1722A format which contains:

o System Generator (Optionally Menu driven)
  To generate a system disk
o Instrument drivers (One or more disks)
  Contains the drivers for instruments
o Master Operator Disk
  Is the original of new operator disks
o System disk

A pre-generated disk for training purposes.

3.3 PHASES IN USING PTL

The following phases can be distinguished in using PTL:

- System generation
The system generator is used occasionally, to combine the PTL language with the instrument information, contained in the instrument drivers. The result is a system disk, which will be used regularly to create procedures.

- System Configuration
Before a system disk can be used, details about the system configuration must be provided. These are: instrument addresses, logical names and the connections between instruments and the unit under test.

- Writing a PTL procedure
Once configured, the system disk is ready to perform its functions. Test procedures can be written, using a text editor or the advanced PTL menu editor.

- Testing a procedure
Procedures can be tested, using immediate commands, using the PTL interpreter and using the compiler.

- Creating an operator version
Once a test procedure is created, it can be stored on an operator disk. Tests can now be performed by inexperienced users, without the need of an operator keyboard. Procedures are protected against modifications.

For a detailed description, refer to the next chapters.

3.4 PTL OPERATING MODES

PTL has three modes of operation: Immediate, Interpreted, and Compiled.
The Immediate Mode is the default mode, i.e. the mode PTL starts in whenever it is entered from FDOS. Other modes are accessible only from Immediate Mode.

3.4.1 Immediate Mode

In Immediate Mode the PTL Interpreter accepts and executes PTL commands, typed in from the keyboard, or selected from the available command list using the 'MENU' function. No BASIC statements can be executed in Immediate Mode.

In addition, the Immediate Mode accepts the selection of the Main Menu functions (like 'SELECT', 'FUP' or 'PRINT') as a command and executes it. Refer to the description of the Main Menu for information on how to select Main Menu functions.

3.4.2 Interpreted Mode

The Interpreted Mode accepts input from a file, rather than from the keyboard and can be used to execute a procedure by means of the 'RUN' command from the PTL Main Menu. The file, with the filename, extension and device selected by the 'SELECT' function, will be executed.
As in Immediate Mode, only PTL commands are interpreted and executed. All standard BASIC statements included in the procedure are skipped during run time.

For editing of existing or new procedures, the PTL Interpreter provides two different edit functions, one being the standard Fluke Edit program which can be executed directly from within PTL, the other being the 'MENU EDIT' function of the Main Menu.

3.4.3 Compiled Mode

The PTL Compiled Mode first optimises and compiles (translates) all PTL commands before they are executed. This also means that all normal Fluke extended BASIC statements can be added to a PTL Procedure. The optimization process combines both PTL commands and BASIC statements to a file, executable from FDOS. Refer to the description of the PTL Main Menu for more information on how to compile Procedures.

3.5 EXAMPLE PTL PROCEDURE

The following is a typical example of a PTL procedure:

```
Procedure("Name='Amp Test';Programmer='P.Olar'")
  Start_Task("Nr=1;Name='Power')
    Setup("Instr=Dmm;Function=Vdc")
    Setup("Instr=Power;Output=12")
    Meas("Instr=Dmm;At=+5V;Min=4.8;Max=5.2)
    Write("To=Screen;Type=Result")
  End_Task

  Start_Task("Nr=2;Name='Gain'")
    Setup("Instr=Dmm;Function=Dbm")
    Setup("Instr=Generator;Dbm=10;Herts=1000")
    Apply("Instr=Generator;To='Amp In'")
    Meas("Instr=Dmm;At=Outp;Min=33.6;Max=34.9")
    Write("To=Screen;Type=Result")
  End_Task
End_Proc
```

3.6 STRUCTURE OF PTL PROCEDURES

A PTL Procedure consists of a number of Tasks which in turn contain a number of Steps. This definition means that the structure of a PTL Procedure always looks like:

```
PROCEDURE TEST

  START_TASK 1
    STEP

    .
    STEP
  END_TASK

  .
  START_TASK N
    STEP
    .
    .
    STEP
  END_TASK

END_PROC
```

The elements of the procedure will be explained in the following paragraphs.

3.7 PROCEDURES

A Procedure contains the elements needed to control all desired in- and output signals for testing a unit under test.
The start and end of a Procedure must be identified by means of the terms PROCEDURE and END_PROC :

```
PROCEDURE("...")
.
.     (Other procedure elements)
.
END_PROC("...")
```

Parameters can be used at both the beginning and the end of a procedure. Refer to the description of the PTL Commands for more information on parameter passing.

3.8 TASKS

A Procedure consists of one or more Tasks. Each Task consists of a number of Steps and has the following structure:

```
START_TASK("...")

      (Steps)
.
END_TASK
```

Parameters can be given at the beginning of the task. Refer to the description of the PTL Commands for more information on parameter passing.

3.9 STEPS

Each Task contains an unlimited number of Steps. These steps can be Basic statements or PTL statements (actually high-level commands, which are often composed of hundreds of normal BASIC statements).
More information on STEPS is given in the description of the PTL Commands.

3.10 BASIC STATEMENTS

As stated before, the PTL Interpreter is not a limitation, rather an enhancement of standard Fluke 1722A Extended BASIC. This means that all legal Fluke Extended BASIC statements can be used together with PTL commands without any restriction. However, BASIC statements added to a PTL Procedure are only executed when PTL operates in Compiled Mode.

Refer to "advanced programing" for further details.

3.11 BASIC SUBROUTINES

Procedures which have to be compiled with the standard FLUKE Extended BASIC Compiler, must be embedded in a legal BASIC subroutine. The structure of such a routine is:

```
SUB TEST (Task)
  On error subret
  Import Result()

     Procedure("....")
     .
     .
     End_Proc("...")

SUBEND
```

As a parameter the actual task number is being passed.
Only one procedure can be defined per BASIC subroutine.

3.12 IMPORTS

In order to better control your test, several global variables are defined, the so called imports:

Task — A number which consists of two parts, separated by a period. The part preceding the period indicates the task number, the part following the period indicates the step within the task momentarily being executed. The task number will be adjusted automatically and can be utilized in BASIC by a procedure-writer.

Pass%(0) Step pass

Pass%(1) Task Pass

Pass%(2) Procedure Pass

These three (status)flags indicate if the last executed step, task or procedure respectively, passed or failed. A particular flag will be reset to 0 if the associated part of the test failed, otherwise the flag will be set to a value < > 0.

These flags can be used in Basic, for example to repeat or skip (a part of) a test, depending on the flag value.

No_result% Contains the number of results in the Result() array the last measured or entered data.

Type% The type of results to be passed by a MEASURE, EVAL or ENTER statement.

0= Not a valid result

1= Bit value(s)

2= Integer value(s)

3= Real value(s)

4= String value(s)

Result() Contains the last numeric result from a MEASURE, EVAL or ENTER statement.

Result$ Contains the last string result from a MEASURE, EVAL or ENTER statement.

Min The minimum value specified in the last MEASURE, EVAL or ENTER statement.

Max The maximum value specified in the last MEASURE, EVAL or ENTER statement.

Units$	The units specified in the last MEASURE, EVAL or ENTER statement.

3.13 FILENAMES

Procedures are treated as normal files and are stored on floppy, e-disk or other file-structured media by name. File names consist of 1 to 6 letters,numbers,spaces,or $ characters. File names may be extended by a period character, followed by up to 3 letters, numbers, spaces, or $ characters.
Some file name extensions have a special meaning to the Fluke 1722A Instrument Controller:

.BAL	Lexical form of BASIC programs
.BAS	ASCII-text form of BASIC programs
.FD2	Binary utility programs
.HLP	System data files (Help)
.SYS	System binary programs
.CMD	Command files
.LIB	Library for assembly modules
.LBX	Library for extended basic modules
.OBX	Extended basic module

PTL adds the other extensions:

.PTL	The extension for a PTL procedure
.DRV	An instrument driver

3.14 SPOOLFILES

For serial devices, PTL knows special spool files. Instead of directly printing to the

device, PTL can store the testresults in a temporary file, which is printed at the end of a procedure. Therefore the actual running of a test procedure is not delayed because of the printing. Only at the end of the procedure, in most cases when a operator has to change boards, a delay will occur.

Serial devices are:	KB0:..KB2:	RS232 ports
	SP0:..SP3:	RS232/422 ports
	GP0:..GP1:	IEEE ports
	PC0:..PC9:	Optional PTL network

Results will be spooled when a filename is specified after the device name. E.g. KB1:SPOOL.DAT

It is necessary to specify the space available on disk for each spool file at start-up of the PTL program. Use a /S option behind PTL:
E.g.: PTL 10/S to specify 10 blocks

CHAPTER 4

SYSTEM GENERATION

4 SYSTEM GENERATION

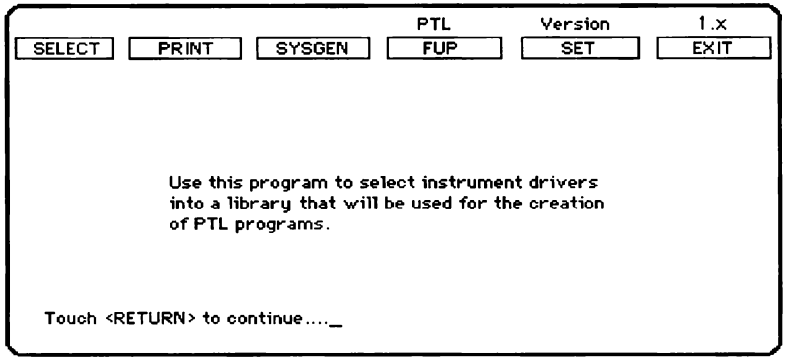
The System Generation program must be used to create a new system disk, optimised for a specific test environment. This means that only the software modules (instrument drivers) which are specified by the user, are linked together to create a new PTL program. This approach has the advantage that as much disk space is left as possible for other purposes,(editing and compiling) and that speed, especially during editing, will be improved.

The following paragraphs describe the process of generating a new system disk which contains all required instrument drivers.

4.1 GETTING STARTED

Switch on the 1722A Instrument Controller, and insert the disk with the label 'Polar Test Language System Generator'.
In case the system has already been switched on, press <RESTART> + <ABORT> simultaneously after the disk has been inserted.

After execution of the start-up command file, which copies several files to E-disk, the following screen lay-out will appear:



This indicates that the System Generator program has now been loaded and is ready to be used.

In addition, the program 'SYSGEN' can be run from the FDOS prompt by typing the filename: SYSGEN. Refer to the Fluke 1722A System Guide for information on how to run the FDOS program.

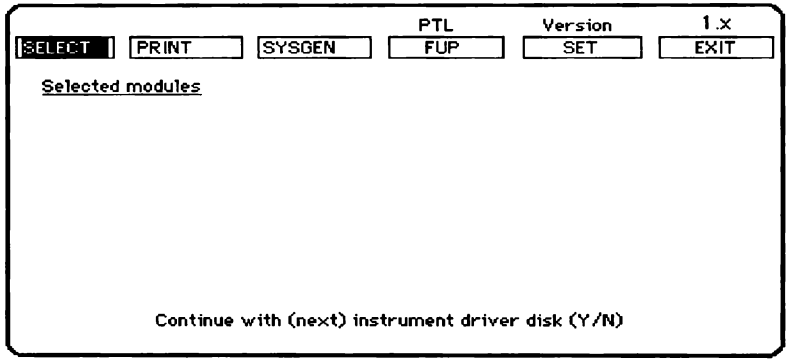
4.2 SELECT INSTRUMENT DRIVERS

To generate a system disk, the user has to specify which instruments are used in the system. This can be done by selecting software modules, called 'instrument drivers', to create new PTL system software that will 'understand' all instrument dependent commands and functions.

After <RETURN> has been pressed, use the cursor control keys to move the

reverse video field to the 'SELECT' option from the menu and press <RETURN>.

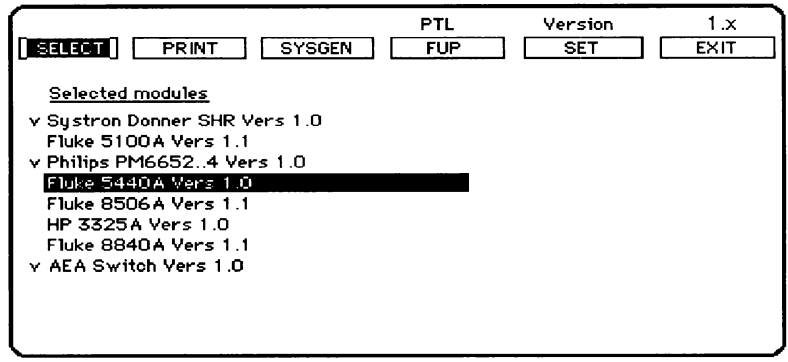
The following display appears:



To select the desired instrument drivers from the supplied driver disk(s), insert the disk 'PTL Instrument Drivers' (which contains the driver library) and press <Y>.

If one of the other menu options has to be used then respond with <N>.

After <Y> has been pressed to select drivers, the system will start reading the disk to display the names of files, recognized as instrument driver. Although all these filenames are read from disk, only the names of the first nine drivers which are found, are listed on the screen.
The following figure shows a typical 1722A display when using the 'SELECT' menu option.



If more drivers are available, then the names which are not on the screen can be made visible by using the cursor control keys.
Using these cursor control keys makes it possible to scroll or step through the list of instrument names. A reverse video field indicates the driver being selected. When <RETURN> is pressed, a 'v'-mark will appear, in front of the selected driver, to indicate all drivers currently selected. If, accidentally, a wrong driver is selected, it can be de-selected by again pressing <RETURN> after positioning the reverse video field on the driver to be deleted.

The selection process can be terminated by pressing the <ESC> key, but before the system returns to the menu, the user will be asked if the selection of instrument drivers has to be continued with a next disk.
If desired, the user can insert the next disk containing instrument drivers and press <Y>. When reading the new disk, the drivers which have been selected previously, are scrolled from the screen but can be made visible again by using the cursor control keys.

After selecting all desired instrument drivers, the selection procedure can be terminated by pressing <N> whenever the question 'Continue with (next) instrument

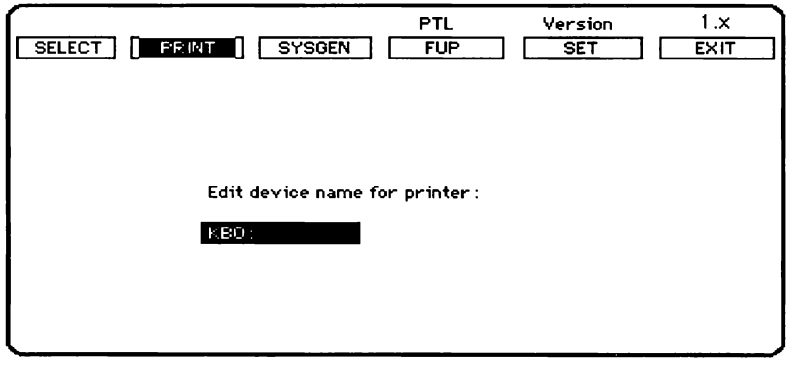
diver disk (Y/N)' is displayed on the bottom display line.

4.3 PRINT SELECTED DRIVERS

After selecting the desired instrument drivers, it is possible to list the selected drivers on the screen or on a printer, connected to one of the available printer ports.

To generate a listing of all selected drivers, the 'PRINT' function must be selected from the menu. This can be done with the cursor control keys as described in a previous section. Before the listing is sent to a printer, it is necessary to select the type of device where the listing has to be sent to. This can be done by pressing <ESC> when the 'PRINT' field is selected.

The following screen will appear:



The current printer device name is shown. All standard editing functions can be used to edit the device name and the printer device.

The following device names are selectable:

KB0:	Directs output to Screen	
KB1:	..	Serial port
KB2:	..	Serial port
SP1:	..	Serial port
SP2:	..	Serial port
GP0:2	..	IEEE address 2

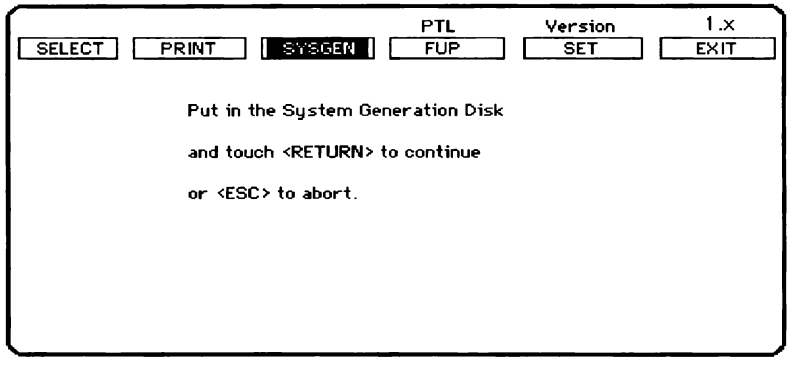
The default selection is KB1:
After typing in the desired device name, <ESC> must be pressed to confirm the input.

Pressing <RETURN> while the 'PRINT' option is selected sends a list of all selected instrument drivers to the previously defined list device.

WARNING: If an RS-232 printer is used, please make sure that the port is correctly set before printing. Refer to the standard Fluke SET utility.

4.4 GENERATE SYSTEM

To generate a new system disk, select the 'SYSGEN' function from the menu by means of the cursor control keys and press <RETURN>.
The program will inform the user that the System Generation Disk must be inserted again before system generation can be started.



After the System Generation disk has been inserted and <RETURN> is pressed, the creation of a new system disk starts automatically.

After a while the program will display the following message:

```
Please insert a blank disk
and touch <RETURN>.
```

If the disk is not empty, the system displays the following warning message:

```
This disk is not empty !!!
Touch <RETURN> to continue
or <ESC> to retry.
```

It is not necessary that the diskette has been formatted. The system will start formatting the blank diskette if this hasn't been done already. After this, the new system will be transferred to the inserted disk.

After all files have been copied onto the floppy disk, the following message will be displayed:

```
Disk ready.....
Take out and touch screen for next disk
or push <ABORT> to use this new disk.
```

By pressing <ABORT> the system performs a 'automatic' start.

4.5 PROGRAMMABLE MENU KEYS

The menu keys: FUP, SET and EXIT are 'programmable' using the <ESC> key. This provides a method to execute the most commonly used system functions without exiting the System Generator shell. Both the text, which identifies the key field, and the system command that will be executed if that particular function is activated, can be changed. Refer to the section 'System Functions' for a description of how to change the functions of the menu keys.

4.6 SIZE OF E-DISK

It is suggested to use one megabyte memory expansion for ease of use. PTL can however be run on a system with half a megabyte. To make sure that PTL will always run at delivery, PTL is shipped for a .5 MByte system.

The size of the E-disk is stored in the file "EDISK.CMD" command files. This file is a copy of either the file "1024KB.CMD" or of "512KB.CMD".

E.g. your system has 1024 KB. To configure the system, copy the file "1024KB.CMD" to "EDISK.CMD" as follows:

1. Insert the **SYSTEM GENERATOR** disk.

2. Go to fup:
   FUP

3. Assign the system to floppy:
   MF0:/a

4. Copy "1024KB.CMD" to "EDISK.CMD":
   EDISK.CMD=1024KB.CMD

CHAPTER 5

SYSTEM CONFIGURATION

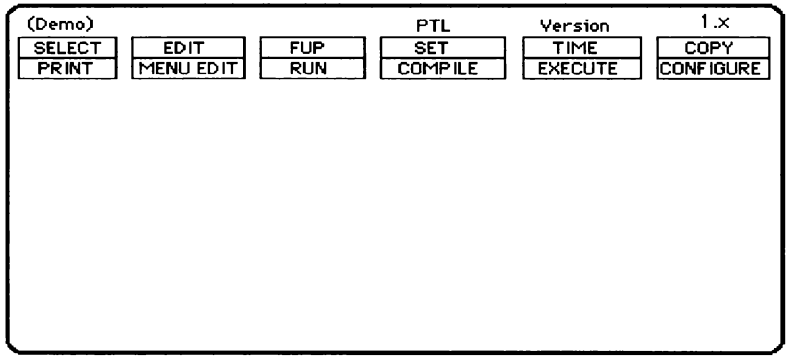
5 SYSTEM CONFIGURATION

5.1 STARTING THE PTL SYSTEM

PTL will be run automatically from the system disk at power-on or when the front panel restart key is pressed. After some time the operator disk is needed to continue and the following text is displayed:

```
Please insert the Operator disk.
Then touch the screen.
```

When PTL is run, the program produces the following menu on the display.



In addition, PTL can be run from the FDOS prompt by typing the PTL program file name: PTL. Refer to the Fluke 1722A System Guide for information how to run the FDOS program.

CAUTION

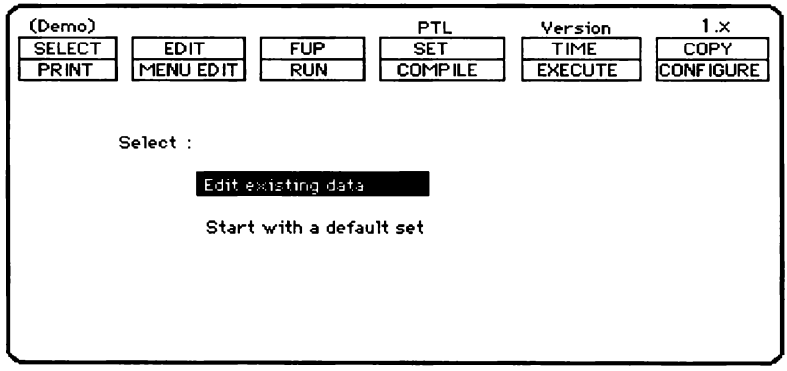
When using PTL, the E-Disk is assigned to be the system device. Files (e.g. configuration) saved on the E-Disk will be lost whenever the 1722A is switched off.

5.2 SELECT CONFIGURE

PTL offers the user the possibility to specify all system devices, from instruments to single relay switches, by attaching a unique name to each of them.
This process of defining names and relations between names is called 'Configuration' and is explained in detail in the following paragraphs, assuming that the 'Configure' option has been selected from the PTL Main Menu. The section 'System functions' describes how to select the 'Configure' option.

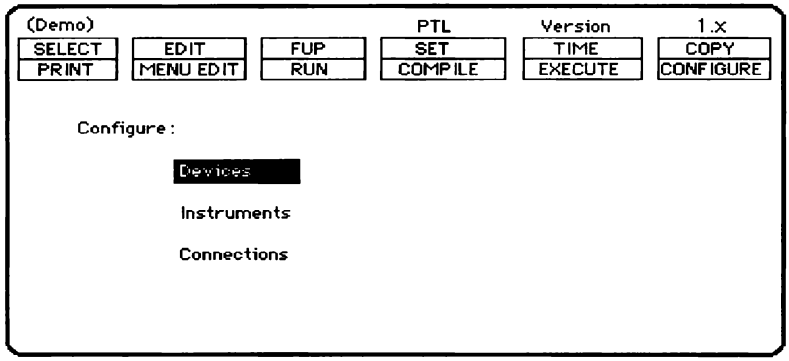
5.3 CONFIGURE

When <RETURN> is pressed after the 'Configure' option has been selected from the PTL Main Menu, the user has to specify whether to start with a default set or edit an existing set that has been configured already. The default set contains the default configurations like IEEE addresses and instrument names, as defined in the device drivers.



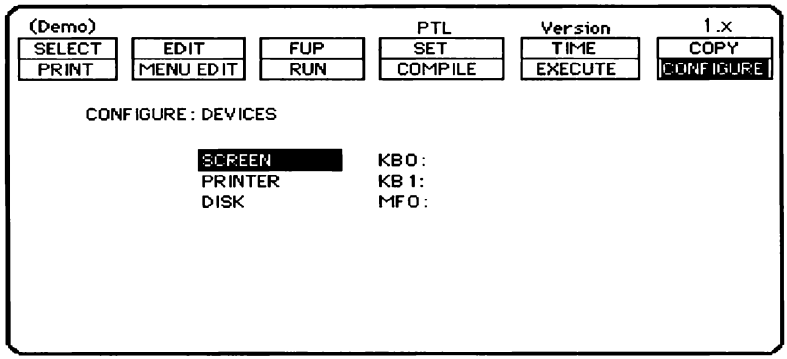
The cursor up and down keys can be used to move the reverse video field to the desired option; <RETURN> must be pressed to confirm the selection.

After a selection has been made, the following menu appears on the screen:



5.4 CONFIGURE DEVICES

This function must be used to define the names of, and the relation between logical and physical devices to write to. The display shows the menu options as shown in the following figure:



The logical device names are used later on, for example, in procedure steps like WRITE("TO=SCREEN") to output information to the 1722A display (or printer or disk).

To change the name of a device, move the reverse video block to the line and column to be edited and use the normal edit functions, like <Linefeed>, <Del line> etc., to make changes. Because PTL does not check physical device names at this phase, the user must be sure to enter a legal 1722A device name.

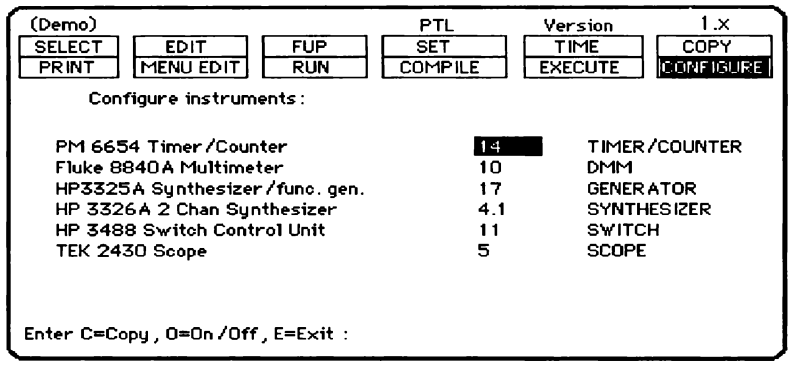
Spool files can be used to hold off printing until the procedure is finished. (see chapter 3).

E.g. KB1:SPOOL1.DAT

After configuration of devices is complete, <ESC> returns control to the Configure menu.

5.5 CONFIGURE INSTRUMENTS

When this option is selected, the display shows a list consisting of instrument driver names, IEEE addresses and instrument identifiers. The following figure shows a typical example of such an instrument list.



The instrument names in the list are the result of the driver selections, made during the System Generation process. The IEEE addresses and identifiers are either the default data from the drivers or the result of previous editing.

IEEE addresses and instrument identifiers must be unique i.e. they can be used only once in the configuration list as it is the only way for the system to identify or to access individual instruments.

If specified in the driver, secondary addressing must be used. Two types can be found:

- Real secondary addresses.
  as specified by the IEEE standard
- Pseudo secondary addresses.

Pseudo means that the instrument is considered as consisting of more modules e.g. a power unit with 4 supplies.

In both cases, the address must be specified as the main IEEE address, followed by the secondary address, separated by a decimal point.

As with most other PTL functions, the normal cursor control and editing keys can be used to step through the available list. or to change both the IEEE addresses and identifiers (=instrument names).
If, by mistake, an illegal IEEE address is specified, a default address will be substituted by the system, which can be edited again.

Pressing <ESC> displays the following options at the bottom of the screen :

C=Copy, O=On/Off, E=Exit

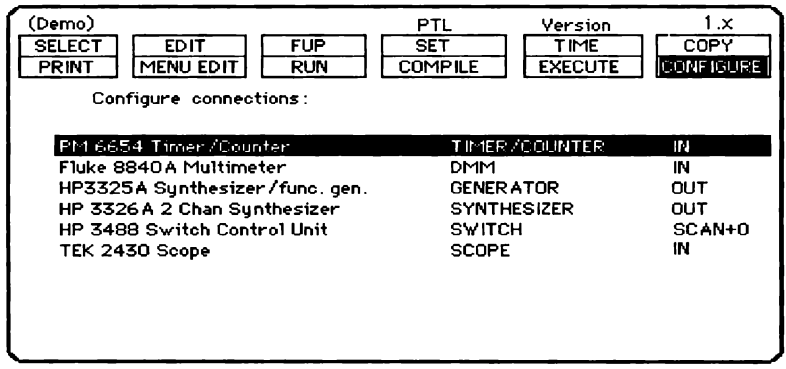
The Copy function allows the use of more then one instrument of the same type. As stated before, the identifier and IEEE address of (identical) instruments must be different, which can be accomplished by editing the appropriate fields.

The On/Off option can be used to temporarily 'remove' an instrument from the system or to 're-install' it, preventing error messages being displayed when the instrument is not available.

Finally, the Exit function must be used to return to the Configure sub-menu. A warning message will be displayed and no return takes place if two identical names or IEEE addresses are used.

5.6 CONFIGURE CONNECTIONS

This option from the Configure sub-menu must be selected when connections between individual instruments in the system and the UUT have to be defined.
When this function is selected, a list of the available instruments is displayed as, for example, shown below.



The cursor up and down keys can be used to select the instrument, of which connections in the system have to be defined. <RETURN> must be pressed to confirm a selection.

Instruments in the list are either indicated as IN, OUT or SCAN or as a combination of that. The following paragraphs explain the typical functions for all three instrument types.

IN

An instrument is indicated as 'IN' if it can handle input signals and return information to the Instrument Controller about voltage level, temperature, resistance, wave form etc. Typical examples are Digital multimeters and Oscilloscopes. The function 'MEAS' is a typical example of functions to be used for instruments of type 'IN'.

OUT

Instruments that can output anything, like voltage, current, resistance or AC signals, are defined as 'OUT' instruments. The 'APPLY' function has been especially designed to control this type of instrument.

SCAN

To make connections between system instruments and a UUT, remotely controllable switches are needed. Instruments which contain these switches are defined as 'SCAN'-type instruments. Although the function of all available switches can be defined by using the 'Configure Connections' option, individual switches can be closed by means of the PTL 'SWITCH' function.

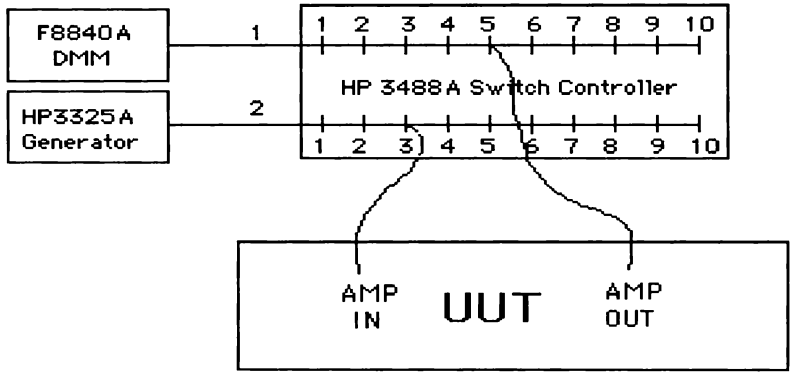
Note: Many 'intelligent' instruments today are capable to support more than one of the 'IN', 'OUT' and 'SCAN' functions simultaneously. This explains why definitions like 'SCAN+O' are used.

The following example has been included for illustration purposes only.

Example:

Assume that the input of a digital multimeter (e.g. Fluke 8840A) has to be connected to the output of an Amplifier on the UUT to measure the output level. As a stimulus, a signal generator (e.g. HP 3325) must be connected to the input of the UUT

amplifier. A Switch Control Unit (HP 3488) has been chosen to control the in- and outputs. To illustrate this, the figure below shows a diagram of the relevant hardware connections.



Assume that, during 'Configure instruments', the F8840, H3325 and H3488 have been configured as 'Multimeter', 'Signal generator' and 'Switch' respectively. Multimeter is connected to the UUT by means of relay no. 5 of scanner card no. 1 inside Switch, Signal generator by means of relay no. 3 of scanner card 2.

The connections for the multimeter are defined as follows:

Scanner	: Switch
Name	: Amplifier out
Command	: 1-5
Switch delay	: 20

For the signal generator :

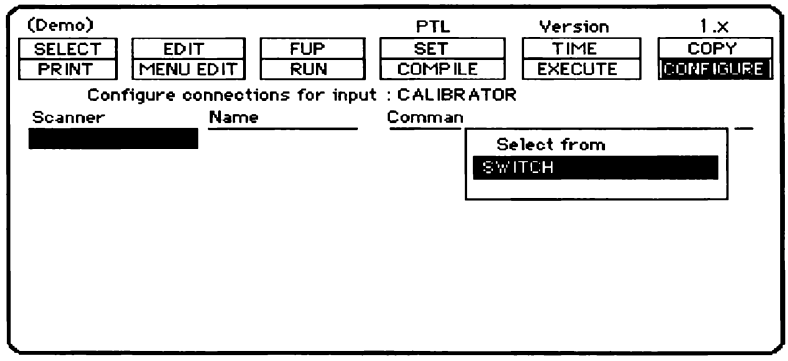
Scanner	: Switch

Name : Amplifier in
Command : 2-3
Switch delay : 20

Note: Commands are of the form x-y where x and y are legal, instrument dependent numbers. More connections may be defined on the same line by using a 'slash' character as separator e.g.
2-3/4-6 or 2-3/4-6/7-9.

To enter these configurations, first select from the 'Configure connections' menu the FLUKE 8840A Multimeter and press <RETURN>.
If no connections have been defined before, the display shows nothing else then just 'Scanner', 'Name', 'Command' and 'Switch delay'.
Changes to the displayed system connections, if any, can be made by using the standard cursor control and editing functions, while new entries only have to be typed in.

To simplify entries for the 'Scanner' column, a window will be displayed, allowing the user to select the desired 'Scan" device from the list, displayed in the window. The following figure illustrates this.



This window shows the identifiers of all instruments which are defined as 'SCAN' types; in this case the window shows nothing else then the name 'SWITCH', as it is the only instrument of type 'SCAN'. The cursor control keys can be used to select the desired instrument from the window; <RETURN> confirms a selection and returns control to the connection list.
Use the data from the given example to configure the F8840.

5.7 TEST CONFIGURATION

When these configurations have been defined, the user can return to the PTL Main Menu by pressing <ESC> several times.

For this example, use the command line to select 'MEAS' and 'Instr=Multimeter' from the menu's. Note that, in this case, because only one connection has been configured for the multimeter, the system automatically adds 'At=Amplifier out;' to the command line.
After returning to the command line entry level, select a new line by <LINEFEED> <RETURN> and select the 'APPLY' function. When 'Instr=Signal generator' is selected, the system displays 'To=Amplifier in' and returns automatically to the command line entry level, as only one connection has been defined for the Signal generator.

CHAPTER 6

SYSTEM FUNCTIONS

6 SYSTEM FUNCTIONS

File and device management, serial communications control and text editing are important functions of an operating system. The standard FLUKE 1722A FUP, SET and EDIT programs are accessible from PTL by means of easy to use menu selections. Exiting one of those programs automatically returns control to PTL.
In addition to the standard FDOS functions, PTL introduces a few new "system" functions like 'PRINT', 'CONFIGURE' and 'COMPILE'. Menu selections have been provided for these functions which would otherwise need a significant amount of user interaction.

6.1 OVERVIEW

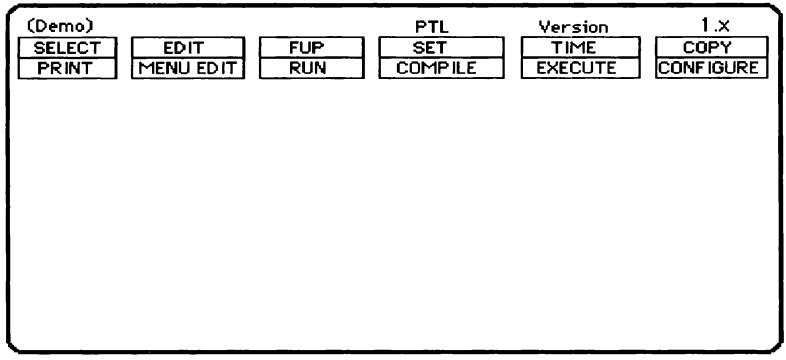
The default PTL menu offers the following functions:

SELECT EDIT FUP SET TIME FCOPY
PRINT MENU EDIT RUN COMPILE EXECUTE CONFIGURE

The upper right 5 functions are software programmable i.e. the function of these keys can be changed. For other functions, several important parameters can be changed as described in this chapter.

6.2 THE PTL MENU (SHELL)

After loading the PTL program, the normal FDOS prompt will be replaced by the following menu:



Selections from the main menu can be made by using the cursor control keys. The function momentarily selected will be indicated by a reverse video field.

6.3 DIRECT COMMAND LINES

Except for the two lines with menu keys, 10 other lines are available for immediate execution of any legal PTL command, typed in on the keyboard or selected from the available menu options. All commands entered this way will be executed when the 'Esc' key is pressed, before the system returns to the Main Menu.
Also refer to the chapter "Testing a procedure" for further information.

6.4 CHANGING THE MENU FUNCTIONS

The function of several 'keys' in the top row of the menu can be changed by pressing the ESC key. This is described in the following paragraphs.

The information about the keys is stored in a file called "PTL.DAT" on the system disk. After changing the menu functions, you should copy this file to the system disk to make them permanent.

6.5 USING THE MENU FUNCTIONS

The use of the menu functions is fully explained in the following paragraphs. Individual Main Menu functions have to be selected as follows:

1. Use the cursor control key to move the reverse video field to the field which contains the desired function name.
2. Press the 'Return' or 'Esc' key, depending on the action required as described below for each function individually.

For the following descriptions, it is assumed that the described function has been selected from the Main Menu as explained earlier.
For each function always two descriptions are given i.e. one explains what will happen when the <Esc> key is pressed, the other describes the effect of pressing the 'Return' key after the function is selected.

6.5.1 SELECT

The 'SELECT' function key can be used in two different ways:

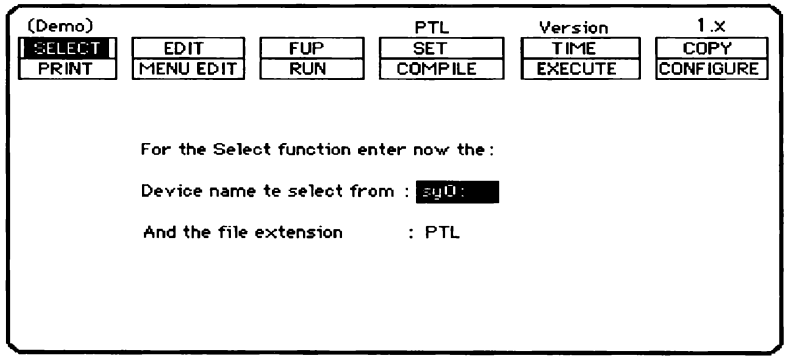
1. To select the file to be used by other Main Menu functions.

2. To define the device name and the filename extension.

The following sequences will be recognized by the PTL program:

SELECT/<Esc>

To change the default device identifier or the filename extension, press 'Esc'. The 'SELECT' field starts flashing and the following display appears:



All normal edit functions can be used to make changes to the default device and extension.
After the required changes have been made, a return to the Main Menu can be made by pressing the 'Esc' key or, if the extension field is selected, the 'Return' key.

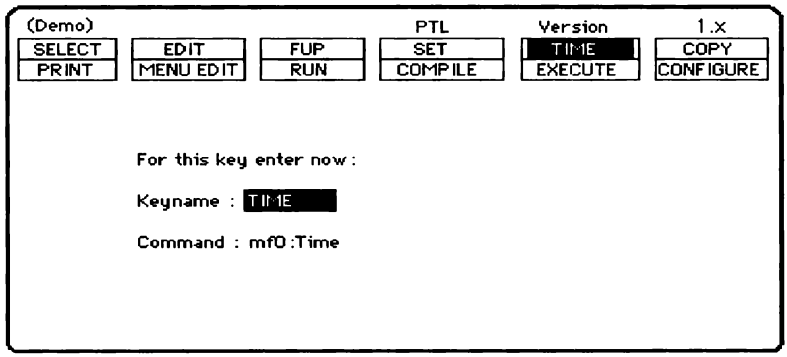
SELECT/<Return>

To select a file, press <RETURN>. This forces the system to display all files from the device and with the extension previously selected by means of the SELECT/<ESC> function.
The 'SELECT' field will be highlighted to indicate that this function is active.
The cursor control keys can be used to step through the list of filenames. The file presently selected is indicated by a reverse video block. After the desired file has been selected this way, both <ESC> or <RETURN> may be pressed to confirm the selection.
If the text '<NEW>' is selected and <RETURN> is pressed, the user will be asked to type in the name of the new file.
After the filename has been typed and 'Return' is pressed again, the filename will be checked for validity. If no errors are detected, the system returns to the Main Menu, displaying the selected filename in the upper left hand corner of the screen.
If an illegal filename is used, a general error message will be displayed together with several options to retry, abort or exit from the error handler.

6.5.2 EDIT,FUP,SET,TIME,FCOPY

The function of those fields is to provide 5 user programmable function keys to execute the most commonly used system functions.

The user can change both the text which identifies the key field and the system command that will be executed if that particular function is activated. After selecting one of those fields, e.g. 'TIME', the following message is displayed after <ESC> is pressed:



'Keyname' is the name used in the menu to identify a menu key and 'Command' the function to be executed when that specific key is selected and <RETURN> is pressed.
The 'Keyname' or 'Command' fields can be selected with the normal cursor control keys.

Old text or typing errors can be deleted with the <DEL LINE> or <DEL CHAR> keys. The use of the <RETURN> key is optional.
Pressing <ESC> returns control to the Main Menu. The selected field always reflects any change that has been made to the key identifier.

Changes made to the menu are automatically saved on the system device (SY0:).

If <RETURN> is pressed then the command associated with the selected function will be executed.
When the execution is terminated, the system returns to the PTL Main Menu, except when the 'EXIT' command is selected. This command returns control permanently to FDOS.

A command can be any valid FDOS command i.e. the calling of a command file or of a program. As with normal command files and programs, parameters can be passed. In this case three special parameters are available:

?: The device selected under SELECT/ESC
? The filename selected with SELECT/RETURN
.? The extension selected under SELECT/ESC

To call the Fluke editor e.g. the command

```
MF0:EDIT ?:?.?
```

could be used

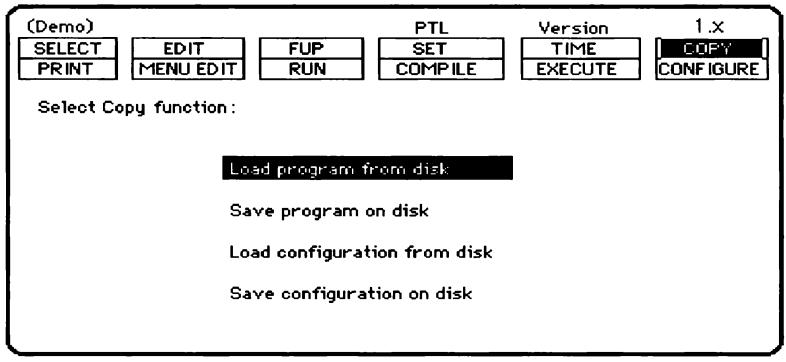
For the functions EDIT, FUP, SET and TIME refer to the Fluke 17xxA manuals.

FCOPY

The program FCOPY provides a more friendly way to copy files between memory and disk than the standard FUP program. However, FCOPY can only be used to transfer procedure and configuration files.

```
FCOPY/<RETURN>
```

Pressing <RETURN> displays the following menu:



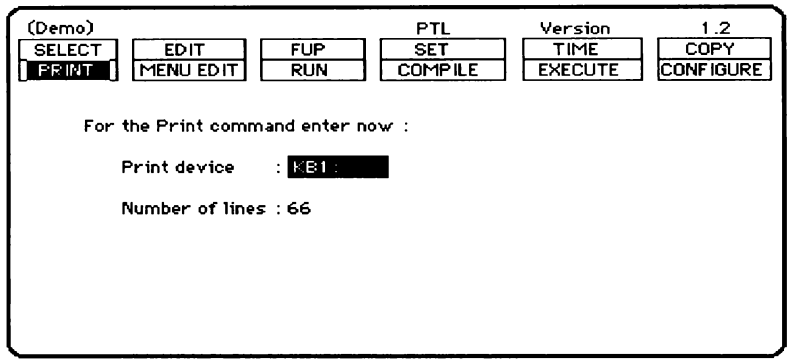
A selection can be made by moving the reverse video field to the appropriate menu option. <RETURN> must be pressed to start the file transfer, <ESC> can be used to abort the operation and return to the main menu.

6.5.3 PRINT

The PRINT function can be used to make a hard copy of programs, the system configuration or the capabilities of the drivers (i.e. a list of all instrument dependent PTL 'statements' and functions, available)

```
PRINT/<ESC>
```

Pressing the <ESC> key offers the possibility to change the print device as indicated in the following figure.



The following device names can be selected:

KB0:	Directs output to Screen	
KB1:	..	Serial port
KB2:	..	Serial port
SP1:	..	Serial port
SP2:	..	Serial port
GP0:2	..	IEEE address 2

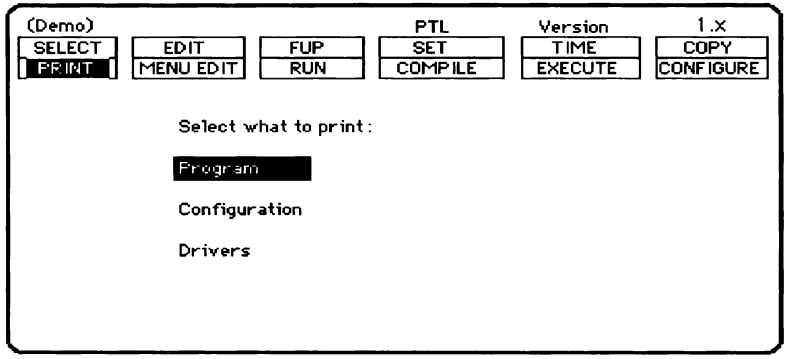
The default selection is KB1:. Spoolfiles are allowed, e.g. KB1:SPOOL1.DAT.

The number of lines of a page must be set to be equal to the paper used. This ensures that each new page will end with a page number on the last line. If the number of lines equals 0, then no page brakes will be generated.

As with other functions, all standard edit functions can be used to change the indicated device name.

PRINT/<RETURN>

After <RETURN> is pressed, the display shows the following print menu:



PRINT PROGRAM

This function sends the contents of the program file, selected by means of the 'SELECT' function and displayed on top of the PTL main menu, to the output device, selected with the PRINT/<ESC> function. The appendix contains an example.

PRINT CONFIGURATION

The complete system configuration will be listed when this option is used. The Appendix contains an example of such a listing.

PRINT DRIVERS

Instrument dependent PTL 'statements' and functions are part of the instrument

drivers. When developing a PTL procedure, this information is needed to create the test procedure.

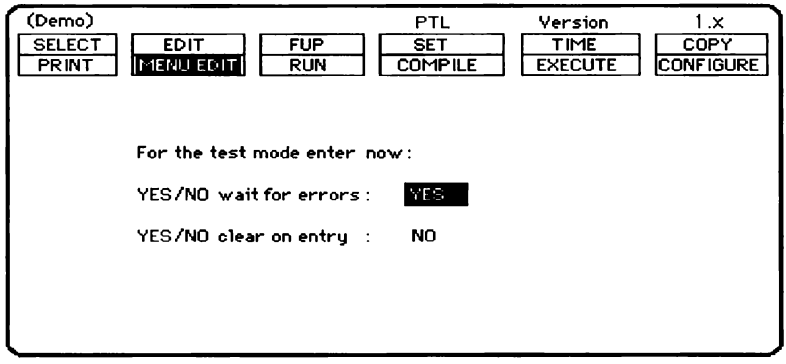
The PRINT DRIVERS option provides a way to list all driver parameters, including the associated (IEEE) control codes of those drivers, selected in the configuration. The appendix contains a sample printout of a typical driver listing.

6.5.4 MENU EDIT (Optional)

This function allows the user to create or edit test procedures by simply making selections from several, in many cases instrument-type dependent, sub-menu's.
With this method, little knowledge is needed to produce test procedures for even complicated test systems.
As all menu options, which can be selected, are displayed simultaneously, 'programming' means moving around with the cursor control keys and 'picking up' required functions by pressing <RETURN>.

```
MENU EDIT/<ESC>
```

This sequence allows the user to define two different options for direct execution (immediate mode). The following will be displayed:



The displayed options allow the user to define:

- o whether execution of a procedure has to be suspended or not whenever an 'error' condition is detected.

- o if the 'command lines' have to be cleared or not each time they are used.

MENU EDIT/<RETURN>

This sequence starts the 'MENU EDIT' function and is described in detail in the section 'Writing And Editing A Procedure'.

6.5.5 RUN

The 'RUN' option from the Main Menu must be used to execute a PTL procedure in Interpreted mode. The only sequence accepted by the system is RUN/<RETURN>.

RUN/<RETURN>

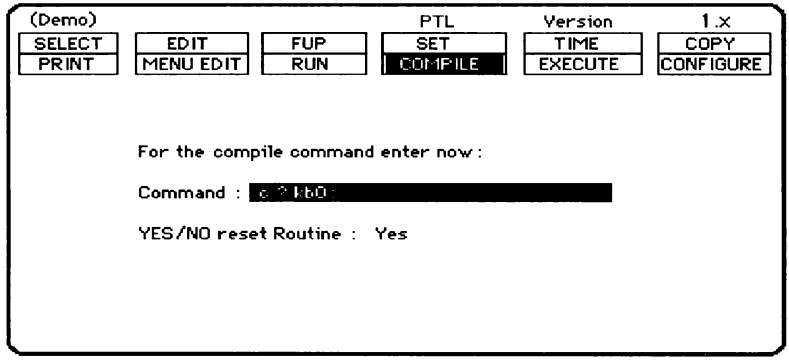
This sequence results in the execution of the procedure, selected by the 'SELECT' function and indicated in the top left hand corner of the display. Normal BASIC statements, if included in a procedure, are not executed.
After execution, the system returns to the PTL Main Menu.

6.5.6 COMPILE

If it is desired to compile a procedure, the 'COMPILE' function must be selected from the PTL Main Menu. This will start execution of a system Command File which handles all commands to perform the required compilation and linking tasks. The process of compiling and linking is described in the section Compiling And Linking.

COMPILE/<ESC>

If selected, the following menu appears:



The command gives the calling of the command file needed to compile the procedure.

Also it can be selected whether to Reset all instruments before executing the

procedure.

Also refer to the section "Advanced programming" for further information.

6.5.7 EXECUTE

After a procedure has been compiled, it can be executed by means of the 'EXECUTE' function.

```
EXECUTE/<RETURN>
```

This sequence starts execution of the procedure file which has previously been selected with the 'SELECT' function but with a file extension 'FD2', no matter which extension has been selected with the 'SELECT/<ESC>' sequence.

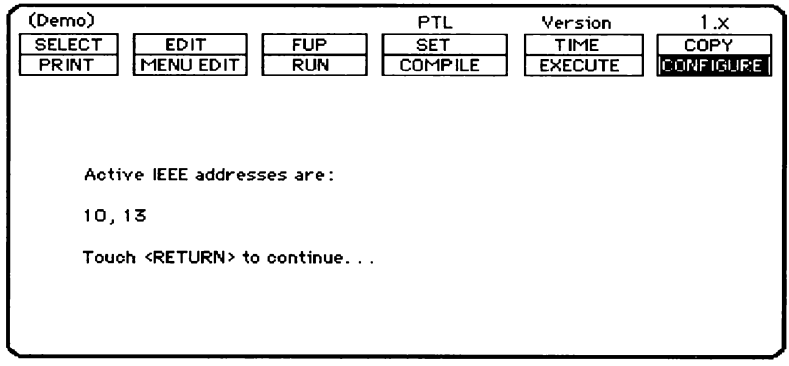
The `EXECUTE/ESC` sequence has no effect.

6.5.8 CONFIGURE

The 'CONFIGURE' function is intended to define system parameters like I/O devices, IEEE addresses, instrument identifiers (names) and connections.

```
CONFIGURE/<ESC>
```

To determine which IEEE addresses are used by the instruments, connected to the Instrument Controller, the sequence 'CONFIGURE/<ESC>' can be used. This function returns the IEEE addresses of active instruments in the system. The following figure shows an example.



CONFIGURE/<RETURN>

The sequence 'CONFIGURE/<RETURN>' allows to configure the system. Refer to chapter "SYSTEM CONFIGURATION" for a detailed description.

CHAPTER 7

PTL COMMAND REFERENCE

7 PTL COMMAND REFERENCE

PTL Commands are predefined operations available in the Immediate, Interpreter and Compiled Modes. The availability of those commands can significantly simplify the process of writing a procedure.
In addition, a wide range of parameters is allowed for each command to adjust its function to an appropriate task.

7.1 OVERVIEW

This section is divided into four subject areas: General Purpose Commands, Instrument Control Commands, Input/output Commands and Special Function Commands.
Only a little instrumentation background is required for use of any of the PTL Commands.

PTL Commands always consists of a "command" field and one or more optional "parameter" fields between brackets i.e.: Command("Parameters").

The following commands are recognized by PTL :

PROCEDURE("...")	Starts a PTL procedure
END_PROC("...")	Terminates a PTL procedure
START_TASK("...")	Starts a new task within a PTL procedure.
END_TASK	Indicates the end of a PTL task.
RESET("...")	Resets an instrument to a known state.
SETUP("...")	Programs an instrument for a specific function.
MEAS("...")	Instructs an instrument to take a new measurement.
APPLY("...")	Enables the designated instrument.
DISAPPLY("...")	Disables the designated instrument.
SWITCH("...")	Switches on or off the designated connection.

DISCONNECT("...")	Disconnects the designated instrument.
WRITE("...")	Outputs text or results to screen, printer or disk.
ENTER("...")	Instructs the operator to input information.
DELAY("...")	Delays the execution of a procedure.
EVAL("...")	Evaluates a result.

In addition, the ampersand (&) character can be used to split a command line into two or more physical lines on the screen. Single language elements like commands, numeric constants, may not be continued on a next line.
For long strings the plus (+) character can be used as shown in the next example:

```
WRITE    ("TO=SCREEN;TEXT='A string like this one may NOT be
          continued on a new line
          like this.'")
```

This example line is illegal because the string is broken in the middle of the line. A long string may be broken in the following manner:

```
WRITE ("TO=SCREEN;TEXT='A string like this one" &  + "may be continued
on a new line like this.'")
```

NOTE: The ampersand **must** be the **last** character of a line. Even spaces are not allowed behind an ampersand. Use the <LineFeed> key to go to the end of the line.

7.2 STRUCTURAL COMMANDS

The General Purpose Commands supplied with PTL allow for easy structuring of a procedure.
This version of PTL supplies the following General Purpose Commands:

```
PROCEDURE("...")
END_PROC("...")
START_TASK("...")
END_TASK
```

This section describes those commands and their optional parameters in logical order.

7.2.1 PROCEDURE( )

Format: PROCEDURE[("parameter;...parameter;][parameter;...parameter;]")

Parameters:		
	NAME='...' ;	The name of this particular PTL Procedure.
	VERSION='...';	The version of a PTL Procedure.
	PROG='...';	The name of the procedure writer.
	DATE='...' ;	The date the procedure was created.
	OPERATOR='...';	Inquiry after the operator name.
	S/N='...' ;	Inquiry after an instrument serial number.

Each PTL Procedure must begin with the PTL Command 'PROCEDURE'. This Command initializes the required system resources.

Name and version of the procedure, the name of the programmer and the date are used for documentation purposes and can be included, if desired, in test reports.

If an inquiry after the operator name has been specified, the operator will be asked

by means of a user defined display message to enter a name, using the alphanumeric keyboard drawn on the screen. This inquiry will only be made once, at the start of a procedure. Each new time PROCEDURE will be encountered before the END_PROC, the inquiry will be skipped.

If an inquiry after the serial number has been included, a numeric keypad will be displayed and the operator will be asked to input a serial number by means of a user defined display message, before execution of the procedure continues. The inquiry will be done each time the PROCEDURE statement will be encountered.

Example:

```
PROCEDURE("NAME='Example';VERSION='1.0';"      &
      +"PROG='P. Olar';DATE='21/09/91';"    &
      +"OPERATOR='Enter your name';"        &
      +"S/N='Enter serial number'")
```

Details:

The procedure statement has the following functions:

- Store parameters
- If required and task=-1: ask for operator input
- If required: ask for serial number
- Make task=0

7.2.2 END_PROC

Format: END_PROC[("parameter;")]

Parameters: CONTINUE='...' Asks the operator if the procedure has to be repeated.

Every PTL Procedure must be terminated by an 'END_PROC' command, to indicate the end of a procedure. If the statement is not found, the procedure will continue to run until stopped manually (by CTRL/C)

The optional parameter can be used to ask the operator if the procedure has to be repeated or not, by means of a user defined display message and a Yes/No keypad on the screen. If testing has to be continued (i.e. the 'Yes' key has been pressed), the number of the Task to be executed next will be preset to 1, forcing the system to start at the beginning of the procedure. Otherwise the Task number will be reset to 0 which terminates execution of the procedure. If a 'null' string (i.e. <return> only) is entered after 'CONTINUE=', then execution of the procedure will be terminated as if no 'CONTINUE=' was used.

Example:

```
END_PROC("CONTINUE='Test a new modem?'")
```

Details:

The end_procedure has the following functions:
- If required: ask for operator input
- If not to stop: make task=0

7.2.3 START_TASK()

Format: START_TASK [("parameter")]

Parameters: NAME='...' Identifies a particular procedure task.

PTL procedures may contain a number of tasks. The command START_TASK identifies the beginning of a task. The text, entered for the optional parameter 'NAME' can be used to explain the function of the task and will be displayed during execution of that specific task.

Example:

```
START_TASK("NAME='Initialize'")
```

Details:

The functions are:
- Store the parameters entered
- Task= Task+1

7.2.4 END_TASK

Format: END_TASK

Parameters: None

The command END_TASK must be used to identify the end of a Procedure Task.

Example:

```
END_TASK
```

Details: none

7.3 INSTRUMENT CONTROL COMMANDS

PTL also includes three Instrument Control Commands: RESET, SETUP and MEAS. These functions allow for complete control of any function of the instruments in the system, like function and range selections, sample rate or triggering.

7.3.1 RESET

Format: RESET("parameters")

Parameters: INSTR=... Reset the indicated instrument.

The RESET command can be used to reset any instrument in the system to its default conditions. Always refer to the user manual of the instrument in question for information on those default conditions.

Example:

```
RESET("INSTR=DMM")
```

NOTE: In each compiled procedure, a complete reset of all instruments is automatically included at the start of the procedure.

Details:

The driver is called with a reset command.

7.3.2 SETUP

Format: SETUP("parameters")

Parameters:		
	INSTR=...	Indicates instrument to be 'set up'.
	????	
		Instrument dependent parameters
	.	
	????	

The SETUP command can be used to prepare the individual instruments in the system for a specific task. The command includes parameters for all instrument dependent functions like range selection, function control, measurement speed etc. Refer to the print-out of the driver of the particular instruments for detailed information on all remote-controllable functions.

Example:

```
SETUP("INSTR=DMM;FUNCTION=dBm;RANGE=-60")
```

Details:

The driver is called with the setup parameters

7.3.3 MEAS

Format: MEAS("parameters")

Parameters:		
	INSTR=...	Indicates instrument to be used for a measurement.
	AT=...	Identifies the connection(s) to be made by the system.

MIN=...	Minimum value allowed for the measurement.
MAX=...	Maximum value allowed for the measurement.
UNITS=...	Engineering units to be used for the measurement.
REMARK=...	A remark to be printed

The MEAS command can be used to perform measurements on the unit under test, using the indicated instrument. The command includes all "EVAL" parameters for minimum and maximum values to generate pass/fail information as the result of a specific measurement. 'UNITS' are displayed together with the measurement value.

If specified, a user defined REPORT subroutine is performed after each "MEAS" statement.

A 'MEAS' command normally follows a 'SETUP' command.

Example:

```
MEAS("INSTR=COUNTER;AT=Tp5;MIN=15.5;MAX=16;"  &
     +"UNITS='KHz'")
```

Note: A new MEAS command automatically 'resets' connections, defined by a preceding MEAS command.

Example:

```
MEAS("Instr=Counter;At=Tp5") : Connect Tp5
     MEAS("Instr=Counter;At=Tp6") : Disconnect Tp5
                          + Connect Tp6
```

Details:

The following functions are performed:

- The scanner driver is called with the codes specified in the configuration to set the relays.
- The measurement instrument driver is called to take measurements and store them in RESULT.
- The evaluation function is called to check the results.
- The REPORT subroutine is called

7.3.4 APPLY

Format: APPLY("Parameters")

Parameters:

INSTR=...	Indicates instrument to be enabled.
TO=...	Indicates connection to be made by an instrument defined as 'Scan'

The APPLY command can be used to enable the output of an instrument, normally after it has been set up with the SETUP command. The parameter 'TO' is relevant only if the INSTR parameter designates an instrument that is defined as 'SCAN'. In such a case, the effect of the APPLY command is that a connection will be made, according to the configurations of the 'SCAN' instrument. Refer to the section 'System Configurations' in this manual for more information on how to configure Devices, Instruments and Connections.

Example:

```
APPLY("Instr=Synthesizer;To=Amp input")
```

Notes:

- Synthesizer and 'Amp input' are names, chosen by the user during configuration.
- A new APPLY command automatically 'resets' connections, defined by a preceding APPLY command.

Example:

```
APPLY("Instr=Synthesizer;To=Amp input")
          Connect Synth. to 'Amp input'

     APPLY("Instr=Synthesizer;To=Mod input")
          Disconnect from 'Amp input' + connect to
          'Mod input'
```

Details:

The scanner driver is called with the parameters in the configuration file

7.3.5 DISAPPLY

Format: DISAPPLY("Parameters")

Parameters:

INSTR=...	Indicates instrument to be disabled.
TO=...	Indicates the connection to be interrupted by an instrument defined as 'Scan'.

The DISAPPLY is the opposite of the APPLY command and may be used to disable the output of instruments, previously enabled with the APPLY command. The parameter 'TO' is relevant only if the INSTR parameter designates an instrument that is defined as 'SCAN'. In such a case, the effect of the DISAPPLY command is that the connection, specified in the configurations of the 'SCAN' instrument, will be interrupted. Refer to the section 'System Configurations' in this manual for more information on how to configure Devices, Instruments and Connections.

Example:

DISAPPLY("Instr=Power-supply")

Note: 'Power-supply' is a name, chosen by the user during configuration.

Details:

The driver is called with the parameters in the configuration file

7.3.6 SWITCH

Format: SWITCH("parameters")

Parameters: '...'=On/Off Identifies the connection to be switched. The connection can either be switched on or off.

The SWITCH command can be used to control individual system connections by using names, rather than low level identifiers like numbers of scanner cards,relays etc.,
Names are defined during configuration of the 'SCAN' instrument(s).

Example:

```
SWITCH("Feedback=On;Freq comp=Off")
```

Note: 'Feedback' and 'Freq comp' are names for switchable system connections, chosen by the user during configuration.

Details:

The driver is called with the parameters in the configuration file

7.3.7 DISCONNECT

Format: DISCONNECT("parameter")

Parameters: INSTR=... Specifies the instrument to be disconnected.

The result of a DISCONNECT command is that **all** switches in the system, which have been closed for the designated instrument, will be opened.

Example:

```
DISCONNECT("Instr=Counter")
```

Note: 'Counter' is a name, chosen by the user during configuration.

Details:

All scanner drivers associated in the configuration file with the instrument are being called to disconnect the instrument

7.4 INPUT/OUTPUT COMMANDS

7.4.1 WRITE

Format: WRITE("parameters")

Parameters:		
	TO=(Options)	Directs the output to a selectable device.
Options:	SCREEN	Directs output to the device 'screen'.
	PRINTER	Directs output to the device 'printer'.
	DISK	Directs output to the device 'disk'.
TYPE=(options)		Indicates the type of data to be output to the device, selected with the 'TO ' parameter.
Options:	HEADER	Outputs a header
	SHORT HEADER	Outputs a shortened header.
	RESULT	Outputs the last result.
	PASS	Outputs the last result if a step passed.
	PASS STEP	Outputs the last result if a step passed.
	PASS TASK	Outputs results if a task passed.
	PASS PROC	Outputs results if a procedure passed.
	FAIL	Outputs the last result if a step failed.
	FAIL STEP	Outputs the last result if a step failed.
	FAIL TASK	Outputs results if a task failed.
	FAIL PROC	Outputs results if a procedure failed.
	PASS/FAIL	Outputs the last result if a step passed.
	PASS/FAIL STEP	Outputs the last result
	PASS/FAIL TASK	Outputs the result of a task.
	PASS/FAIL PROC	Outputs the result of a procedure.
REMARK='...'		Outputs a user defined text to the device,

selected with the 'TO ' parameter.

TEXT='...' Outputs a user defined text

The WRITE command can be used to output information to a user selectable device. The output must be defined with the 'TYPE' parameter and can be used to control the layout and contents of test reports.
When the 'HEADER' type is used, a header will be displayed or printed as the first full page of a report. This header contains the following items:

Procedure version	Step
Programmer name	Comment
System configuration	Minimum
Test	Actual
Unit	Maximum
Operator name	Units
Date	Pass/Fail

An example of the screen lay-out is given below.

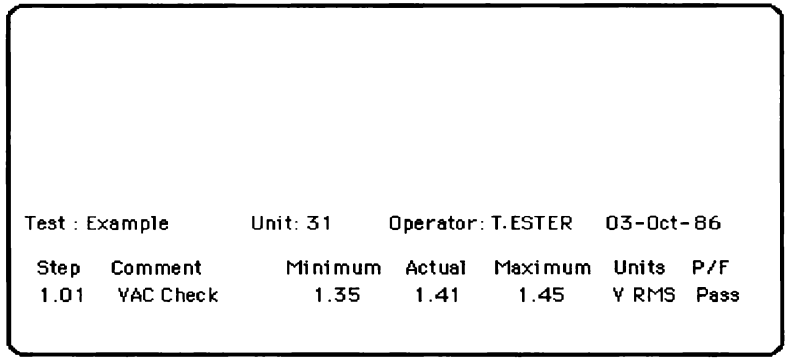
```
Test          : Example                    Test date : 03-Oct-88 14:22
Version       : 1.x / 21-08-88
Programmer    : H. Bär
Configuration: 03-Oct-88 13:02 / PTL 1.x

Operator      : T ESTER
Unit          : 31          Operation mode : AUTO

Task : Set-up

 Step   Comment        Minimum   Actual   Maximum   Units   P/F
 1.01   VAC Check        1.35     1.41      1.45   V RMS   Pass
```

The 'SHORT HEADER' type differs from the 'HEADER' type in several ways. First, only the Test, Unit, Operator name and Date will be output. Second, only a few lines of the first page of the report will be reserved for the header. Refer to the figure below for a representation of the 'SHORT HEADER' screen lay-out.



Test results always contain the following items:

Task number
Remarks
Minimum value
Actual value
Maximum value
Units

Example:

```
WRITE("TO=PRINTER;TYPE=PASS_TASK; REMARK='Test'")
```

Details:

The results are being written to the device in the configuration file

7.4.2 REPORT

Report is a special subroutine which is called after each evaluation function, to automatically perform one or more WRITE statements. The standard REPORT subroutine is empty. However the user may put a special REPORT subroutine with the appropriate WRITE statements before the test subroutine. This will then replace the standard empty subroutine.

Example:

```
Sub Report
  WRITE ("To=Screen;Type=Result")
  WRITE ("To=Printer;Type=Fail")
Subend
Sub Test (Task)
  ---
```

Details:

in the PTL library, there is an empty Report subroutine, which will not be linked if resolved beforehand

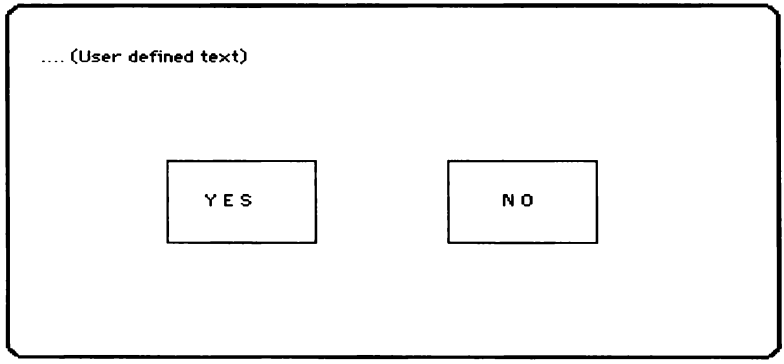
7.4.3 ENTER

Format: ENTER("parameters")

Parameters:	WAIT='...'	Halts execution of a procedure until the operator touches the 1722A screen.
	QUESTION='...'	Asks the operator a user defined question.
	NUMBER='...'	Waits for the operator to input a number.
	STRING='...'	Waits for the operator to input text.

The WAIT parameter can be used to temporarily stop execution of a procedure. A user defined text can be used for operator instructions etc.
This blinking, highlighted message will be displayed at the bottom display line. Execution continues after the operator has touched the 1722A screen.

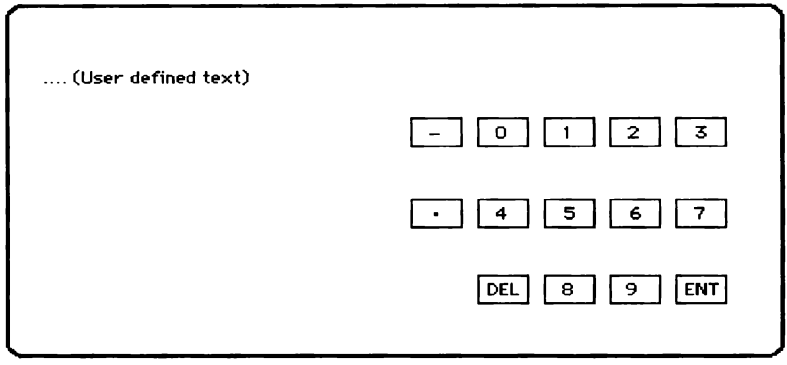
The parameter 'QUESTION' can be used to get simple yes/no input from the operator. A user defined text (question) will be output to the screen, together with YES and No key fields as illustrated in the figure below.



The operator can make a selection by pressing the appropriate key field on the screen. The result will be stored in RESULT(0).

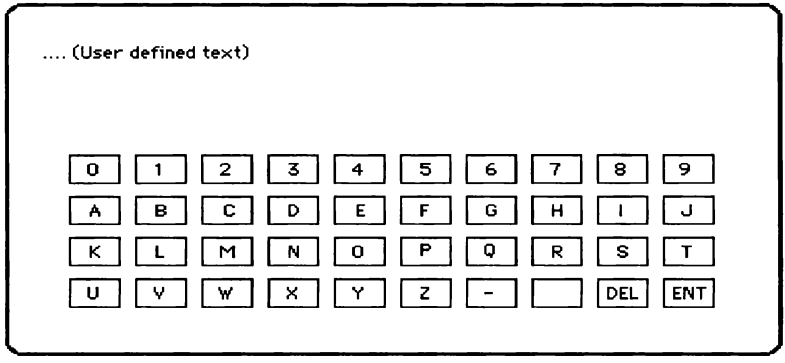
No=0, Yes< >0

Whenever a number is required during execution of a procedure, the 'NUMBER' parameter can be used. The user defined text will be placed on the screen, together with a numeric keypad, which can be used by the operator to enter the required number. The figure below shows the screen when this function is used.



The result will be stored in RESULT(0)

The STRING parameter is almost equivalent to the 'NUMBER' parameter. The only difference is that instead of a numeric keypad, an alpha-numeric keyboard will be drawn on the screen to allow alpha-numeric input as can be seen in the following figure.



The result will be stored in RESULT$.

Example:

ENTER("QUESTION='Measure between 1.23 and 1.34'")

Details:

- The operator is asked to enter the response
- The result is put in RESULT in the form:
  NO = 0, YES <> 0 in RESULT(0), TYPE%=1
  Number in RESULT(0), TYPE%=3
  String in RESULT$, TYPE%=4
- The result is evaluated against the specified limits
- The REPORT subroutine is called

7.5 SPECIAL COMMANDS

7.5.1 DELAY

Format: DELAY("parameters")

Parameters: TIME=(value) The delay time in milliseconds

The 'DELAY' command can be used to slow down execution of a procedure. This can be necessary, for example, to allow set-up time for slow instruments like DMM's.
The 'TIME' value must be in the range 0..100000.

Example:

```
DELAY("TIME=65")
```

Details:
- The controller halts during the specified time.

7.5.2 EVAL

Format: EVAL("parameters")

Parameters:

MIN=...	Minimum value allowed for the measurement.
MAX=...	Maximum value allowed for the measurement.
UNITS=	Engineering units to be used for the measurement.
REMARK=	A remark to be printed

Example:

```
EVAL("Min=12;Max=12.4;Units='V'")
```

Details:
- The previously gathered result in RESULT() will be evaluated to see whether it is a pass or a fail.
- As many results as given in NO_RESULTS% will be evaluated. Only the total result will be stored in PASS% i.e. PASS% = 1 if **all** results passed.
- The type of evaluation depends on TYPE%:

0: No evaluation done

1: Each individual bit in Result will be checked against the corresponding limit bit values in min and max.

E.g. if bit 5 in RESULT()=1, in MIN=0 and in MAX=0 then PASS%=0

2: A check is done on RESULT against the MIN and MAX limits

3: A check is done on RESULT against the MIN and MAX limits

4: PASS%=0

- If a subroutine REPORT is included in the procedure, EVAL will call this routine to output its results.

CHAPTER 8

EDITING A PROCEDURE

8 EDITING A PROCEDURE

Writing a procedure is the first step toward getting the controller to do your bidding. This section describes the process of entering a procedure into memory of the controller and making corrections to that procedure.

8.1 OVERVIEW

A PTL procedure is a meaningful sequence of PTL commands, possibly supplemented by normal BASIC statements that (in RUN Mode) directs the Instrument Controller and its associated instrumentation to accomplish a desired task.

8.2 FUNCTIONS FOR WRITING/EDITING

The PTL menu offers two different functions for writing or editing PTL procedures: EDIT and (optionally) MENU EDIT.

The first method is to use the standard Fluke 1722A Edit program, with all its standard features as described in Section 6 of the 1722A System Guide.
This method can be selected by means of the EDIT field in the PTL Main Menu.

The second and easiest method is the use of the MENU function. This method is recommended for inexperienced users. Although with this method procedures are generated, which contain PTL commands only, BASIC statements can be added at any time if desired.
The rest of this chapter contains more information on how to use the menu selections.
As an alternative to the methods described above,the use of any other text processing system to generate procedures can be mentioned. This approach has the advantage that new procedures can be developed on a low cost system while the Instrument Controller continues execution of previously written test procedures, thereby

increasing the overall system efficiency significantly.
Procedure files, when generated on a second computer, have to be transferred to the 1722A Instrument Controller which, for example, can be accomplished with the standard 1722A FUP Utility or a simple BASIC program.

Files created in one of the methods mentioned above, can be edited, using any of the other methods.

8.3 USING MENU EDIT

Although the name implies an edit function, "MENU EDIT" can be used to both generate new procedures as well to edit existing procedures.
When the 'MENU EDIT' function has been selected as described in the chapter "System Functions", the first page of the procedure, selected with the "SELECT" function (indicated in the top left corner of the screen) will be displayed.
The MENU EDIT has a selected set of functions, compatible to the standard EDIT program. On top of this it has a powerful menu function.

The standard 1722A edit keys (i.e. the cursor control keys, DELETE, DEL LINE, DEL CHAR, BACKSPACE and LINE FEED) can be used to add or delete text.

Also the following commands are implemented:

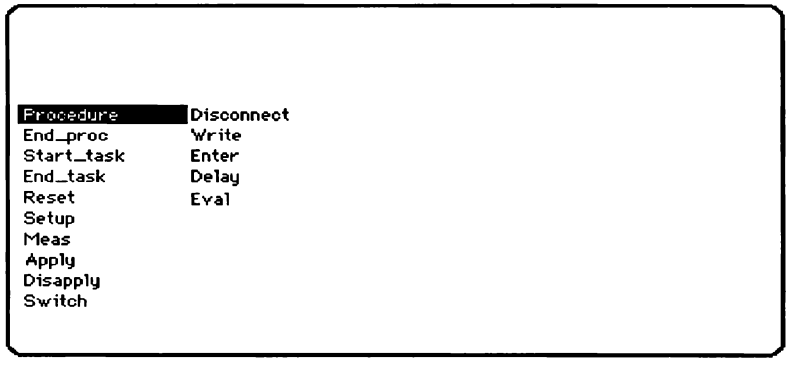
<ESC> n G	Go to line n (If n=0 go to end)
<ESC> n ^	Go n lines up
<ESC> n v	Go n lines down
<ESC> / text	Get the text specified looking downward
<ESC> ? text	Get the text specified looking upwards
<ESC> n	Get the same text again
<ESC> N	Get same text again in the other direction
<ESC> : m	Display the remaining memory

<ESC> : q	Quit the editor
<CTRL/C>	Quit the editor
<ESC> : q!	Quit the editor without update
<ESC> n #	Go to task number n.
<ESC>bb	Begin of block
<ESC>be	End of block
<ESC>bc	Copy block
<ESC>bd	Delete block

If <RETURN> is pressed on the end of a line (use: LINE FEED, RETURN), then a new line is inserted. If <RETURN> is pressed anywhere else on the line, the text will be replaced by a list of possible menu options.

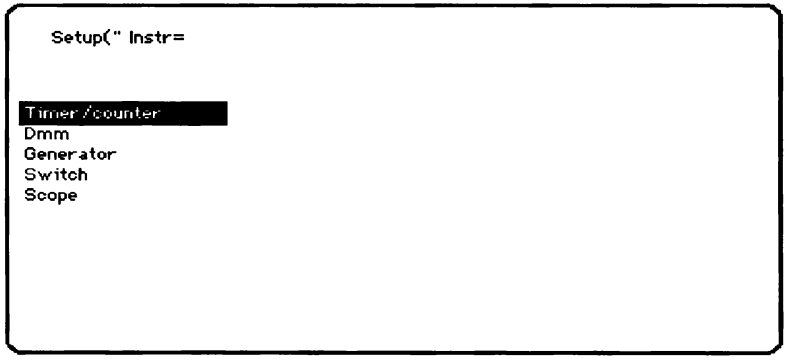
NOTE: At the beginning of each line 4 spaces are automatically inserted to allow for this function. By using the LINE FEED before inserting new text manually, automatic indentation is accomplished.

The figure below represents a typical display when the 'MENU EDIT' function is used.



The option list always reflects the actual entry level. This means that for any level, e.g. procedure, task or step, the appropriate menu options are displayed. Selections from the option list can be made simply by moving around in the list with the cursor control keys and pressing the return key. The field presently selected will be indicated by a reverse video field. After all selections have been made, the Esc key must be pressed. Each time the Esc key is pressed, the system returns to the next higher level and finally back to the file being edited.
Whenever <RETURN> is pressed while the cursor is not on an empty line, this line will be deleted and replaced by the new menu selections being made.

The following figure shows another menu which is typical to the 'MENU EDIT' function.



CHAPTER 9

TESTING A PROCEDURE

9 TESTING A PROCEDURE

After the source code of a procedure is created, it can be tested in several ways to finally have the test code.

9.1 OVERVIEW

This section describes how testing of Procedures can be done. Three modes of operation will be explained:

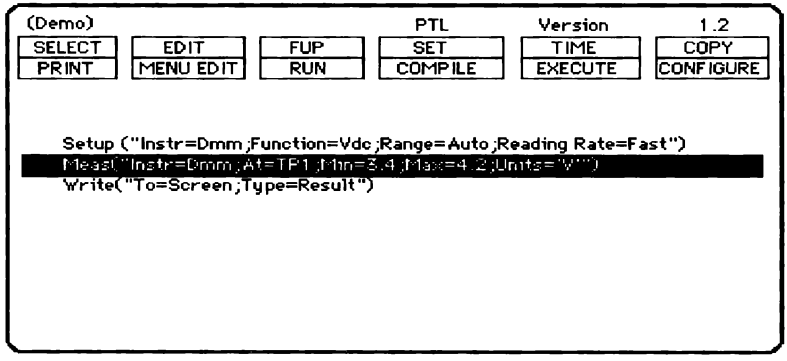
- Immediate mode
- Interpreted mode
- Compiled mode

The direct mode allows to type and execute commands directly, without storing them for later use.

To use the system to perform all desired functions like taking measurements and outputting stimulus, a suitable procedure must be created and executed. To distinguish between the Interpreted and Compiled Mode of Operation, two different functions are provided in the PTL Main Menu. The 'RUN' function has to be used whenever a procedure has to be executed in Interpreted Mode, while the 'EXECUTE' function must be used to execute procedures after they have been compiled with the 'COMPILE' function. The only noticeable difference for the user between those two functions is the speed of execution (when not using Basic statements).

9.2 IMMEDIATE MODE

The immediate mode is created specifically for testing special commands or command sequences. Direct mode can be started from the main menu by moving the cursor below the function keys. Below these keys up to 10 lines can be used for direct commands:



The entry of commands will be done the same as with the MENU-EDIT function described in the previous chapter. PTL commands can be typed in directly or a menu can be used to select the appropriate commands.
To execute these commands, the <ESC> key should be pressed when finished with entering the commands. At this moment the PTL commands entered will be executed immediately. All PTL statements can be used. All non-PTL commands will not be executed.

The MENU-EDIT/ESC sequence can be used to select from two functions for immediate mode:

- Wait for errors : Asks to enter RETURN after execution of the commands.

- Clear on entry : Clear all command lines when going to direct mode

Refer to the section "System Functions" for more details.

9.3 INTERPRETED MODE (RUN)

After a procedure has been created using an editor, the procedure can be interpreted by selecting RUN/RETURN.
At that moment the entire procedure will be run sequentially, not looking at non-PTL statements. This means that Basic statements will be skipped. At the end of the procedure, the main menu will automatically return.

Each new statement is interpreted (translated) at the moment that it is executed which means that execution will be slower than with compilation.

9.4 COMPILED MODE (EXECUTE)

To have a faster execution, a PTL compiler is included. This compiler translates a procedure once before executing her. Therefore for each new execution, there is no need for translation anymore, making execution much faster than interpretation.

Two function keys are available in the main menu of PTL: COMPILE and EXECUTE.

COMPILE translates the procedure and the embedded BASIC statements into executable code.
EXECUTE allows to run a compiled procedure.

The compilation process consists of three steps:

- Optimization: Creates direct instrument commands out of the PTL statements.

- Basic compilation: Translates all statements into machine readable code, using the extended Basic compiler.

- Linking libraries: Links standard library functions with the code, using the extended Linking Loader.

During each of these three steps error messages may appear.
During optimization the line numbers in the procedure file are being displayed. If a PTL error occurred, an appropriate error message will be put behind the line number.
During Basic compilation, Standard extended Basic compiler error messages may appear.
During Linking standard extended linking loader error messages may appear.

9.5 STORING A COMPILED PROCEDURE

In the PTL Main Menu, no special functions are provided to save procedures. The reason for this is that the PTL program saves procedures automatically any time edit functions have been used or the 'SELECT' function is used.

Note however that procedures will be stored on the system device SY0: which defaults to ED0 whenever you are running the PTL program. This implies that files which have to be used after power of the System Controller has cycled, must be saved on a non-volatile device like a floppy disk. This can be easily accomplished with the FCOPY command, as described in the chapter "System Functions".

CHAPTER 10

OPERATOR DISK GENERATION

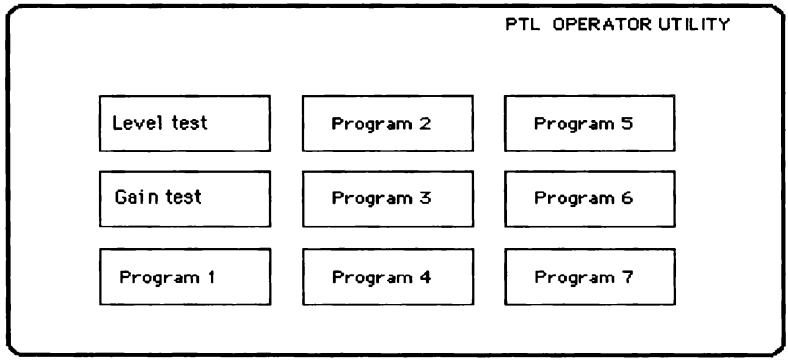
10 OPERATOR DISK GENERATION

Although the PTL program is very easy to use by a programmer, it is not intended to be used by unskilled operators. Therefore, a 'Master Operator Disk' is provided. This disk contains a so called 'operator interface program' which offers an operator to execute test procedures by touching the screen. With this approach, the system can be used by anyone, with very little chance of making mistakes.

Operator disks can be generated by making copies of the Master Operator Disk.

10.1 SELECTION MENU

The operator interface program can be loaded by pressing <RESTART>, after the operator disk has been inserted.
To allow an operator to make a selection from one of several programs (e.g.test procedures) by pressing the screen, a menu will be displayed that contains the names of nine executable programs. If more than nine programs are available, both a 'NEXT PAGE' and 'LAST PAGE' key are provided on the display as well. The figure below shows the screen lay-out.



A normal ASCII file is used for the display text inside the boxes. The programmer/procedure writer can create these text files as follows:

- o Make a copy of the original 'Master Operator Disk' to create a 'work disk'.
- o Run the normal EDIT program to create an ASCII file 'MASTER.DAT'.
- o Type in the desired display text and associated program name, separated by a command terminated by <RETURN>.
- o Repeat the third step for all filenames.
- o Save the file on the work disk.

After this, the file looks like:

```
Level test, LEVEL
Gain test, GAIN
  .

  .     .
Program 5, PROG5
Program 6, PROG6
Program 7, PROG7
```

Also system programs like FUP can be executed. If more than nine lines are typed, the software will automatically create a new 'page' on the display.
The text is displayed in the following order:

Text 1	Text 4	Text 7
Text 2	Text 5	Text 8
Text 3	Text 6	Text 9

Text will be centered as good as possible within the boxes but is limited to 20 characters.

After exiting a selected program, control will be automatically returned to the menu.

10.2 PROCEDURE TRANSFER

After the text for the selection menu has been created, also the appropriate procedures have to be transferred from the PTL System Disk to the Operator work disk. This can be done using the FCOPY program from the PTL Main Menu.

10.3 CONFIGURATION TRANSFER

In order to be able to execute the procedures, the associated configuration file must be available. The configuration can be transferred using the FCOPY program from the PTL main menu.
Please note that a procedure cannot be executed with a configuration file, other than the one with which it was created.

CHAPTER 11

ADVANCED PROGRAMMING

11 ADVANCED PROGRAMMING

In this chapter, programming hints are provided for advanced programmers. Specifically, the combination of PTL and Basic will be highlighted.

11.1 STANDARD PTL

A PTL program is a sequence of PTL statements, possibly enhanced with Basic statements. In the following examples, PTL statements are shown in lower case and Basic statements are shown in upper case.

The standard lay-out of a PTL procedure is as follows:

```
SUB Test(Task)
  On Error Subret
  Procedure("")

  End_proc
SUBEND
```

The SUB and SUBEND statements are actually the begin and end of a true Basic subroutine. The "On Error Subret" makes sure that possible errors are handled properly, with the standard handler on a higher level.
The procedure("") and End_proc are the beginning and the end of the PTL procedure.

A simple PTL procedure could look as follows:

```
Write("To=Printer;Type=Header")
Setup("Instr=Dmm;Function=Vdc;Range=20")
Meas("Instr=Dmm;At=Testpoint 1;Units='Vdc'")
Write("To=Printer;Type=Result")
```

This program sets up the printer and the DMM, takes one reading and prints the results.

For long procedures and program flow statements like "Loop on Fail", a procedure can be divided in several tasks:

```
SUB Test(Task)
On Error Subret
Procedure("Name='Unit 15 test'")

    Start-task("Name='Setup'")
       -----------
       -----------
     Endtask("")

    Start-task("Name='Power supply test'")
      ----------
      ----------
    Endtask("")

    Start-task("Name='Tune Amplifier'")
      ----------
      ----------
    Endtask("")

    Start-task("Name='Verify total unit'")
      ----------
      ----------
    Endtask("")

End_proc
SUBEND
```

Dividing a procedure into logical blocks (tasks) makes a procedure easier to understand and easier to edit.

Each task again can consist of many steps (statements).

E.g. to set the system up for measuring:

```
Write("To=Screen;Type=Header")
Setup("Instr=Power-supply;Voltage=12.6")
Setup("Instr=Dmm;Function=Vdc;Range=20")
```

or to perform a certain test:

```
Apply("Instr=Power-supply;To=Amp input")
Meas("Instr=Dmm;At=Testpoint 1;Min=5;Max=8.75;" &
    +"Units='Vdc'")
Write("To=Screen;Type=Result")
```

11.2 IMPORT PTL VARIABLES

PTL uses a number of global variables. These variables are made available to a programmer via the Basic "IMPORT" statement (See chapter 3). This allows the user to get results from PTL or to pass results to PTL.

E.g. The following example shows how to use this feature:

```
SUB Test(task)
On Error Subret
IMPORT RESULT(),NO_RESULT%,TYPE%

Procedure("Name='Amp test';Version='1.2'")
  Start_task("Name='Supply test'")
    Write("To=Screen;Type=Header")

    OUTPUT=0
    FOR L%=0 TO 11
      OUTPUT=OUTPUT+RESULT(L%)
    NEXT L%
    RESULT(0)=OUTPUT/12        ! Store average
    NO_RESULT%=1               ! 1 reading
    TYPE%=3                    ! of type 'real'
    Eval
    Write("To=Screen;Type=Result;" &
      +"Remark='Level test'")
  End_task("")
End_proc

SUBEND
```

Chapter 3 gives an overview of the imports possible.

11.3 CALL BY REFERENCE

Sometimes it is desired to use Basic variables instead of constant numbers within PTL statements. PTL supports such "Call by Reference"

In the next example e.g. the Basic variable VARMIN and VARMAX are calculated from the global variable array RESULT():

```
SUB Test(task)
On Error Subret
IMPORT Result()
Procedure("Name='Amp test';Version='1.2'")
  Start_task("Name='Supply test'")
    Write("To=Screen;Type=Header")
    Setup("Instr=Power-supply;Voltage=12.6")
    Setup("Instr=Dmm;function=Vdc;Range=20")
    Apply("Instr=Power-supply;To=Amp input")
    Meas("Instr=Dmm;At=Testpoint 1;Min=5.25;" &
     +"Max=8.75;Units=Vdc'")
    Write("To=Screen;Type=Result")

    VARMIN=RESULT(0)-0.25
    VARMAX=RESULT(0)+0.25
    Meas("Instr=Dmm;At=Testpoint 2;"          &
     +"Min=[VARMIN];Max=[VARMAX];units='Vdc'")

  End_task("")
End_proc
SUBEND
```

Explanation:
The PTL statement MEAS will take a measurement at Testpoint 1. VARMIN and VARMAX are calculated as the limits for the measurement at testpoint 2.

11.4 BASIC STATEMENTS

11.4.1 For .. Next loop

At any point in a procedure, Basic statements can be inserted between PTL statements.

```
SUB Test(task)
On Error Subret
IMPORT RESULT()
Procedure("Name='Amp test';Version='1.2'")
  Start_task("Name='Supply test'")
    Write("To=Screen;Type=Header")
    Setup("Instr=Power-supply;Voltage=12.6")
    Setup("Instr=Dmm;function=Vdc;Range=20")
    Apply("Instr=Power-supply;To=Amp input")

    Meas("Instr=Dmm;At=Testpoint 1;Readings=12")
    FOR L%=1 TO 11
      RESULT(0)=RESULT(0)+RESULT(L%)
    NEXT L%
    RESULT(0)=RESULT(0)/12
    Eval("Min=0.53")

    Write("To=Screen;Type=Result")
  End_task("")
End_proc
SUBEND
```

This program calculates the average of the 12 readings. The result is evaluated against a minimum value.

The next program uses a FOR-NEXT loop with a STEP clause to repeat a part of the procedure a number of times with another (higher) value for LEVEL.

```
SUB Test(task)
On Error Subret
Procedure("Name='Amp test';Version='1.2'")
  Start_task("Name='Supply test'")
    Write("To=Printer;Type=Header")
    Setup("Instr=Power-supply;Voltage=12.6")
    Setup("Instr=Dmm;function=Vdc;Range=20")
    Apply("Instr=Power-supply;To=Amp input")
    Meas("Instr=Dmm;At=Testpoint 1;Min=5.25;" &
     +"Max=8.75;Units=Vdc'")
    Write("To=Printer;Type=Result")

    FOR LEVEL=2.25 TO 14.50 STEP 0.25
      Meas("Instr=Dmm;At=Testpoint 2;" &
           +"min=[LEVEL0.5];max=[LEVEL+0.5]")
      Write("To=Printer;Type=Result")
    NEXT LEVEL

  End_task("")
End_proc
SUBEND
```

The test results could look like:

Minimum	Actual	Maximum	Units	P/F
1.75	2.10123	2.75	Vdc	PASS
2	2.28947	3	Vdc	PASS
2.25	2.51437	3.25	Vdc	PASS
2.5	2.67591	3.5	Vdc	PASS
2.75	2.91780	3.75	Vdc	PASS
3	3.74373	4	Vdc	PASS
3.25	3.49532	4.25	Vdc	PASS
3.5	4.00723	4.5	Vdc	PASS
3.75	4.22395	4.75	Vdc	PASS
4	4.36194	5	Vdc	PASS
4.25	4.94184	5.25	Vdc	PASS
4.5	5.02357	5.5	Vdc	PASS
4.75	5.48501	5.75	Vdc	PASS
5	5.29983	6	Vdc	PASS

11.4.2 Repeat .. Until loop

E.g. the next program uses a REPEAT .. UNTIL loop to wait for the voltage on test point 1 to rise above 12.45 Volt.

```
SUB Test(task)
On Error Subret
IMPORT RESULT()
Procedure("Name='Amp test';Version='1.2'")
  Start_task("Name='Supply test'")
    Write("To=Screen;Type=Header")
    Setup("Instr=Power-supply;Voltage=12.6")
    Setup("Instr=Dmm;function=Vdc;Range=20")
    Apply("Instr=Power-supply;To=Amp input")

    REPEAT
      Meas("Instr=Dmm;At=Testpoint)
    UNTIL RESULT(0)>12.45

    Write("To=Screen;Type=Result;")
  End_task("")
End_proc
SUBEND
```

In a PTL listing virtually all other Basic statements can be used like IF .. THEN .. ELSE and CASE statements. The [LEVEL+0.01] statement will do the calculations at RUN-TIME. (Any other calculation with one or more variables and/or constants is possible here).

11.5 "LOOP ON FAIL"

In a PTL procedure, any group of steps of tasks can be looped repeatedly until a certain condition is satisfied.
In Basic there are several loop structures that can be used for this feature:

11.5.1 Continous

This loop will only terminate if the criteria have met.

```
REPEAT
  ----------          ! Statements to
  ----------          ! calibrate
  ----------          ! an instrument
  Meas("Instr=Dmm;Min=3.48")   ! Test performance
UNTIL PASS%(0)
```

11.5.2 With loop count

This is a loop on fail for which the amount of loops is limited by the loop counter.

```
FOR COUNT%=1 TO 10
   -------------          ! Statements to
   -------------          ! calibrate
   -------------          ! an instrument
   Meas("Instr=Dmm;Min=3.48")      ! Test
   IF PASS%(0) THEN LEAVE
NEXT COUNT%
```

In this example it is tried to calibrate an instrument, until the verification passes the criteria. A maximum of 10 verifications may be done.

11.5.3 With fault indication

In most cases it is advisable to provide more than one way to terminate a loop. Conditions are:

1. A test object is OK thus the test procedure can continue with the next step.

2. A test object fails repeatedly (E.g. out of calibration or defective). The procedure must be stopped with an indication what (or where) the problem arose.

```
SUB Test(Task)
On Error Subret
Procedure("")
  -------------          ! Statements to
  -------------          ! setup system
  LOOP
    -------------          ! Statements to
    -------------          ! calibrate
    -------------          ! an instrument
    Meas("Instr=Dmm;Min=3.48")  ! Test if working
    IF PASS%(0)=0 THEN GOTO FATAL_ERROR
    Meas("Instr=Dmm;Min=3.54;Max=3.56") ! Test
    IF PASS%(0)=1 THEN LEAVE
  ENDLOOP
  Write("To=Screen;Type=Result")
End_proc
SUBRET

FATAL_ERROR:
  Write("To=Screen;Remark='Fatal Error in Power")
SUBEND
```

NOTE:

The **loop-endloop** is a basic loop structure without termination test nor loop counter. The Basic statement "IF PASS%(0) THEN LEAVE" is actually a "GOTO" to the next statement after the "END LOOP".
Any number or leaves (or goto's) can be used in one LOOP-ENDLOOP structure.

11.6 SUBROUTINES

In compiled Basic, subroutines can be made with local variables and with own names. PTL supports those subroutines in compiled mode. Three types can be distinguished:

- o REPORT subroutines
- o PTL subroutines
- o Basic subroutines

11.6.1 The REPORT subroutine

A MEAS statement includes (besides scanning and measuring) an EVAL statement. In a PTL procedure, often many MEAS, and therefore EVAL, statements can be found.
In most cases the result of such evaluation must be printed. In order to avoid many "WRITE, TO=SCREEN" or "WRITE, TO=PRINTER" statements, one REPORT routine is sufficient in compiled procedures. Normally, an empty report routine is supplied in the PTL library, which means that nothing will be printed after an evaluation (or measurement). This routine can be replaced with a user-supplied report routine with "WRITE" statements.

The following is an example of a simple report:

```
SUB Test(Task)
ON ERROR SUBRET
Procedure("Name='Example Report'")
  Write("To=Screen;Type=Header")
  Write("To=Printer;Type=Short header")
  Meas("Instr=Dmm;Min=3;Max=6;Remark='Test3'")
End_proc
SUBEND

SUB Report
ON ERROR SUBRET
  Write("To=Screen;Type=Result")
  Write("To=Printer;Type=Fail")
SUBEND
```

For each meas statement the result will always be put on the screen. The results will only be printed if the step failed.

NOTE: The Remark in the Meas (or Eval) statement will be used in all Write statements in the report. It stays valid until a new remark is given.

In certain cases, the Report subroutines must not be used in all Meas statements. For this purpose, a Report-Flag can be used as follows:

```
SUB Test(Task)
ON ERROR SUBRET
EXPORT Report_Flag%     ! Create a global flag
Report_Flag%=-1         ! Turn Report on
Procedure("Name='Example Report flag'")

  .......               ! With
  .......               ! Reporting

  Report_Flag%=0        ! Report off
  FOR L%= 1 TO 10       ! Loop

                        ! Without
                        ! Reporting

  NEXT L%
  Report_Flag%=-1       ! Report on

    .......             ! With
    .......             ! Reporting

End_Proc
SUBEND

SUB Report
ON ERROR SUBRET
IMPORT Report_Flag%

  IF Report_Flag% THEN
    Write("To=Screen;Type=Result")
```

```
    Write("To=Printer;Type=Fail")
  ENDIF
SUBEND
```

11.6.2 PTL Subroutines

A PTL procedure may be divided into subroutines e.g. for clarity or because the procedure becomes too big for the compiler. In this case subroutines can be created for each task. The main procedure may consist of calls to these subroutines only.

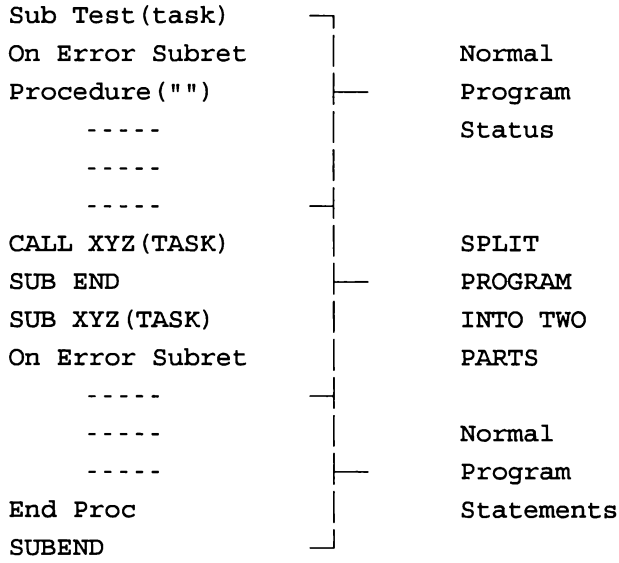
The following is a (partial) example of such procedure:

```
SUB Test(Task)
ON ERROR SUBRET
Procedure("Name='Example Sub-Tasks'")
  CALL Initialise
  CALL Test_Power
  CALL Impedance
  CALL Gain
End_Proc
SUBEND

SUB Impedance
  Start_Task("Name='Impedance'")
    Setup("Instr=Dmm;Function=Ohms")
    Meas("Instr=Dmm;At=Resist;Min=49.5;Max=50.5")
  End_Task
SUBEND
```

Although not shown here, also parameters can be passed to a subroutine. For such

example see the next paragraph. For details refer to the Fluke extended compiled Basic manual.
Due to the limitation of the Fluke Extended Basic Compiler, one single chunk of code cannot contain more than 500 calls to other modules. This translates into 300 PTL code lines.



A way around this problem is to split up the main program into two (or more) parts.

This is done with the 4 Basic lines in the middle of the PTL listing. These lines

a. Call the second part
b. End the first part
c. Start the second part
d. Handle the errors

NOTE:

The name XYZ can be any name, as long as it is not used elsewhere.

NOTE:

Splits are not allowed within a Basic LOOP or IF_THEN structure. It causes problems with GOTO's and GOSUB's.

PTL procedures, tasks and true subroutines are no problem.

11.6.3 Basic Subroutines

Except for only PTL statements, also Basic statements can be used in a subroutine. E.g. a Basic subroutine to calculate the average could look like:

```
SUB Test(task)
ON ERROR SUBRET
Procedure("Name='Amp test';Version='1.2'")
  Start_task("Name='Supply test'")
    Write("To=Screen;Type=Header")
    Setup("Instr=Power-supply;Voltage=12.6")
    Setup("Instr=Dmm;function=Vdc;Range=20")
    Apply("Instr=Power-supply;To=Amp input")

    Meas("Instr=Dmm;At=TP-1;Readings=12")
    LEVEL=0\ CALL AVERAGE(LEVEL)

    Meas("Instr=Dmm;At=TP-2;Min=[LEVEL-.01]")
    Write("To=Screen;Type=Result")
  End_task("")
End_proc
SUBEND
```

```
SUB AVERAGE(OUTPUT)      ! Sub header
ON ERROR SUBRET
IMPORT RESULT()          ! Conformal dimension
  OUTPUT=0               ! Preset output
  FOR L%=0 TO 11         ! Loop 12 times
    OUTPUT=OUTPUT+RESULT(L%)          ! Add value
  NEXT L%
  OUTPUT=OUTPUT/12       ! Calculate average
SUBEND                   ! End of routine
```

In order to communicate with the PTL statements, a subroutine can import global variables just like the main program.

Subroutines must always be stored at the end of a program listing and can never be embedded in another (subroutine) listing.

11.7 USER LIBRARY

Subroutines can also be compiled separately to be stored in a "user" library. In that case the subroutine is available for all procedures and does not have to be included in each listing to use it.

PTL supports a user library by providing appropriate command files to compile:

- o without a user library : C.CMD
- o to update the user library : CU.CMD
- o to link with user library : CL.CMD

These command files should be installed under COMPILE/<ESC> to use them.

The following paragraphs explain how to exploit user libraries.

11.7.1 Empty user library

Before a user library can be used, it must be created. An empty user library called USER.LBX is provided on the master operator disk which can be copied if required.

11.7.2 Update user library

After a subroutine has been created, using one of the editors, it must be compiled with the special command file CU.CMD which also updates the user library. Do such by changing the command file, specified under COMPILE/<ESC>

PTL normally adds a special reset routine to each procedure which resets all instruments connected. Since this routine is only needed once in each procedure, this option can be turned off for user library routines. Do such, by specifying NO under the COMPILE/<ESC> question "Reset Yes/No".

When the COMPILE function is used now, the subroutine is compiled (PTL and Basic) after which the user library is updated.

When merging a second module with the same name, the latest version of the two will be used in the library (not necessarily the newly merged module).

IMPORTANT

The function 'copy procedure' and 'copy configuration' in FCOPY does not copy the (changed) user library. This has to be done using fup at the end of a (useful) session. Use:

```
FUP
MF0:=EDO.USER.LBX
/X
```

NOTE:

Every subroutine can use other subroutines. If these sub-subroutines are included in the same module, they are also accessible as individual subroutines from procedures. To make sure that the names of these routines do not collide with other subroutine names.

NOTE:

A subroutine can contain Basic and PTL statements. However, if PTL statements are included, the pre-processor uses the current configuration information to produce the compact Basic equivalents. So, if the configuration is changed (e.g. new scanner instructions for "at name"), the subroutines, using this information, needs to be re-compiled.

11.7.3 Include a user library

A special command file CL.CMD is provided to include the user library. The difference with the normal command file is, that the linker starts looking into the user

library, to try to resolve subroutines, before looking in the PTL libraries.

There are a number of simple rules concerning the action of the linker:

1. Since the user library is called first, subroutines will be taken from this library, before looking in the PTL libraries.

2. If there are routines in the library that are not called, the linker will include only the requested routines.

3. If routines are called which are not in the user and PTL libraries, the linker cannot and will not produce a program. The "MAP" file (Report file from the linker) entails what went wrong.

4. If one or more routines export variables whose names are already used elsewhere, a "MULTIPLY DEFINED" error will occur. Again the map file and the Fluke linker documentation will tell what went wrong.

5. If one or more of the user subroutines use the same name as one of the internal routines of the PTL system, a problem could occur. In this case the user library routine will always be used instead of the PTL routines.

Using the library CL continuously does not add in program size. However it will take some extra time for a program to be compiled, since the linker has to inspect the user library.

11.8 USE OF ERROR HANDLERS

The "On Error Subret" statement passes Basic errors back to PTL. Correct handling of errors in PTL depends on this statement, so never omit this statement in any routine.

In certain cases it is needed to include an own error trap in your procedure. You can use an "On Error" trap to a user error handler. To turn this error trap off, use "On Error Subret".

```
SUB Example
ON ERROR SUBRET
  .........              ! Normal PTL or
  .........              ! Basic lines
  ON ERROR GOTO Trap     ! user error handler
  .........              ! to protect
  .........              ! these lines
  ON ERROR SUBRET        ! PTL error handler
  .........              ! for these
  .........              ! lines
SUBRET                   ! End of subroutine

Trap:
  .........              ! Error
  .........              ! handler
  RESUME
SUBEND                   ! End of subroutine
```

CHAPTER 12

SYSTEM DETAILS

12 SYSTEM DETAILS

This section provides detailed information about the internal functioning of PTL.

12.1 SYSTEM GENERATION

During system generation the following steps are being performed:
- A driver library DRIVER.LBX is being created from the selected drivers.
- A subroutine with a driver table is created
- The PTL program is linked together with the libraries DRIVER, PTL and SYSTEM
- A library PTL.LBX is created as a subset from the main PTL library (Run-time only).
- All important Files are copied from the generator disk to the system disk

12.2 STORE MENU CONFIGURATION

The menu configuration is stored in the file PTL.DAT. After making changes in the menu, PTL.DAT will only be changed on E-disk. therefore copy this file to floppy to make it permanent.

The same is true for the menu in system configuration, only this time the information is stored in SYSGEN.DAT.

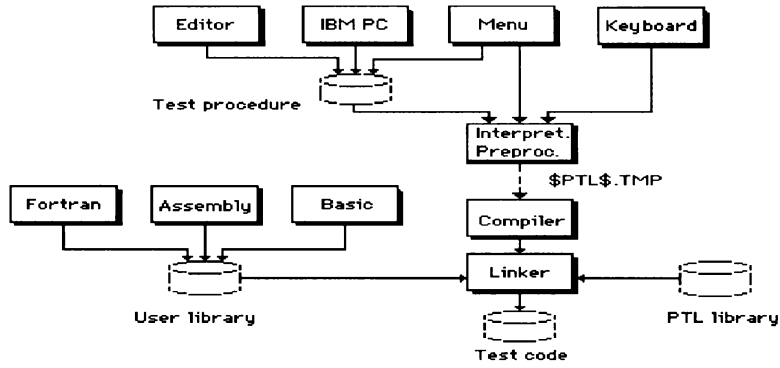
12.3 STEPS REQUIRED FOR COMPILATION

The following steps are required to create an executable procedure:

- o Create a procedure
- o Optimize PTL
- o Compile Basic

o Link libraries

. The following diagram shows the sequence:



12.3.1 Creating a procedure

Procedures which have to be compiled can be created and modified using the PTL MENU EDIT function, the standard System Editor program (EDIT.FD2), or any text processing system that generates ASCII files.

The original (source) procedure can be written in any combination of legal PTL and Extended Compiled BASIC commands and functions. Refer to the description of PTL language elements or the Fluke 1722A Extended Compiled BASIC manual for all information on those commands and functions.

12.3.2 Optimize PTL

Although not strictly required, PTL normally optimises the PTL commands. This means that all English type commands are replaced by the equivalent "IEEE Codes". The reason for this is that the code becomes more compact (a smaller program is created) and the speed is significantly higher.

The standard PTL program performs this function automatically when the COMPILE function is used. The result of the optimization is the file PTL.TMP.

12.3.3 Compile Basic

The standard Fluke Extended Basic compiler is used to compile the basic program. If the normal command file C.CMD is used in the COMPILE function, PTL.TMP will be compiled with the options: /I/E/NL.
The result is <procedure name>.OBX

12.3.4 Link Libraries

The standard Fluke Extended Linking Loader is used to link the object code together with:

- o The main program
- o The PTL.LBX library
- o The DRIVER.LBX library
- o The SYSTEM.LIB library

The result is the file <Procedure name>.FD2.

The main program has the following structure:

```
    Initialize program

While task<=0
  call test(task)
Endwhile

Close files
```

12.4 CREATE A USER LIBRARY

The user may create his own library of subroutines. These subroutines may contain e.g.:

Procedure subroutines:
A procedure may be split up in small subroutines, consisting of one or more steps. Each step can then be written as a basic subroutine, using PTL, and can be stored in a user library. The main procedure could then consist of calls to these subroutines.

Special functions:
Special functions can be written in PTL, Basic, Fortran and Assembler. These functions could include: Special evaluation of results, graphic routines etc. These functions can be stored in a user library to be included automatically when needed.

PTL supports special subroutines and user libraries in two ways, both to be found under the menu function COMPILE/<ESC>.

12.4.1 Compilation

The standard command file for compilation is c.cmd. To support user libraries, two other command files are included.

C Compile a standard test procedure. Usage:
C <source file> <error file>

CU Compile a user subroutine and store it in a user library USER.LBX. This command file assumes the presence of the Extended Library Manager on floppy usage:
CU <source file> <error file>

CL Compile a test procedure and link also a user library USER.LBX. Usage:
CL <source file> <error file>

12.4.2 Main/subroutines

Normally, (when selected in COMPILE/<ESC>) PTL adds a RESET_SYSTEM subroutine to the standard test subroutine, to reset all instruments being used in the test procedure.

This is not desired when subroutines have to be written. For this reason this selection can be turned off in COMPILE/<ESC>.

12.5 VARIABLES IN PTL

In some cases it might be desirable to use variables in the PTL statements. PTL supports such "Call by reference", as a replacement for constants only. This can be done by entering the variable name between square brackets instead of such constants.

Example:

```
For Volts= 5 to 15 step 3
  SETUP ("Instr=Power;Voltage=[Volts];Limit=1")
Next Volts
```

All valid Basic variables may be used.

Note: Interpretation for instrument capabilities will be done during compilation and excludes variables. Therefore extreme care must be taken when using this feature.

Note: Due to the fact that during PTL interpretation, Basic is skipped, these variables can not be interpreted.

12.6 FILES REQUIRED FOR PTL

In addition to your PTL program, the following programs are on the system disk:

Note: The files listed below are the minimum, needed to compile and run a PTL/BASIC procedure. Other files may be needed for more complex programs.

12.6.1 Program Files

The PTL program consists of three files:

- PTL.FD2 The actual PTL program, including all instrument drivers to allow direct and immediate mode.

- PTL.DAT Contains the information of the function keys: The names and the ESC

settings. Each time these settings are changed, this file must be put on floppy to make the changes permanent.

- PTL.BIN Contains the configuration information for: devices, instruments and connections. Also contains the screen lay-outs.

- SCREEN.DAT Contains all screens needed for the system.

12.6.2 Extended Basic Compiler

The Standard Fluke Extended Basic Compiler XBC.FD2 has been used for PTL.

12.6.3 Extended Linking Utility

The standard Fluke Extended Linking Utility XLL.FD2 has been used.

12.6.4 Extended Basic Runtime

The standard Extended Basic Runtime System BSXRUN.FD2 has been used

12.6.5 Libraries

Three libraries must be linked together (in this order) with the test procedure:

PTL.LBX	:	With all PTL statements
DRIVER.LBX		All instrument drivers selected
SYSTEM.LIB		Standard system functions for the 1722A hardware.

A USER library may be added to include special user functions or test steps. This library must be linked with a procedure before any of the previous ones.

CHAPTER 13

APPENDICES

13 APPENDICES

A. Example Configuration Listing

The sample listing below represents a typical listing of a system configuration. To clarify the terms used in the listing the following explanation is given.
The listing is divided into three groups : devices, instruments and connections.

DEVICES

The group 'devices' shows the relation between device names and physical devices (hardware). This means that a name like 'Screen' will be translated to 'KB0:' any time it is used.

INSTRUMENTS

This group shows the relation between the identifiers which can be used in programs, the names of the instruments, their IEEE address and the type of instrument.
As can be seen in the listing, some instruments can be configured for any combination of input, output and scan functions.

CONNECTIONS

The connections for each instrument are listed in this group, showing the following items:

- Instrument identifier. This column contains the identifiers, defined during 'Configure Instruments'
- Type. The type of connection i.e. input,output or scan, is listed in this column.
- Switch matrix. This column lists the identifiers of instruments defined as 'SCAN'.

- Connection name. This column contains the names which can be used in programs to identify a particular connection.
- Code. The code field lists all connections to be made. These codes consist of the scanner number, separated from the number of the relay (or electronic switch) by a '-' (minus, NOT an underscore) character. If more connections have to be defined on the same line, they must be separated by a '/' (slash) character. The number of connections that can be defined on one line is limited to 3.
- Switch delay. This column contains the delay time in milliseconds the system will wait (after the connection has been made) before it continues with the next program step.

```
--- Polar Test Language. Version 1.2 - Configuration -
--- Customer: Polar --- Printed at 08-Jun-87 14:04 ---
-----------------------------------------------------

-----------Device names---------------

Nr    Name               Device
--    ----               ------
0     Screen             KB0:
1     Printer            KB1:
2     Disk               MF0:


-----------Instrument names-----------

Nr  Instrument Name          Address Program name  Type
--  ---------------          ------- ------------  ----
0  HP5335A Counter      1. 1    Counter       in
1  HP3326A 2-Chan synth.    2. 1    Synthesizer   out
2  Fluke 8842A Multimeter   3       Dmm           in
3  HP3488A Sw.Control unit  6       Switch       c+i/o
4  Kepco TLD 488-16/4A PS   8. 0    Supply        out
5  Thorn EMI SHR PS         7       Test voltage  out
6  Delta PSC 625R PS        4. 1    Ref level     out
```

```
                    ------ Page 1 -----

----------------------Connections----------------------
Instruments    Type    Matrix    Name     Code    Delay -
------------  ----    --------  ------   -----   -----DMM
          Input:  SWITCH    AMP IN   1-1      33
                  SWITCH    AMP REF  1-2      44
                  SWITCH    AMP OUT  1-3      55
                  SWITCH    AUX REF  1-4      55

SUPPLY          Output: SWITCH    +5       2-3      44

TEST VOLTAGE    Output: SWITCH    TEST 1   4-1      33
                  SWITCH    TEST 2   4-2      35
                  SWITCH    TEST 3   4-3      35

REF LEVEL       Output: SWITCH    PLUS REF 5-1      55
                  SWITCH    MIN REF  5-2      55
                  SWITCH    AUX REF  5-3      55

                    ------ Page 2 -----
```

B. Example Driver listing

The sample listing below represents a typical listing of instrument drivers in the form of tables. To clarify the terms used in the listing the following explanation is given.

The driver software contains several so called levels. These levels determine the menu options being displayed whenever a selection from the menu has to be made. Assume for this explanation that the function 'SETUP' has been chosen from the first menu to prepare a Fluke 8840A multimeter to do some measurements. As can be seen in the listing, level

1 is indicated as 'FUNCTION'. The parameters,

i.e. the options that can be used with 'FUNCTION', are VDC,VAC,2 WIRE OHM... etc. The (IEEE) commands to be sent to the 8840A are F1, F2, F3...etc. respectively.

The 'New level' column in the listing indicates the new menu options that will be displayed after one of the above parameters have been chosen. Assume FUNCTION=VDC has been selected. The new menu (level 2 , last column) contains the options AUTO,0.2,2,20,200,1000 as parameters to the RANGE function. After a selection from this menu has been made, return will be controlled to level 1, as indicated in the column 'New level' on each line of the RANGE function, displaying the first menu again. This approach is used for any of the other menu options.

```
--- Polar Test Language. Version 1.2 ----- Drivers ---
--- Customer: Polar --- Printed at 08-Jun-87 14:04 ---
-----------------------------------------------------

----------< Kepco tld 488-16/4A power supply >---------
Name   Level Function   Parameter         IEEE        New
level
----   ----- --------   ------------ -------  ---------
Setup    1   Volt=      -1000..1000  SET VOLT#    2
         2   Cur limit= 0..1000      SET CURL#
         1   Current=   0..1000      SET CURR#    2
         2   Volt limit -1000..1000  SET CURL#
         1   Max volt=  -1000..1000  SRX VOLT#
         1   Min volt=  -1000..1000  SRN VOLT#
         1   Max curr=  0..100       SRX CURR#
         2   Min curr=  0..100       SRN CURR#
         1   Max v lim= 0..100       SRX VLTL#
         1   Min v lim= 0..100       SRN VLTL#
         1   Min c lim= 0..100       SRX CURL#
         1   Min c lim= 0..100       SRN CURL#
```

C. Example Procedure

```
--- Polar Test Language. Version 1.2 ----- Example ---
--- Customer: Polar --- Printed at 08-Jun-87 14:04 ---
-----------------------------------------------------

SUB Test(task)
On Error Subret
Procedure("")

  Start_task("Name='SETUP SYSTEM'")
    Write("To=Screen;Type=Header")
    Write("To=Printer;Type=Header")
    Setup("Instr=Supply;Volt=5;Current limit=4")
    Apply("Instr=Supply;To=+5")
  End_task("")

Start_task("Name='Test Power Supply'")
  Setup("Instr=Dmm;Function=Vdc;Range=20;" &
  +"Reading rate+Fast")
  Setup("Instr=Test voltage;To=Test 2")
  Meas("Instr=Dmm;At=Amp ref;Min=8.2;Max=8.4;" &
  Units='Vdc';Remark='Amplifier ref voltage'")
End_task("")

Start_task("Name='Test Amplifier'")
  Setup("Instr=Dmm;Function=Vac;Range=20;" &
  +"Reading rate=Medium")
  Meas("Instr=Dmm;At=Amp in;Min=12.4;Max=12.6;" &
```

```
  +"Units='Vac';Remark='Amplifier local input'")

                ---------Page 1---------



--- Polar Test Language. Version 1.2 ----- Example ---
--- Customer: Polar --- Printed at 08-Jun-87 14:04 ---
-----------------------------------------------------

  Meas("Instr=Dmm;At=Amp in;Min=6.05;Max=6.45;" &
  +"Units='Vac';Remark='Amplifier Ref voltage'")
  Meas("Instr=Dmm;At=Amp out;Max=18.5;" &
  +"Units='Vac';Remark='Output voltage'")
End_task("")

Write("To=Screen;Type=Pass/fail test")
End_proc
SUBEND

SUB REPORT()
On Error Subret
    Write("To=Screen;Type=Result")
    Write("To=Printer;Type=Result")
SUBEND

              --------- Page 2 ---------
```

D. Example Result

Minimum	Actual	Maximum	Units	P/F
1.75	2.10123	2.75	Vdc	PASS
2	2.28947	3	Vdc	PASS
2.25	2.51437	3.25	Vdc	PASS
2.5	2.67591	3.5	Vdc	PASS
2.75	2.91780	3.75	Vdc	PASS
3	3.74373	4	Vdc	PASS
3.25	3.49532	4.25	Vdc	PASS
3.5	4.00723	4.5	Vdc	PASS
3.75	4.22395	4.75	Vdc	PASS
4	4.36194	5	Vdc	PASS
4.25	4.94184	5.25	Vdc	PASS
4.5	5.02357	5.5	Vdc	PASS
4.75	5.48501	5.75	Vdc	PASS
5	5.29983	6	Vdc	PASS

POLAR SYSTEMS

OPTION PTL-DRV

PTL Driver Development Package

1 CONTENTS

2 CREATING INSTRUMENT DRIVERS

This manual describes how to create instrument drivers for the Polar Test Language PTL Version 1.3.

This information is supplied as option : PTL/DRV PTL Driver Development Package

2.1 INTRODUCTION

A special "Driver Engineering Disk" is provided to create or change instrument drivers. To start, insert this disk in the controller and press "RESTART". All relevant files will be copied to the E-disk. (Please note that all information on the E-disk will be destroyed).

To edit a driver source, use the command "EDIT NAME.BAS". "NAME" stands for the instrument name and is limited to 5 characters. PTL uses the following convention:

- First letter: Instrument manufacturer
  E.g. F for Fluke, H for HP
- 4 characters: Instrument identification
  E.g. 8840 for a Fluke 8840A
  3325 for a HP 3325A

For preparing (compiling and linking) a driver, use the command "MAKE NAME". The command file "MAKE" will take care that:

- the program is compiled by the basic compiler
- a small library module is made with the name "NAME.DRV".

If Basic errors are encountered in the Basic compiler, the process stops and no valid ".DRV" module is produced. If there is a ".DRV" module after a Basic error message, this must be the result of a previous compilation, and is not the new module.

The newly generated module should now be copied to an "INSTRUMENT DRIVER DISK" which can be used for generating a PTL system.

NOTE:

The system generator program will select all the compiled drivers from a driver disk, by looking for ".DRV" modules and checking these for valid driver codes. Only correct files are considered to be candidates.
Other files on the same disk will not confuse the selection mechanism, so the engineering disk could be used as an instrument driver disk.

2.2 STRUCTURE DRIVER

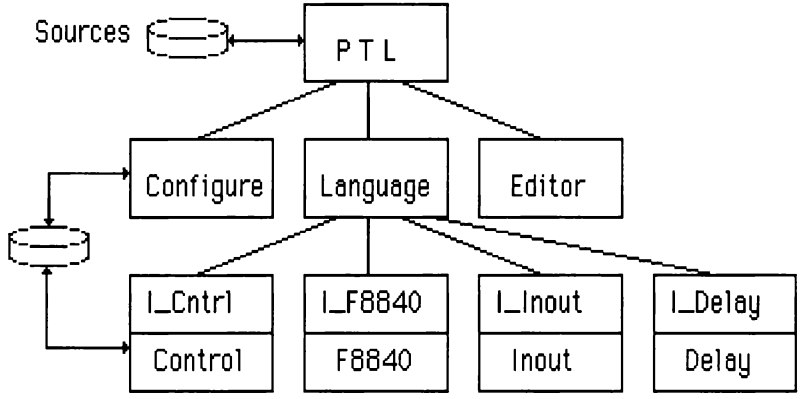
An instrument driver is a module with two true Basic subroutines: an interpreter and a run-time subroutine.

```
SUB I_NAME (passing area)
   .............
   .............
SUBEND

SUB NAME (passing area)
   .............
   .............
SUBEND
```

The interpreter routine is used in the PTL menu editor and in the interpreter and compiler. It contains all capabilities of the instrument plus the translation into "ieee" codes.
The run-time part is the actual driver for the instrument. This is the only part which is included in the (final) compiled procedures.

The following diagram gives an indication of the PTL structure:



The following is a series of listings that will be built up to a real driver for a DMM from Fluke, the 8840A.

```
SUB I_F8840 (Action%, Address, Block%, Command$)
ON ERROR SUBRET
  Pcode$="XXXXXXXXXX CODE XXXXXXXXXX"
  IeeeS=""

  Interp("F8840", Address, Block%, Pcode$,      &
                           Command$, Ieee$)
  F8840 (Address, Block%, Ieee$)
SUBEND

SUB f8840 (Address, Block%, Ieee$)
  PRINT @Address, Ieee$     ! Send ieee string
SUBEND
```

This (simplified) listing, shows that the task for the interpreter part, is to fill a string Pcode$ with driver information and to call the routine "interpreter".

This interpreter routine will translate the command from a user, given in the variable Command$, with the use of the driver code in Pcode$. It produces the driver runtime command Ieee$.

Ieee$ is passed to the run-time part of the driver where it is send to the instrument.

The following paragraphs describe the two parts of the driver in detail.

2.3 INTERPRETER PART

2.3.1 Calling the driver

The driver must be called as follows:

```
I_F8840 (Action%, Address, Block%, Command$)
```

The following parameters are needed in the passing area of the interpreter part:

Action% = The type of action to be performed:
- 10 = Reset
- 11 = Setup
- 12 = Measure

Address = The address of the device connected:

0..31 = IEEE address

The fractional part is the secondary address (if allowed). E.g. 12.05 means address 12 secondary 5

Block% = pointer to reserved space for local variables

Command$ = Command to be analyzed

In the following paragraphs, the driver will be detailed out.

2.3.2 Detailed Example

The interpreter of the driver for the Fluke 8840A could look like:

```
SUB I_F8840 (Action%, Address, Block%, Command$)
ON ERROR SUBRET

SELECT Action%
  CASE -1        ! Identify
    Command$="1NAME)Fluke 8840A Multimeter ~" &
           +"D10,T1,L1,H31,B0,IDMM,V1.0{}|"
    SUBRET
  CASE 10        ! Reset
    Pcode$=""
    Ieee$="*"
  CASE 11        ! Setup
    Pcode$="1FUNCTION}VDC~F1,{}OHM~F3,{}|"
    Ieee$="@a,"
ENDSELECT

Interp("F8840", Address, Block%, Pcode$,    &
                         Command$, Ieee$)
F8840 (Address, Block%, Ieee$)
SUBEND
```

2.3.3 Actions

The action is used to define what action is required from the driver. The following actions are available:

```
IDENTIFY        = -1

PROCEDURE       = 01
END_PROC        = 02
START_TASK      = 03
END_TASK        = 04

EVAL            = 09
RESET           = 10
SETUP           = 11
MEAS            = 12
APPLY           = 13
DISAPPLY        = 15
SWITCH          = 16
DISCONNECT      = 17
to              = 18
at              = 19

WRITE           = 20
ENTER           = 21

DELAY           = 30
```

Identify is needed to identify the instrument to the system generator and the configurator.

Only action codes 10..19 must be resolved in the driver. Depending on the instrument used, only a few actions could be used. E.g.:

- Measurement devices will generally use Reset, Setup and Meas.
- Generators will use Reset, Setup, Apply and Disapply.
- Scanners and digital output instruments will use Reset, Switch and Disconnect (as direct commands from a user) and will also use "to" and "at" for scanning other instruments.

2.3.4 Identification code

During system generation and system configuration the driver is called with action code = -1 to get its identification code. In this identification code, information about the type of driver is specified as follows:

```
"1NAME}Fluke 8840A Multimeter "     &
          + "~D10,T1,L1,H31,B0,IDMM,V1.0{}|"
```

The length of the information code MUST BE ODD ! If the length is even, than add a space somewhere in the code (e.g. behind Multimeter) to make the string odd.

The identification code is embedded in a rigid format as follows:

```
1NAME} <identifier> ~ <code> {}|
```

to be compatible with all other select codes used. This format is needed to be able to be recognized as an instrument driver by the system generator. The select codes will be explained in one of the next paragraphs.

The <identifier> contains the name and type of instrument. In the previous example this was:

```
Fluke 8840A Multimeter
```

The <code> contains several parameters. For the previous examples these where:

```
D10,T1,L1,H31,B0,IDMM,V1
```

with:

D= Default ieee address or port number
  0 .. 31 = ieee address
  70 .. 72 = screen - printer - disk

T= Type of instrument
  0 = none
  1 = input
  2 = output
  3 = input and output
  4 = scanner
  5 = scanner and input
  6 = scanner and output
  7 = scanner and input and output
L= Ieee address lower boundary
H= Ieee address upper boundary

l= Secondary address lower boundary
h= Secondary address upper boundary

B= Number of user blocks for local variables

+n = n block in the dim file
0 = no space reserved
-n = n blocks in main memory

I= Default identifier
V= Version of driver

2.3.5 Driver code

The driver code describes all capabilities of the instrument and its translation into the "ieee" code. This is the coded version of the information which is printed when "PRINT DRIVERS" is selected in the PTL program.

In the previous example we had as driver code:

```
"1FUNCTION}VDC~F1,{}OHMS~F3,{}|
```

This should be understood as:
On level 1 we have as functions
- VDC which translates to ieee code "F1,"
- OHMS which translates to ieee code "F3,"

A driver code is one of the many select codes used in PTL. The format and capabilities of the select codes is explained in the following paragraph.

2.3.6 Select code

A select code is a generalized format for all selections to be made at a certain part in PTL. It gives a tree of choices.
The select code has the following format:

```
code$="1SELECT}CHOICE 1~F1,{2}CHOICE 2~F2,{2}|"
```

with:

1	= Level in the tree
SELECT	= Start of command to be selected
}	= Separator
CHOICE 1	= Choice for a possible command
~	= Separator
F1	= Code to translate to
{2}	= Level for next choices
|	= End of code string

E.g. the command "SELECT=CHOICE 1" will translate into code "F1,".

All selections with level 1 are always possible: In menu edit, they will always be displayed. If a next level is specified, then ALSO selections on this new level are valid. If no next level is specified, the level will stay unchanged.

There are 3 types of data possible:

1. Select from several possible words given by the code
2. Select a number
3. Select a string

The general format for the code is:

<n><command>}
choice>~<ieee>{<m>}..<choice>~<ieee>{<m>}|

With:

<n>	=	Level for command
<command>	=	A possible command at level n (upper case only)
<choice>	=	Possible choices for command: - Text : must be exact match - Min..Max : value between 2 limits - $: can be any string
<ieee>	=	Equivalent 'ieee' format for command - Text : ieee will be text - ..$.. : String will replace $ - ..#.. : Value will replace #
<m>	=	Second level for new command (first is always 1). m may be deleted if no new value is required

Example 1: Numeric entry

```
code$="1VALUE}1..999~N#,{2}|"
```

1	=	Level for command
VALUE	=	Name of variable to enter
}	=	Separator
1	=	Lower boundary of value
..	=	Separator
999	=	Higher boundary of value

~	=	Separator
N#,	=	Code for driver, where # will be replaced with number
{2}	=	Level for next choices
|	=	End of code string

With this code a command like "VALUE=234;" will be translated into "ieee" code "N234,"

Also "Call by reference" with Basic variables is possible. A command like "VALUE=[LEVEL]" will be translated into "N"+NUM$(LEVEL)+"," (compiler only)

Example 2: String input

```
code$="1TEXT}$~T'$',{2}|"
```

1	=	Level for access
TEXT	=	Name of function to select first
}	=	Separator
$	=	Code for text entry
~	=	Separator
T'$',	=	Code for driver, where $ will be replaced by entered text
{2}	=	Level for next choices
|	=	End of code string

With this code a command like "TEXT='Hello world';" will be translated into 'ieee' code:
"T'Hello world',"

Also "Call by reference" with Basic variables is possible. A command like "TEXT=[GREETING$]" will be translated into "T'"+GREETING$+"'," (compiler only)

Unnecessary to say that in this case, the compiler cannot test the value of the passed variable. So the user must either test himself with basic statements or write an instrument driver that can handle invalid information.

2.3.7 Example code

To demonstrate selection codes, we now include the actual interpreter code for the Fluke 8840A Digital Multimeter:

```
pcode$= "1FUNCTION}VDC~F1,{2}2 WIRE OHM~F3,"  &
      + "{4}4 WIRE OHM~F4,{4} MA DC~F5,{5}"  &
      + "MA AC~F6,{5}|"                      &
      + "2RANGE}AUTO~RO,{1}0.2~R1,{1}2~R2,"  &
      + "{1}20~R3,{1}200~R4,{1}"             &
      + "1000~R5,{1}AUTO OFF~R7,{1}|"        &
      + "3RANGE}AUTO~R0,{1}0.2~R1,{1}2~R2,"  &
      + "{1}20~R3,{1}200~R4,{1}"             &
      + "700~R5,{1}AUTO OFF~R7,{1}|"         &
      + "4RANGE}AUTO~R0,{1}200~R1,{1}2K~R2"  &
      + "{1}20K~R3,{1}200K~R4,{1}"           &
      + "2M~R5,{1}20M~R6,{1}AUTO OFF~R7"     &
      + "{1}|"                               &
      + "5RANGE}AUTO~R0,{1}2000~R5,{1}AUTO"  &
      + "OFF~R7,{1}|"                        &
      + "6V-&RANGE}AUTO~R0,{}0.2~R1,{}2~R2," &
      + "{}20~R3,{}200~R4,{}"                &
      + "700~R5,{}AUTO OFF~R7,{}|"           &
      + "6IMPEDANCE}1..999~@c#,{}|"          &
      + "1READING RATE}SLOW~S0,{}MEDIUM~S1"  &
      + "{}FAST~S2,{}|"                      &
      + "1TRIGGER MODE}INTERNAL~TO,{}"       &
      + "EXTERNAL~T1,{}|"                    &
      + "1OFFSET}OFF~B0,{}ON~B1,{}|"         &
      + "1DISPLAY}NORMAL~DO,{}BLANK~D1,{}|"  &
```

Note that the {level} is used to select which range is used for a certain function.

2.4 RUN-TIME PART

2.4.1 Calling the driver

The second part of an instrument driver is used during run-time. It is the part which adapts PTL to the instrument, whether it connected to the IEEE bus or to any other interface. It is therefore not possible to give a fixed solution for all drivers.

The run-time subroutine is the only part which is linked to a compiled procedure. This part must therefore be optimised for speed.

The structure of such driver is as follows:

```
SUB F8840 (Address, Block%, Ieee$)
  IMPORT Type%, No_result%, Result()
  IF Ieee$="*" THEN          ! if it is a reset command
    clear @address           ! reset instrument
  ELSE
    IF LEFT(Ieee$,2)="@a" THEN! command to measure
      TRIG @Address           ! Ask for a measurement
      INPUT @Address, Result(0)! Get measurement
      Type%=3                  ! Real result
      No_result%=1            ! and tell it's only one
    ELSE                       ! setup the instrument
      PRINT @Address, Ieee$    ! Send ieee string
    ENDIF                      ! end in/output test
  ENDIF                        ! end reset test
SUBEND
```

The driver must be called as follows:

```
F8840 (Address, Block%, Ieee$)
```

The following parameters are needed in the runtime part:

Input parameters:
- Address = The address of the device connected
- Block% = pointer to local variables
- Ieee$ = Command to be executed

Imports :
- No_result$ - Number of results
- Type% - Type of result
  - 0 = No result
  - 1 = Bit
  - 2 = Integer
  - 3 = Real
  - 4 = String
- Result(0) - Result (real or integer) also be used for array 0..14
- Result$ - Result (string)
- Next_time(add)- array with wait times (indexed with ieee address)

The actual driver must handle all (exceptional) situations. It must therefore be able to handle an instrument which is off-line, defective, over-ranged or non-existing. In certain case this could result in time-outs during bus transfer, which could be trapped with on-error statements.

2.4.2 Detailed example

The following example is the actual Fluke 8840A driver:

```
WT%=0
IMPORT Type%, No_result%, Result(), Next_time()
ON ERROR GOTO WRONG
TIMEOUT 3000
IF Ieee$="*" THEN      ! Reset command
  Next_time(Address)=Time+3000    ! 3 sec wait time
  ON ERROR GOTO No_instrument        ! trap no
instrument
  CLEAR @Address       ! reset instrument
 End_init:
  ON ERROR GOTO Wrong
ELSE                   ! Not a reset command
  IF Next_time(Address)>Time+5000 THEN
    Next_time(Address)=Next_time(Address)-86400000
  ENDIF
  WHILE Next_time(Address)>Time  ! wait for time
  ENDWHILE

  IF LEFT(Ieee$,2)="@a" THEN     ! do we measure?
    Type%=0            ! Not a valid result
    TRIG @Address      ! Ask a measurement
    INPUT @Address, Result(0)     ! Get measurement
    Type%=3            ! Real result
    No_result%=1       ! One result
  ELSE
```

```
    PRINT @Address, Ieee$          ! send ieee string
    Next_time(Address)=Time+1000 ! 1 sec wait
 ENDIF                  ! end meas/setup
ENDIF                   ! end reset
SUBRET

! --- Error handler ---

No_instrument:
  RESUME End_init

Wrong:
  RESUME Back

Back:
  Runtime_error (Ieee$, Err)      ! Normal error

SUBEND
```

Note that wait times are handled by the Next_time() variable. During reset, wait_time() is set to a moment of 3 seconds in future. If this driver is called again for a setup or a measurement, and this moment has not been reached yet, the drive will wait. This enables the whole system to perform other tasks while one or more instruments are busy.

POLAR SYSTEMS

OPTION PTL-GUI

PTL Graphical User Interface

1 CONTENTS

LIST OF FIGURES

2 INTRODUCTION

In order to be able to display results from tests & measurements, a graphics driver may be included in the PTL-system. Graphs as results from measurements and/or typed input may be directly, displayed "live", on the computer's display.
Single line-pieces at a time, as well as whole arrays may be plotted, thereby having the possibility of automatically displaying minimum and maximum limits.

A wide range of default-settings, all of which are determining the visual shape on the screen, may be optionally altered with PTL's user friendly menu selection system.

The graph may be plotted to one of six popular printers such as Epson, Apple or Hewlett Packard.

In addition, to further enhance the 1722's screen output, several test & measurement adjustment features are available:

- Single or multiple bargraphs may present numerical data;

- Input from the operator may be obtained through slew-keys or menu selections;

Single bargraphs, with either a bargraph- or needle-pointer, as well as arrays of up to 8 bargraphs may be presented at a time, thereby having the possibility of automatically displaying minimum and maximum limits.

For user's input, a so called slew-key is incorporated, as well as a user-definable menu-selector.

The single bargraph- or needlepointer options may be used together with the slew-key module in one picture.

To enable both the plot and the adjust screen enhancements, two commands are added to the PTL language:

- Plot, to implement the plot options, and
- Adjust, to implement the adjustment screen enhancements

In the next chapters, first the Plot option will be described. Next, all the options for the screen enhancement will be explained.

3 COMPONENTS OF A GRAPH

The following figure will demonstrate the way a graphical image is build up:

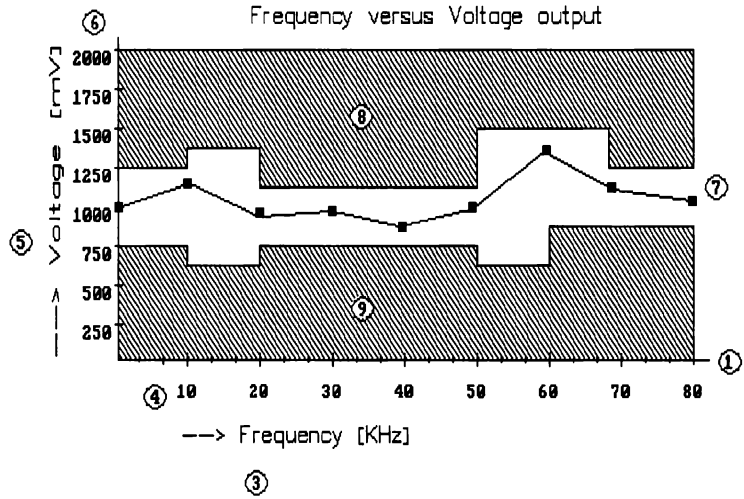


Figure 1 Sample graph

The picture is build up from the following components:

(1)	Axes
(2)	Title
(3), (5)	Legends
(4), (6)	Formatted scale
(7)	Line-pieces

(8), (9) Limits

Except for component (7), each of the above items are defined under the PTL menu selection Plot Action. The following syntax diagram is available:

Syntax:

```
Action = Clear screen;
         Def window;
         Def axes;
         Def Limits;
         Def text;
         Redraw definition;
         Arrayplot;
         Lineplot;
         Dump;
```

To start with a plot, the Clear Screen may be selected to clear the current contents of the screen. From PTL, many options are available to define a plot to the wishes of the user. The next paragraph will explore all the options of the Define selection.

When a plot is defined, result data may be on a line-piece by line-piece fashion, or drawn all at once, using the Lineplot or Arrayplot options. Finally, a plot may be dumped to a graphics printer. A selection of 6 types of printers is available to suit most applications.

3.1 Define

As mentioned above, many options are available to define a graphical PTL picture. All of these options are available through the DEF <Type> menu selections. They are:

- DEF WINDOW

This option defines the data-range of a plot. Four parameters Xmin, Xmax, Ymin, and Ymax are defined;

- DEF AXES

This option enables the user to setup axes with ticks.

- DEF LIMITS

To visualize limits on measurements, PTL is also capable of drawing limits in the graph.

- DEF TEXT

This option allows the user to draw some labels and titles for the graph;

- REDRAW DEFINITION

This option redraws the currently defined graph, based on both default and defined parameters;

3.1.1 Def window

Each plot uses four parameters to determine the data-range of both the x-data and y-data, called a **window**. They are for the x-data:

- Xmin
- Xmax,

and for the y-data:

- Ymin
- Ymax

Example: When data is to be plotted, of which the data in the x-axis ranges from 0..100 mV, and the data in the y-axis varies from -100..100 mA, the four parameters may have the following values:

Xmin = 0
Xmax = 100
Ymin = -100
Ymax = 100

Syntax:

```
Action = Define window;      Xmin=<-1e99..1e99>;
                             Xmax=<
                             1e99..1e99>;Ymin=<-
                             1e99..1e99>;   Ymax=<-
                             1e99..1e99>
```

3.1.2 Def Axes

Axes are defined using the following parameters:

- Xticks

- Xskip
- Xformat

- Yticks
- Yskip
- Yformat

Both the x-axis and the y-axis have corresponding parameters, which have identical functions. Therefore, only one explanation is required:

- Xticks, Yticks
Ticks represent the number of ticks drawn on an axis. The way a tick looks, is determined together with the next parameter, the skip count.

- Xskip, Yskip
A skip-factor may alter the way ticks and the corresponding scales are drawn: At each major tick a scale factor will be plotted. Every Xskip and Yskip ticks, a major tick is drawn. A major tick is drawn a little larger than a minor tick.

Example: In the plot-example of figure 1, Xticks=24, Xskip=3.

- Xformat, Yformat
Scales are drawn as well on the plot, just below the x-axis, and just left of the y-axis. The representation of both scales is defined using these parameters, with a standard Fluke BASIC format (See PRINT USING syntax in the BASIC Reference manual for details on the format). Example: "S###.#"

Syntax:

```
Action =Def axes; Xticks=<1..100>;Yticks=<1..100>;
             Xskip<1..25>; Xformat=<Format string>;
             Yformat=<Format string>
```

Format string:= Basic format string (S###.###)

Warning: An erroneously defined formatstring may cause a fatal program error. Be sure to enter a correct format.

3.1.3 Def Limits

The following figure illustrates the four possible types of limit presentations, if enabled, of the graphical package:

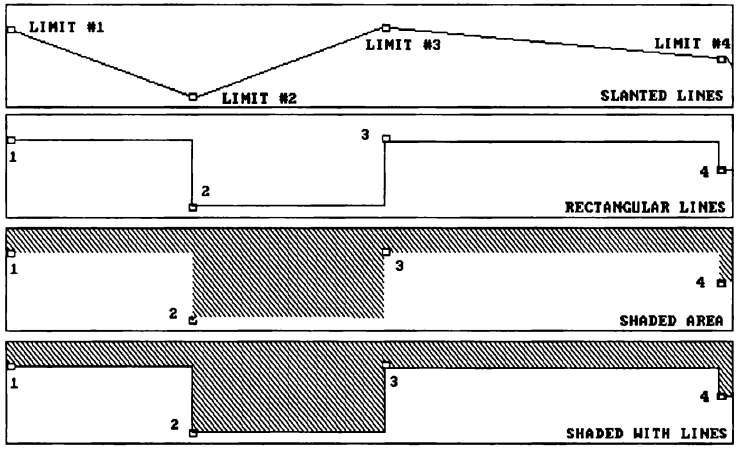


Figure 2 Limit types

Each limit value is drawn at a major x-axis tick. Using limits, an operator obtains a direct visual representation of, for instance, limit values on measurements. When the shaded type is selected, a shading percentage may be entered to define the shading raster. A typical value would be 2, 3 etc. A value of 1 will draw a solid box, a value of 20 will draw a shading grid with shading lines 20 dots apart.

Syntax:

```
Action =Def limits; <Type>; <Set limit data>;
```

Type := None
:= Slanted
:= Rectangle
:= Shaded; Shading raster= <1..20>

:= Shaded+line; Shading raster=<1..20>

```
Set limit data := All limits; Minima=<- 1e99..1e99>;
                  Maxima=<-1e99..1e99>

                := One by one; [X=<-1e99..1e99>; Min=<-
                  1e99..1e99>; Max=<-1e99..1e99>] []
```

3.1.4 Def text

The graphical support of PTL allows it to define text on the graphics screen. The options are:

- Title;
- Xlegend;
- Ylegend;

- Title

A plot title, when defined, will be displayed at the center-top of the graphical screen. Example:
"Voltage/current plot"

- Xlegend

A legend for the x-axis will be drawn, when possible, at the center of the right part of a plot. Additional units, and/or an arrow may be added to this string. **Example:**

```
"--> Voltage [mV]".
```

- Ylegend

As on the X-legend, a legend for the y-axis may be defined as well. The text is drawn at a rotation of 90 degrees, and is, when possible, in a centered fashion. Example: "--> Current [mA]".

Syntax:

```
Action= Deftext;Title=<text>;Ylegend=<text>;
        Xlegend=<text>
```

4 PLOTTING DATA

4.1 Plottypes

Traditionally, two types of plots can be distinguished: an X-t plot, and an X-Y plot. Both type of plots can be implemented in PTL by supplying the graphics driver the correct database.

4.1.1 X-t plots

An X-t plot is typically a plot, where the time-relation of a signal, such as a measurement, is being presented on the graphics screen. The x-axis functions as the x-axis, and the y-axis as the signal.

In the graphics driver of PTL, a data point (x_i,y_i) is drawn as a line-piece to another point (x_j,y_j). Data points are retrieved from two array variables supply() and result(). The x-points x_i are retrieved from supply(i), and y-points from result(i). By filling the supply-array with equally spaced values, the effect of time will be simulated. For instance, when x-axis parameters are as follows:

Xmin=0, Xmax=100, Xlegend="--> Time (min)", Xticks=10, Xskip=1

and supply(0)=0, supply(1)=10, supply(2)=20 ..., the effect of an x-t plot will become a fact.

It is the responsibility of the engineer to store all required data in result(), supply() and no_result% before calling the plot-driver.

4.1.2 X-Y plots

An x-y plot is a plot, where different signals may be plotted versus other signals. For instance, the relation between voltage and current may be plotted in an x-y plot.

To define such a plot in PTL, both the result() and supply() array can be filled with measurement data. By selecting the correct data-window, plotting of both array's will be result in a neat x-y plot.

4.2 Plot methods

Besides the type of plots, two possibilities for the plot-action itself may be selected: plotting all data in one plot-action, or, at a piece-by-piece fashion, where results of measurements may be monitored on the graphics screen step by step.

4.2.1 Linetypes

There are several line-types defined. The following figure will demonstrate the types:

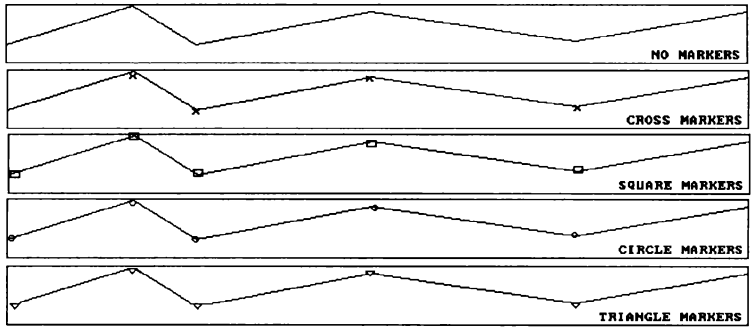


Figure 3 Linetypes

Each type specifies the type of marker drawn at each point of the graph. When array- or lineplot is selected, a linetype may be defined.

4.2.2 Arrayplot

When a plot is defined, data may be plotted. There are different types of plotting available: each point (x,y) is connected with a line from the previous point (x0, y0), or all the available data points (x_i, y_i) will be plotted at once, one after another. In the Action= menu, a selection can be made. Arrayplot selects PTL to plot all available data at once. Data is plotted as follows:

All x-values are taken from the global array: supply(), and all y-values are obtained from the global array result(). A point (x_i, y_i) is retrieved in the array's in location (i) of the array:

x_i = supply(i)
y_i = result(i)

The number of results is stored in the global variable: no_result%, which is imported in the plot-module. The engineer is responsible for maintenance of the three variables.

Syntax:

```
Action=Arrayplot; Linemarker=<Type of linemarker>;

Type of linemarker =  None / Cross / Square / Circle
                    / Triangle
```

4.2.3 Lineplot

Instead of plotting all data at the same time, a line-by-line method may also be selected. This method enables an operator to watch the result of each measurement separately. Each line-piece will be drawn using a point (x,y). Data may be plotted from the keyboard, or from the variable result(0).

Each point (x,y) may be entered from the keyboard, or result(0) may be used for the y-value.

Syntax:

```
Action=Lineplot; Linemarker=<Type of linemarker>;
              ; X=<-1e99..1e99>;
```

```
; Y=<Result / -1e99..1e99?;

Type of linemarker =    None / Cross / Square / Circle
                       /Triangle
```

4.3 Printing

When a plot is completely drawn, it may optionally be dumped to a printer. PTL has drivers for several printers:

- Epson FX
- Epson Rx
- Epson Mx-100
- Apple Write
- Star Delta
- HP Thinkjet

Syntax:

```
Action=Dump; Printer=<Printertype>
```

5 ADJUST OPTIONS

Using the adjust statement, the following screen enhancements are available:

- Single bargraph

A large, horizontally drawn bargraph meter, with a bargraph pointer, filling a 1722 display's top half;

- Single pointer

A large, horizontally drawn meter, with a needle-pointer type of pointer, filling the display's top half;

- Multiple bargraphs & pointers

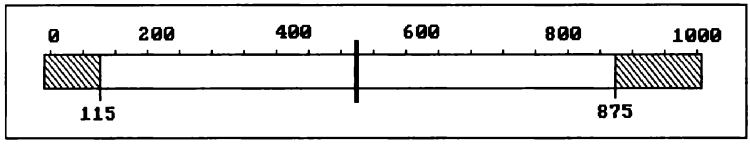
Up to 8 vertically drawn bargraphs, each with an independent set of limit values, tags and units, as well as pointer type;

In the following paragraphs each of the options will be described.

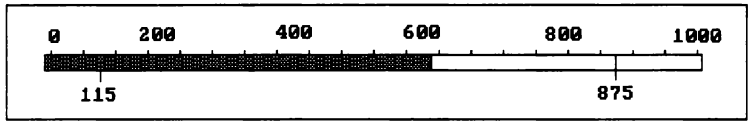
5.1 Single bargraph

A bargraph is an illustrative method of presenting a varying numerical quantity, on the computer's display screen.

Figure 1, demonstrates the display of such a single bargraph.



Single pointer



Single Bargraph

Figure 4 Single bargraphs

A single bargraph may be defined using the following parameters:

- Type
- Tag
- Hihi limit value
- Hi limit value
- Lo limit value
- Lolo limit value

5.1.1 Type

Two types of pointers for the single bargraph are available:

1. Bargraph

This type of pointer resembles a thermometer scale: A bar will move on the screen

from left to right: when a value is low the bar moves to the left of the picture, when a value increases, the bar will move towards the right edge of the screen.

2. Meter scale

This type of pointer resembles an analog meter instrument, where a needlepointer moves left or right dependent on the result value.

The latter pointer is somewhat faster to update, due to the fact that a lot more drawing is involved in updating a bargraph. On the other side, a bargraph may be visually more clear, since from a larger distance, a bargraph will be more recognizable.
Using the bargraph pointer, another advantage is that when a signal goes beyond it's limit, the bargraph will be drawn highlighted (100% filled versus 50% filled normally).

5.1.2 Tag

A tag may be drawn at the lefttop corner of the bargraph. Any text may be used here, such as text, describing the bargraph ('Voltage output [mV]' etc).

5.1.3 Hihi- & Lolo- limits

Hihi- and lolo limit values will define the data-range of the data, plotted on the bargraph. Typical limits are hardware channel limits, for example 4..20 mA, or 0-10 V. These two parameters are used internally by PTL to calculate the position on the screen for live data (scaling factors).

5.1.4 Hi- and Lo- limits

Hi- and Lo- limits are signal limits , determining the type of bar drawn on the screen. When a signal exceeds either a high- or a low-limit value, the bar changes from 50% filled, to a full 100% filled solid bar. When the signal de- or increases to a level within these limits, the bar changes again back to a 50% filled bar. Hi- and low limit detection is therefore realized visually. Of course, the latter is only true for a bargraph pointer.

5.1.5 Syntax

To display/define a bargraph, use the following syntax diagram:

```
Adjust("Action")

 Action=  Display; Mode= Single bargraph; <Options>

Options=    Type=<Bargraph / Meter scale>;
            Tag= 'Tag';
            Hihi limit= <-1e99..1e99>;
            Hi limit= <-1e99..1e99>;
            Lo limit= <-1e99..1e99>;
            Lolo limit= <-1e99..1e99>.
```

To update a bargraph, use the following command:

```
Adjust("Action=Update");
```

5.1.6 Example

When an IEEE temperature meter gives a signal output of 0..100 DEG C, and the process we want to monitor does not allow temperatures over 75 DEG C, and not under 50 DEG C, a definition of a bargraph could be:

```
Type=Bargraph
Tag='TEMP [DEG C]'
Hihi limit= 100
Hi   limit= 75
Lo   limit= 50
Lolo limit= 0
```

5.2 Multiple bargraphs

With the Graphics User Interface, it is also possible to display an array of up to 8 bargraphs at the same time on the display. A picture may appear as shown in Figure 5 for all 8 bargraphs, or Figure 6 for all 8 meter scales :

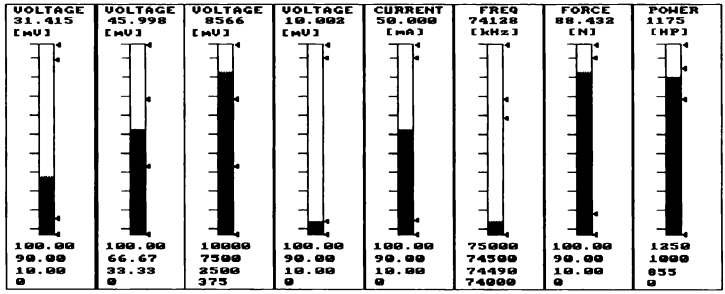


Figure 5 Multiple bargraphs

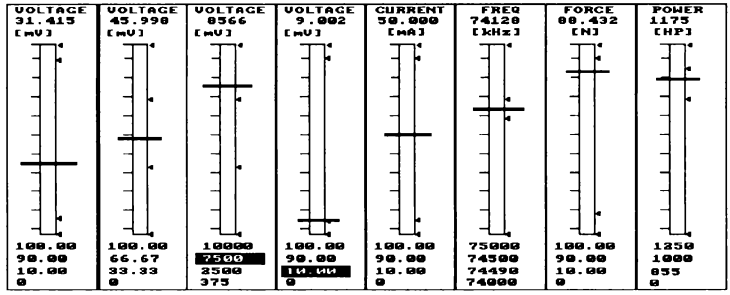


Figure 6 Multiple pointers

Each bargraph may be defined separately using the following parameters:

- Type
- Tag
- Units
- Hihi limit
- Hi limit
- Lo limit
- Lolo limit

5.2.1 Type

As with single bargraphs, a type may be defined for each bargraph. Again, a type may be **bargraph** or **meter-scale**, as well as **Off**. In the latter case, no bargraph is drawn at all, in order to leave the location empty.

The needle-pointer may be selected in the more time-critical situations, and the bargraph type in situations where an operator requires a clear indication of a signal

value.

Both pointer types may be mixed in one picture.

5.2.2 Tag

A tag may be defined for each bargraph. The tag will be displayed in the top section of the bargraph. A tag may be up to 8 characters long.

5.2.3 Units

Units may be defined to each bargraph. The units will be displayed in the top section of the bargraph, just below the value field (See Figure 2). Together with the tag, and 'live' value, a signal will be displayed in the top section of the bargraph as follows:

<Tag>
<Value>
<Units>

Using a tag and units, a numerical presentation of measurement data will complement the graphical section of the bargraph.

5.2.4 Limit values

Each bargraph is defined with four limit parameters: Hihi- and Lolo-limits, parameters used to scale a measurement-value for the bargraph length, and Hi- and Lo-limits, representing signal limit values. Each bargraph may have an individual set of Hihi, Lolo, Hi- and Lo- limits. The example of Figure 2 will demonstrate several

limit values. The four limit parameters will be displayed in the bottom section of the bargraph as follows:

<Hihi limit>
<Hi limit>
<Lo limit>
<Lolo limit>

5.2.5 Limit checking

When a signal exceeds either one of the four limit values, the corresponding field will be displayed in reverse video, in order to signal this situation to the operator. When the signal's error condition resets, the corresponding field will be displayed in normal video again.

Using this method, a visual overview is obtained of the signals status' situation.

Using a bargraph type of pointer, another limit checking method is selected: when a signal exceeds limits, the bargraph will change from a 50% filled bar to a 100% solid bargraph, thereby highlighting the bargraph in case of exceeding a limit.

Updating the pointers may take place one by one, or all 8 at one time.

5.2.6 Syntax

To define a set of 8 bargraph pointers, use the following syntax diagram:

```
Adjust("Action")

Action= Display; Mode= Multiple bargraphs; <Preset
        Option>
```

Preset Option = <All / Single><Options>

Options= Type=<Off / Bargraph / Meter scale>;
Tag= 'Tag';
Hihi limit= <-1e99..1e99>;
Hi limit= <-1e99..1e99>;
Lo limit= <-1e99..1e99>;
Lolo limit= <-1e99..1e99>.

5.2.7 Bargraphs update

To update the bargraphs, use the following command:

```
Adjust("Action=Update");,

or

ADJUST("Action=Update specific; Bargraph no=<No>");
```

When selecting the general update command, data from result(0) will be drawn at bargraph 0, result(1) at bargraph 1,.....,result(7) at bargraph 7.

When selecting the specific update command, data will always be taken from result(0), no matter which bargraph number is specified in the Update command.

6 INPUT OPTIONS

The Graphical User Interface contains two methods of inputting data from the operator:

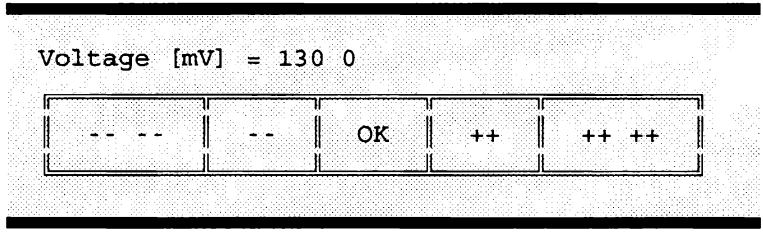
- Slew keys
An alternative method to enter a number from an operator;

- Menu selection
An alternative method to let the operator make a choice between several options;

The following paragraphs will describe both options.

6.1 Slew keys

Obtaining a number from an operator may take place using a slew key-bar, as demonstrated in the following figure:



6.1.1 Parameters

The following parameters are defined for a slew key-bar:

- Slew text
This text will be displayed at the left top corner of the slew-key bar, and may be the tag of a to-be-slewed signal;

- Slew step value
This parameter indicates the stepsize of a change in the PTL system variable result(0), when a specific slew-key is pressed;

When the slew-keys are displayed first, the value in result(0) will be taken as the starting value. Pressing the correct keys on the TSO will in- or decrement the current value. When 'OK' is entered, the current value will be restored in result(0), where it may be picked up by the measurement driver.

6.1.2 Inputting data

Pressing the "-- --" field will decrement current slew value with 10 times the slew step value;

Pressing the "-- " field will decrement the current slew value with 1 time the slew step value;
Pressing the "++" field will increment the current slew value with 1 time the slew step value;

Pressing the "++ ++" field will increment the current slew value with 10 times the slew step value;

Pressing "OK" will end the slew key input session by setting the PTL system variable pass%(0) to 1. The current slew value will be copied back to result(0).

The slew input may concurrently take place with a single bargraph: The bargraph is located at the top of the screen; the slew-key is located at the bottom of the screen.

6.1.3 Syntax

To draw & define the slew key use:

```
Adjust("Action=Display; Mode=Slew"; <Options>)
```

Options= Tag= '<Tag>';
Slew step= <-1e99..1e99>;

To input a value using the slew, define as follows:

```
Adjust("Action=Input; Type=Slew")
```

6.2 Menu selection

Another structure to enter data from the operator may take place using the menu-selector. When used, on the screen, a list with options (fields) appear for the operator. The currently selected field will be displayed in high-lighted, reverse video.

The screen-key, marked with 'Up' will select a field above the currently selected field;

'Down' will select a field under the currently selected field.

The key, marked with 'Select' will select the currently selected field and end the specific procedure. The selected field will be returned to PTL through the system variable result(0).

6.2.1 Syntax

Defining fields:

```
Adjust("Action=Input; Type= Menu; <Parameters>");
```

```
Parameters= Menu title    = '<Title>'
            Max items     =  <1..12>;
            Item no       =  <1..12>;
            Item text     = '<text>';
```

7 DEFAULT VALUES

Plot() and Adjust() may be defined using explicit commands, however, all parameters have default settings. In the next tables both default values will be displayed.

7.1 Plot default values

The following values are being used:

Parameter	Default value
Title	PTL graph
Xlegend	--> X
Ylegend	--> Y
Xmin	0
Xmax	100
Ymin	0
Ymax	100
Xticks 10	
Yticks 10	
Xskip	1
Yskip	1
Xformat	S###

7.2 Adjust default values

The following values are being used:

Parameter	Default value
Screentype	Single bargraph
Type	Bargraph pointer
Tag	"Result value"
Units	""
Hihi limit	100
Hi limit	90
Lo limit	10
Lolo limit	0
Slew key step	1
Slew title	"Slew value ="

7.3 Plot implicit defining

Each plot parameter may be modified using the specific Plot Define commands. To automate the process of composing a graph, several parameters are modified implicitly. For instance, when a new Xmin-Xmax range is selected, automatically the Xformat string is adapted to suit the new data-range. This generated string may be

overwritten again, but in most cases this will not be necessary. A full list of implicit parameter defining is given in the following table:

Parameter	Auto-defines
Xmin or Xmax	Xformat
Ymin or Ymax	Yformat

8 DATABASE

In this chapter, the database as used by the plotting routines, are displayed and defined. Each date is listed below:

8.1 result()

This array is defined in a PTL program as follows:

IMPORT result(25)

1. This array represents the y-axis values, and will be used by the graphics driver to plot in the Arrayplot fashion all of the results of a measurement, or in the Lineplot fashion, where result(0) will be used only.

2. result(0..7) represent the contents of the bargraphs 0..7, when applicable.

The number of elements used for a plot for the
Arrayplot method is defined by the variable no_result%.

It is the responsibility of the engineer to supply the correct values in this array.

8.2 supply()

This array is defined as follows:

IMPORT supply(25)

and represents for the graphics driver the x-axis values, used in the Arrayplot selection. When a line-by-line fashion is selected with Lineplot, only element #0 is used (supply(0)).

It is the responsibility of the engineer to supply the correct values in this array before parsing it to SPLOT().

8.3 no_result%

The number of results of a measurement is stored in this variable. The number of results for a plot, and therefore the number of line-pieces, is limited in PTL to 25.

8.4 pass%()

When a slew is setup, pass%(0) is reset to 0. During the slew procedure, pass%(0) stays at 0 until 'OK' is pressed, at which time this flag is set: pass%(0)=1. Using this method, the test program will know when the operator has entered a new value.

INDEX

POLAR SYSTEMS

OPTION PTL-ICON

PTL Icon Editor Package

1 CONTENTS

2 CAPABILITIES PTL/ICO OPTION

2.1 Introduction

This manual describes how to use the Icon Editor in the Polar Test Language PTL Version 1.3.

This information is supplied as option :
PTL/ICO Icon Editor Support

2.2 The Icon Editor

The Icon Editor Program is a utility program for the Fluke 1722A Instrument Controller and the Fluke 1752A Data Acquisition System. With the PTL/ICO module two powerful features of the 1722/-52A display, graphics and the touch sensitive overlay, are fully integrated with the Polar Test Language.

In almost all test applications there is a need for Interaction with an operator. Data is presented to and commands are accepted from a operator. Graphics are very effective in conveying information to a human, while a touch sensitive display is a very effective control panel. Naturally, PTL users want to take full advantage of these capabilities.

Until now, the bad news was that creating graphics-and-touch-display based "user interfaces" was a very software intensive job. In Fluke's experience no less than 50% of a typical application program consist of user interface software! With standard PTL most normal output is already handled by the language itself. With the Icon Editor option in PTL the generation of effective graphics interfaces is made quick and easy, offering a considerable time-saving in program development.

2.3 Editor for pictures

The Icon Editor enables the user to prepare displays with both alphanumeric and graphic information. This is done by making selections from a set of predefined symbols, called Icons, and positioning these anywhere on the screen. Different sets of icons may be used, depending on the kind of pictures that must be made.

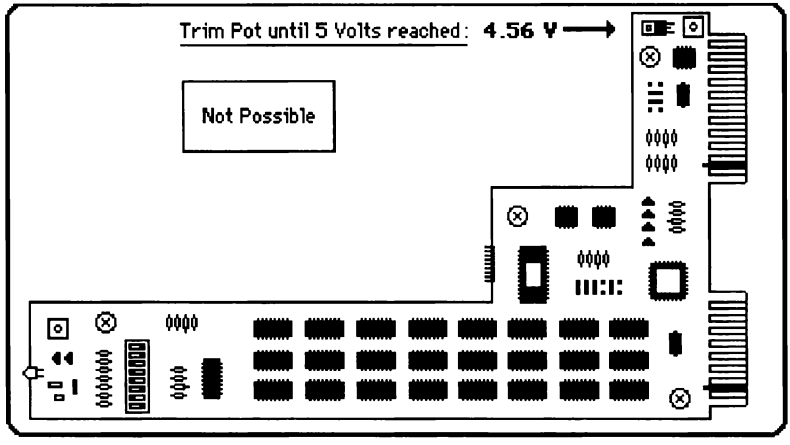


Figure 1: Example of a printed circuit board

Then the picture can be completed by drawing various styles of lines, for instance to interconnect electronic symbols in a circuit diagram. Furthermore text can be added to the picture as well as touch sensitive areas.

From all this information a display file is created which can be used in a PTL procedure.

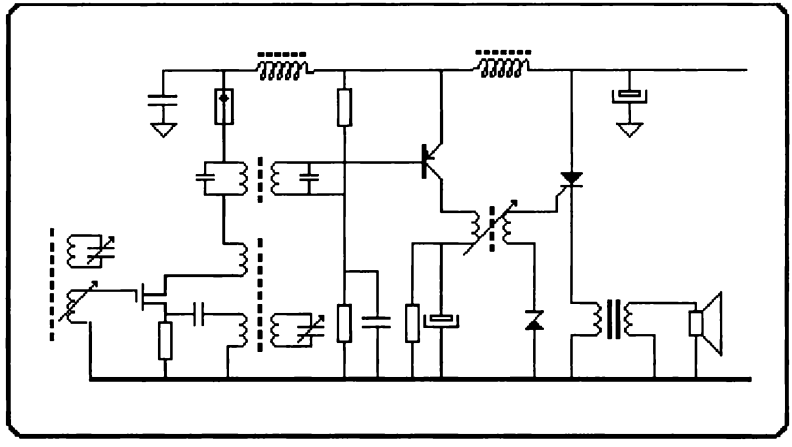


Figure 2: Example of a radio with "electronic" icons

The Icon Editor Program provides a complete set of commands to perform the following functions.

* Selecting an icon from a set
* Positioning an icon on the screen
* Drawing lines of various styles
* Adding text and numeric data fields
* Adding touch sensitive zones
* Creating and editing icons
* Making a hard copy of the picture on a printer

2.4 Use pictures in PTL

After a picture has been stored on disk, it can be called from PTL. Test results can be inserted into it and keys can be interrogated.

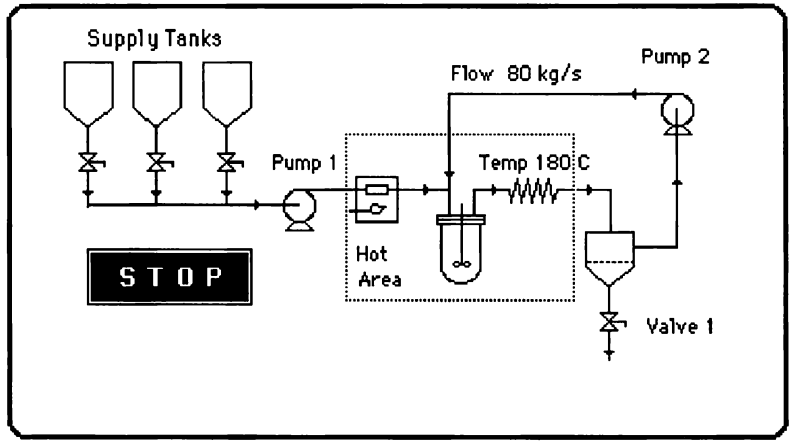


Figure 3: Example of "Chemical" icons

The following commands are available:

* Put a picture on the screen
* Add standard PTL variables
* Add results and parameters
* Get key touched

3 INTRODUCTION

3.1 Icon editor menus

To use the Icon Editor Program you have to make appropriate selections from a number of menus. Figure 4 shows the structure of these menus .

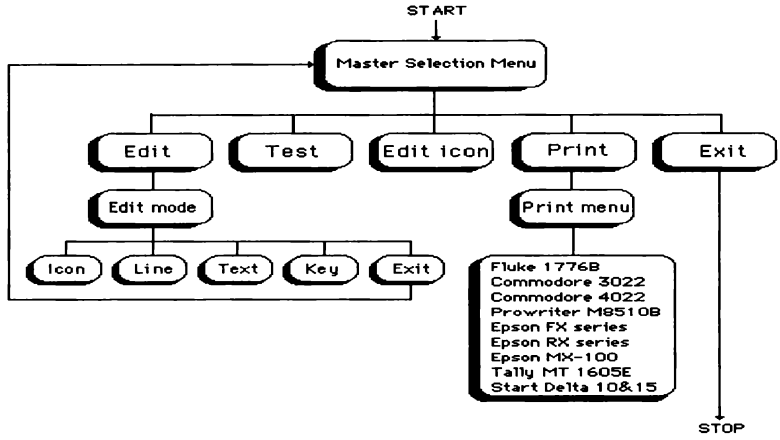


Figure 4: Icon Editor menu structure

3.2 Files and the Icon Editor Program

With the Icon Editor program you can create or edit a Picture Source File, much like creating a procedure with a text editor. When you stop execution of the Icon Editor program, a second file is automatically generated by a built-in compiler: the Picture Object File. The file name extension for Picture Source Files is ".PSR", while the

extension ".POB" is used for Picture Object Files.

Only Picture Source Files can be edited; at the end of the editing session a new Picture object File is compiled. The Picture Object File is the "executable" form of the picture file; it can be used in a PTL procedure

One more file is necessary to create or edit a picture: the Icon Definition File (extension ".IDF). A given Icon Definition File contains a set of predefined icons, e.g. electronic circuit symbols or piping engineering symbols. Several different Icon Definition Files may be present with the Icon Editor program. When you start execution of the Icon Editor program, you may specify the desired Icon Definition File.

During execution of a PTL procedure, neither the picture Source File nor the Icon Definition File are required to use an "operator interface" that was created with the Icon Editor. Only the Picture Object File and the linkable subroutine are needed for that.

4 THE ICON EDITOR

4.1 Start the Icon Editor Program

In the PTL main menu choose the ICON key and press return to start the editor.

From the FDOS > prompt, type:

```
ICON <RETURN>
 or
ICON [<device>]<icon definition file> <RETURN>
```

In the first case the default Icon Definition File is used (ELEC.IDF), containing a set of symbols used in the electronics industry. The second method allows you to specify a different Icon Definition File.
(Besides ELEC.IDF one other set of icons comes standard with version 1.6 of the Icon Editor Support: CHEM.IDF, containing chemical symbols.)

If no device is specified with the Icon Definition File, the Icon Editor assumes that the desired file resides on the system device.

4.2 Exit from the Icon Editor Program

In order to leave the Icon Editor Program, you must select the EXIT field in the master selection menu. Type <CONTROL/C> or <CONTROL/P> to return to the master menu from any of the sub-menus.

4.3 Using the Icon Editor Program

Once you have started the Icon Editor Program, the master selection menu is displayed. From this menu you can select the various operating modes. You do this by selecting the required field with the up/down arrow keys and then depressing the <RETURN> key. (The selected field is highlighted.)

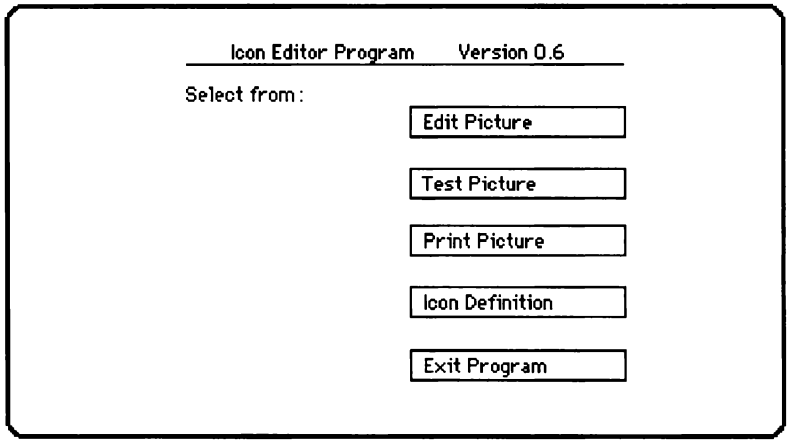


Figure 5: Master selection menu

Select one of the following options:

* Edit Picture
* Test Picture
* Print Picture
* Edit Icon Definitions
* Exit from Program

The following paragraphs give an explanation of these menu options.

4.4 Edit Picture

This command causes the screen to be cleared, after which you are requested to enter the name of the Picture Source File to be edited. This may be either a new file name or an existing one.

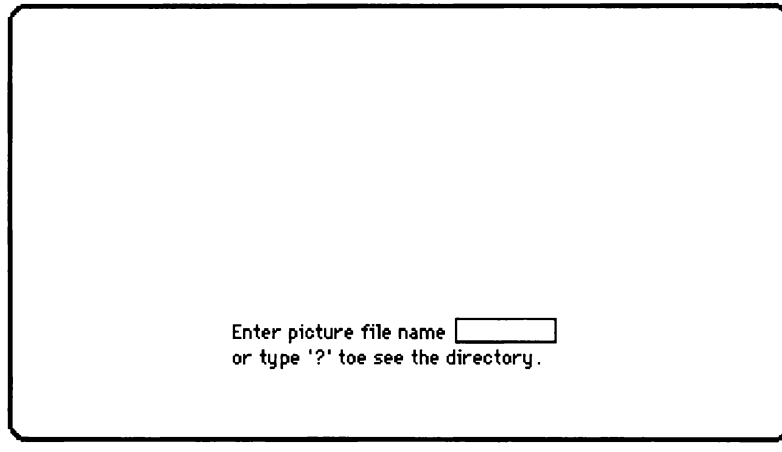


Figure 6: Specifying a picture source file

The directory of the system device can be listed by typing a question mark. This is very handy when you are not sure what the name of the file to be edited is.

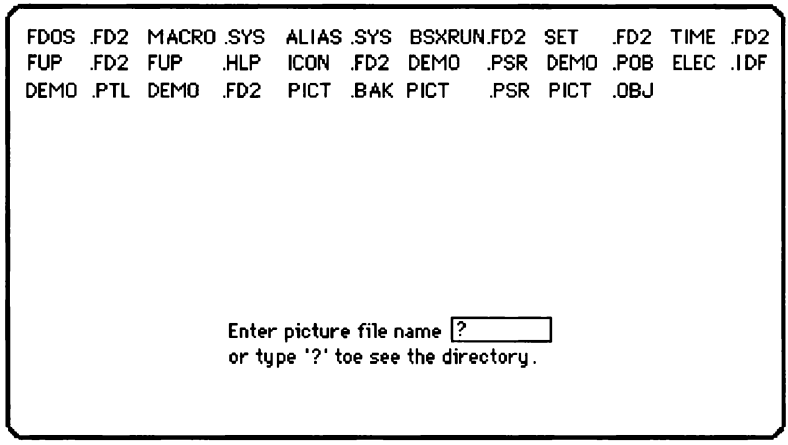


Figure 7: Listing the file directory

Once you have entered the file name the screen is cleared again, whereupon the picture to be edited appears. In case a new Picture Source File was specified, the display will remain blank. The edit menu is displayed at the top of the screen.

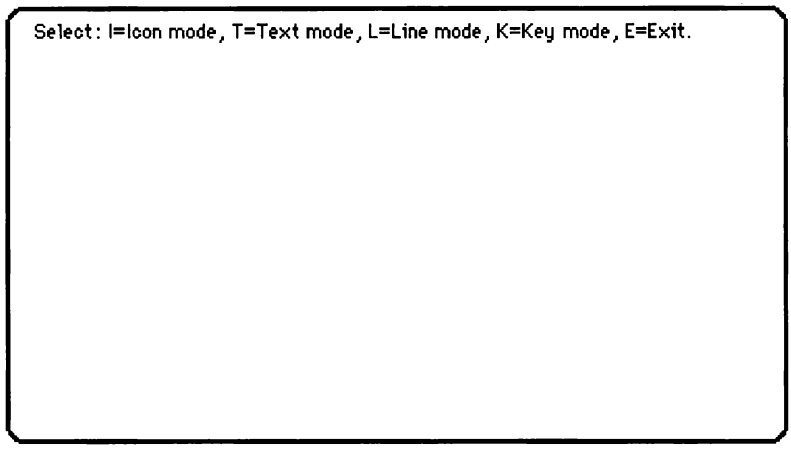


Figure 8: The edit mode menu

Now type I, L, T, K, or E to select from:

* Icon mode
* Line mode
* Text mode
* Key mode
* Exit to Master Selection Menu

After editing the picture, you can make the changes permanent by selecting the Exit command from the edit menu. At this point a new source file is created (*.PSR) and the previous source file, if any, is renamed to "*.BAK (for back-up). In addition a compiled display file (*.POB) is generated. Note that "*" stands for the name of the file. See figures 9 and 10.

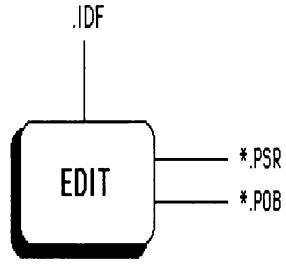


Figure 9: Creating a new picture file

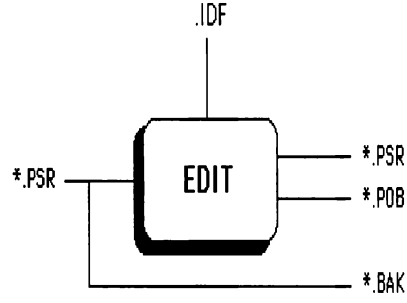


Figure 10: Editing an existing picture file

4.4.1 Icon mode

In this mode you can select icons from the Icon Definition File.
First type "S" (for Select) in order to display all the available icons. Then move the cursor to the desired icon with the arrow keys and press <RETURN> to validate your selection.

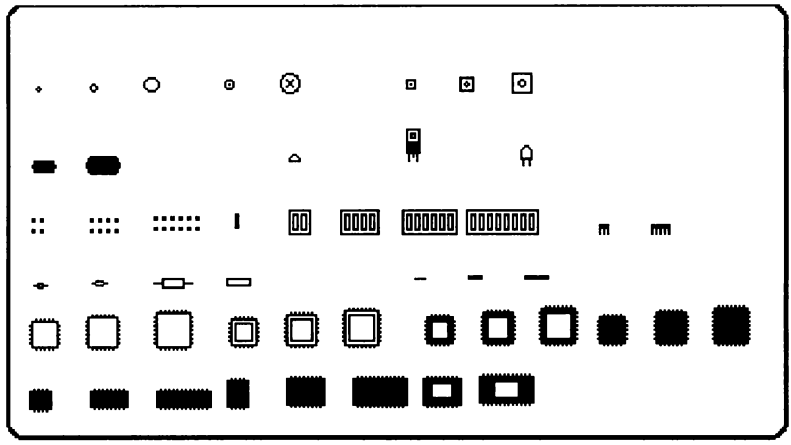


Figure 11: Selecting an icon

The picture that you are editing is displayed again and you can move the solid square cursor with the arrow keys. Press the <RETURN> key to place the selected icon at the cursor position.

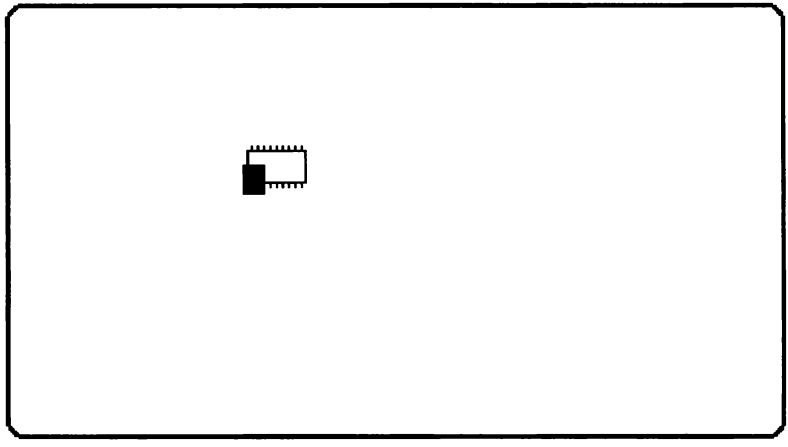


Figure 12: Putting a selected icon on the screen

The same icon may be put on the screen at another location by moving the cursor and hitting <RETURN> again. The last icon selection remains valid until a new one is made.

In order to speed up cursor movements, the arrow keys can be preceded with a repeat factor. For example "100->" moves the cursor 100 pixel positions to the right.

During the editing process you may find out that an icon should be moved to another position on the screen. To do this you must "lock" the cursor on that icon. Four keys serve this purpose: <BACKSPACE> to lock the cursor on the first icon you placed, "F" (Forward) to lock on the next one, "B" (Back) to lock on the previous one and <LINEFEED> to lock on the one you placed last. When you hit any of these four keys, the message "(Locked)" appears at the top of the screen. Now you can use the cursor control keys to move the icon around. You may add a repeat factor if desired.

Once the icon is where you want it, you can rotate it with the "R" key, erase it with the <DELETE> key or fix its place with the <RETURN> key. Either <DELETE> or <RETURN> unlocks the cursor from the icon. The message "(Locked)" will disappear.

Summary of key commands:

S - Selects	Displays the icons in the Icon Definition File.
<RETURN>	Selects an icon from the Icon Definition File. Also fixes the icon at the cursor position and unlocks the cursor.
<ESC>	Returns control to the edit menu.
<BACKSPACE>	Locks the cursor on the icon that was placed first.
<LINEFEED>	Locks the cursor on the most recently placed icon.
R - Rotate	Rotates the icon on which the cursor is locked counter clockwise by 90 degrees.
B - Back	Locks the cursor on the previous icon.
F - Forward	Locks the cursor on the next icon.
<DELETE>	Deletes the icon on which the cursor is locked.

4.4.2 Line mode

In this mode you can draw lines anywhere on the screen and in any direction. A line consists of a series of dots, or "pixels" (picture elements). The screen capacity is 244 x 640 pixels.

In the line mode the cursor is an X type cross. It can be moved around with the four arrow keys, one pixel position at a time.
Because of the high resolution (large number) of pixels/cm) the cursor moves only a small distance each time an arrow key is hit. To speed up cursor movements the arrow keys can be used with a repeat factor. For example "120->" moves the cursor 120 pixels to the right.

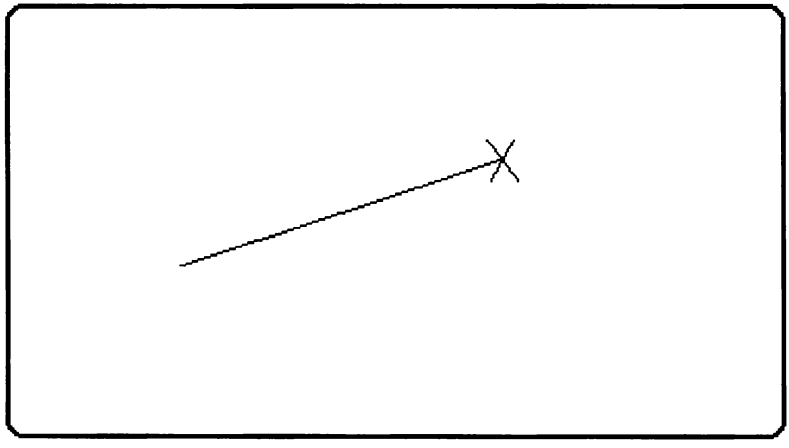


Figure 13: Display in line mode

To draw a line first fix the begin point of the line at the cursor position by hitting the <RETURN> key. Then move the cursor to the desired end point. The line will

follow the cursor like a rubber band. You can fix the location of the end point by pressing <RETURN> a second time.

If an existing line must be moved, then first lock the cursor on either its begin point or its end point. Four keys can be used for this: <BACKSPACE> to lock the cursor on the begin point of the line that was drawn first, "F" (Forward) to lock the cursor on the next point "B" (Backward) to lock the cursor on the previous point and <LINEFEED> to lock the cursor on the end point of the line that was drawn last.

Striking any of these four keys causes a "(Locked)" message to appear at the top of the screen. Now move the line's end or begin point with the arrow keys (optionally with a repeat factor).

Note that if two line segments meet, e.g. at the corner of a rectangle, that the Forward or Backward keys must be pressed to step from the end point of the first line to the begin point of the next and vice versa, even though these points coincide on the screen.

Once the desired place is reached, the line style can be changed with the "C" key (for Change), the line can be erased with the <DELETE> key or fixed in place with the <RETURN> key. Either <DELETE> or <RETURN> unlocks the cursor from the line.

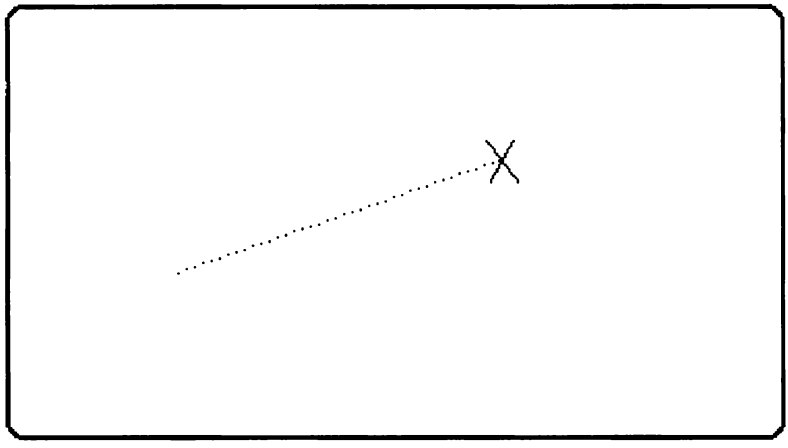


Figure 14: Line style changed to dotted line

Summary of key commands:

<RETURN>	Fixes the begin point of a line at the cursor position.
<ESC>	Returns control to the edit menu.
<BACKSPACE>	Locks the cursor on the begin point of the line drawn first.
<LINEFEED>	Locks the cursor on the end point of the line drawn last.
B - Back	Locks the cursor on the previous point.
F _ Forward	Locks the cursor on the next point.

<DELETE>	Deletes the line on which the cursor is locked.
C - Change	Changes the style of the line on which the cursor is locked. Each time "C" is hit, the next line type will be chosen. The available styles are solid, bold, and three types of dotted and dashed lines.

4.4.3 Text mode

In this mode the Icon Editor Program behaves like a simple text editor. Move the text cursor with the four arrow keys to the place where you want to put text. After fixing the beginning of the text field with the <RETURN> key, type in the line of text. The only editing feature available at this point is the <DELETE> key, which deletes the last entered character. When the line is in order, make it permanent with the <RETURN> key.

Existing text fields cannot be edited or moved like icons and lines. If you need to change or relocate a text line then delete the line at the old location and retype it at the new location.

To delete a text line first look the cursor on the text field. Four keys reserved for this: <BACKSPACE> to lock the cursor on the first text field entered, "F" to lock the cursor on the next text field, "B" to lock the cursor on the previous text field and <LINEFEED> to lock the cursor on the text field that was entered most recently. When you hit any of these four keys, a "(Locked)" message appears at the top of the screen. Now you can erase the text field with the <DELETE> key.

Summary of commands:

<RETURN>	Fixes the beginning of the text field at the cursor

position and enables the insertion of characters. A second <RETURN> terminates insertion and stores the text line.

<ESC>	Returns control to the edit menu.
<BACKSPACE>	Locks the cursor on the first text field entered.
<LINEFEED>	Locks the cursor on the last text field entered.
B - Back	Locks the cursor on the previous text field.
F _ Forward	Locks the cursor on the next text field.
<DELETE>	Deletes the text field on which the cursor is locked.

Two types of text are allowed:

4.4.3.1 Regular text

This text will be displayed in the final program "as is".

4.4.3.2 Variable text

Variable text, PTL variables, can be put between the regular text. These numeric values and strings are inserted in a picture at run-time. All variables are preceded by an '@' sign, followed by format information and a letter to indicate the variable to be displayed.

E.g. the text "The result is @###.#A mV"

will display the last PTL result with one decimal behind and 3 numbers in front of the decimal point.

E.g.: "The result is 123.4 mV"

The variable text must be followed by a space character (if more text follows) or by no characters at all (at the end of a text line).

4.4.3.3 PTL variables allowed

The following variables can be inserted in the text:

Letter	Description	Default length
A	Result(0)	8
B	Units	4
C	Minimum limit	8
D	Maximum limit	8
E	Pass/ Fail step	4
F	Pass/ Fail task	4
G	Pass/ Fail procedure	4
H	Procedure name	32
I	Procedure version	16
J	Procedure date	16
K	Task name	32
L	Task number	2
M	Test date	15
N	Serial number of UUT	16
O	Remark string	32

4.4.3.4 Formatting variables

Variables must be specified with a format. Three different formats are allowed:

- Basic Format

This format is used exactly like the format modifier in the BASIC "PRINT USING" statement. The first character of a format modifier must be a "#" or "S". If a Basic format is used for strings, then only the length of the format string will be used to define the number of characters to be displayed.

Example: @S##.##C will display the minimum with 2 numbers before and two behind the decimal point.

- Free Format

This format uses the default formatting of Basic for numbers and strings. The length specified will be used to cut off to total length of the result. If the format will not fit, *'s will be used to indicate the error. The format modifier must be a number. If no number is specified, then the default length will be used.

Example: @10A will display the result in free format, to a maximum length of 10.

@K will display the task name with trailing spaces, to a default length of 32 characters

4.4.4 Key mode

In this mode you can add touch sensitive areas (keys) to the picture. These touch keys are reverse video rectangles of 6 by 2 characters, that are aligned with the 60 rectangular touch zones on the transparent screen overlay.

Once you have entered the key mode, a rectangular cursor with the size of a touch key appears. With the cursor control keys you can move the rectangle to the desired position. When you then hit <RETURN>, a message appears at the top of the screen, prompting you to enter the key number.

This must be a number in the range of 1 through 255; when the display is used in an application program, the linkable module will pass this key number to the user program when the key is touched.

After entering the key number the text cursor appears in the key. You can now enter two lines of text, although the length of these "lines" is limited by the size of the touch key. Each of the two required, simply hit <RETURN> twice.

A touch key can be deleted from the screen by moving the rectangular cursor to it and hitting <DELETE>.

Summary of key commands:

<RETURN>	Puts a touch key at the current cursor position. Once this is done, enter a key number (1 through 255) and two lines of text.
<ESC>	Returns control to the edit menu.
<DELETE>	Deletes the touch key at the cursor position.

4.4.5 Exit to the Master Menu

When this command is selected, the edited picture is made permanent and the Picture Object File is generated. The display is cleared, after which the Master Selection Menu appears.

4.5 Test Picture

The purpose of the test mode is that you can verify the touch key numbers and the array index numbers for the numeric data fields (explained below). This information is needed to correctly integrate the user interfaces that are designed with the Icon Editor into a user program.

Selection of the test mode causes the screen to be cleared, after which you are prompted to enter the name of the file to be tested. The directory of the system device can be listed by typing a question mark. This is convenient when you do not remember exactly what the name of the file to be tested is. The whole picture with all icons, lines, keys and text will be displayed.

In order to assign values to the numeric data fields in a picture, the user program must pass a number array to the linkable display subroutine. In the test mode, each format field shows its index in this number array.

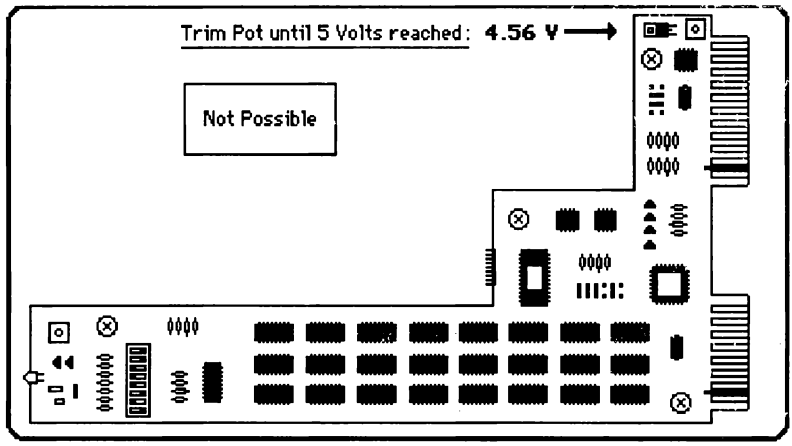


Figure 15: Testing a picture file

When a touch key is pressed, the linkable subroutine passes the key number to the user program. In the test mode the number of each key can be viewed by touching it.

4.6 Print Picture

This commands clears the screen, after which you are prompted to enter the name of the file to be printed. The directory of the system device can be listed by giving a question mark, in case you do not precisely remember the name of the file to be printed.

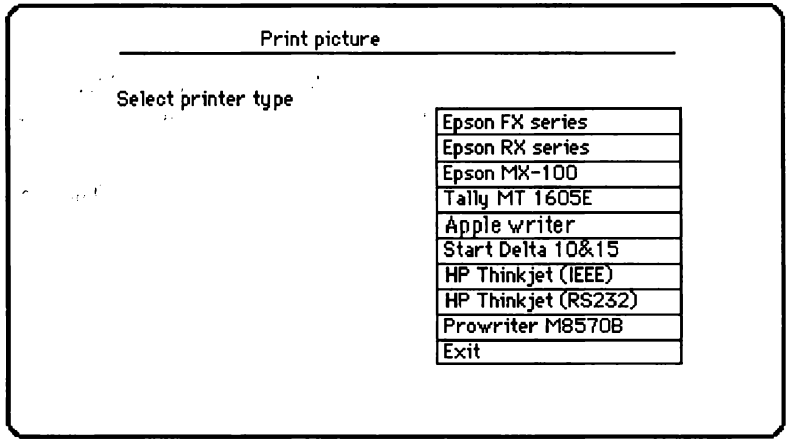


Figure 16: Printer menu

Then the printer menu is displayed. The UP and DOWN arrow keys allow you to select one of the following printers.

1) Epson FX Series
2) Epson RX Series
3) Epson MX-100 W 6
4) Tally MT 1065 E
5) Apple Writer
6) Star Delta 10 & 15
7) HP Thinkjet (IEEE)
8) HP Thinkjet (RS232)

4.7 Edit Icon Definitions

This command allows you to edit the icons in the Icon Definition File.
Selecting this command first causes all icons to be displayed. With the cursor control keys a particular icon can then be selected. Upon hitting <RETURN>, a threefold magnification of the selected icon is displayed inside a rectangle. The rectangular frame indicates the outer limits of the space available for the icon.

Editing an icon is similar to the line edit mode in editing a picture. The commands for drawing lines, moving them around and deleting them are identical, but here the lines are used to create an icon. Once the icon is finished, the icon selection page can be displayed again by hitting the <ESC> key.

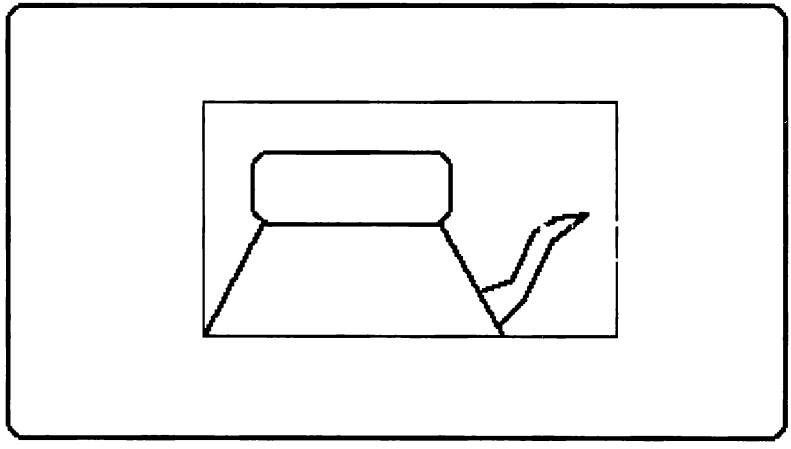


Figure 17: Icon being edited

The edited Icon Definition File will be saved when you select Exit from the master menu.

Summary of commands:

<RETURN>	When icon selection page is shown: changes display to magnified picture of the selected icon, enabling icon editing. When in icon edit mode: fixes the begin point of a line in the icon window at the cursor position.
<ESC>	Makes icon selection page reappear when magnified icon is shown. Causes exit to edit menu when icon selection page is shown.
<BACKSPACE>	Locks the cursor on the begin point of the first line drawn.
<LINEFEED>	Locks the cursor on the end point of the last line drawn.
B - Back	Locks the cursor on the previous end or begin point.
F _ Forward	Locks the cursor on the next begin or end point.
<DELETE>	Deletes the line on the end or begin point of which the cursor is locked.

4.8 Exit from program

When you select the Exit option from the master menu, execution of the Icon Editor program is terminated. If you modified any icons, the changes are made permanent in the Icon Definition File.

5 USE OF ICONS IN PTL

After a Picture OBject File is generated, it can be used within a PTL procedure. For this purpose the following commands are added to PTL:

5.1 Display an icon

```
ICON("Mode=Display;Picture-file-name='...'")
```

This command displays the icon picture with the name specified. The default extension is .POB

5.2 Display text and numbers

```
ICON("Mode=Update")
```

With this command the variable text and numbers for the picture will be updated. The text and numbers are specified in the text fields of the icon picture and can be results and other PTL globals.

5.3 Get the key touched

```
ICON("Mode=Get_fkey")
```

This command will get the function key value, as specified in the icon picture, if the screen is touched. If the screen is not touched, a value of "0" will be returned. The function key will be returned in a new import PTL variable Fkey%.

5.4 Wait until a valid key is touched

```
ICON("Mode=Wait_fkey")
```

This command will wait until an operator touches a valid (non zero) function key, as specified in the icon picture. The function key will be returned in a new import PTL variable Fkey%.

5.5 Clear the screen

```
ICON("Mode=Clear-screen")
```

With this command the screen will be erased.

5.6 Example

The following example will display a icon picture "Test". In this picture a measurement is displayed continuously until a valid function key is touched. Then the screen is cleared.

```
SUB Test(Task)
  On error Subret
  Import Fkey%

  Proc("Name=icon_test")

    Icon("Mode=Display;Picture-file-name='Test'")

    Loop
      Meas("Instr=Dmm;At=input")
      Icon("Mode=Update")
      Icon("Mode=Getkey")
    Until Fkey%<>0

    Icon("Mode=Clear-screen")

  End_Proc
SUBEND
```

1 CONTENTS

2 THE PTL NETWORK

2.1 Introduction

This manual describes how to use an RS232 network for the Polar Test Language PTL Version 1.3.

This information is supplied as option :
- PTL/NET PTL Network support
- PTL/SER Network server Software

2.2 Capabilities

In order to be able to share both files on different PTL systems, as well as resources on an IBM-PC/XT/AT compatible computer, PTL is extended to support these requirements through a network.
With an XT as the network fileserver connected to the network, it's printer, floppy disk as well as it's winchester harddisk become available to all PTL systems. Files can be written by one system, and read back to another. Also, one file may be written to by several other PTL systems, in order to build a large database etc. Each PTL user may be 'logged in' to it's dedicated own subdirectory on the harddisk, or all users may share the same directory.

On the fileserver's display screen a windowing based operator interface will allow to display several status pages, as well as to configure the network's transmission characteristics.

The use of the network itself is made transparent to the user by means of additional (network-) devices PC0..PC9. In this manner, all of the flexability of PTL will be maintained completely.

3 OVERVIEW

3.1 Hardware

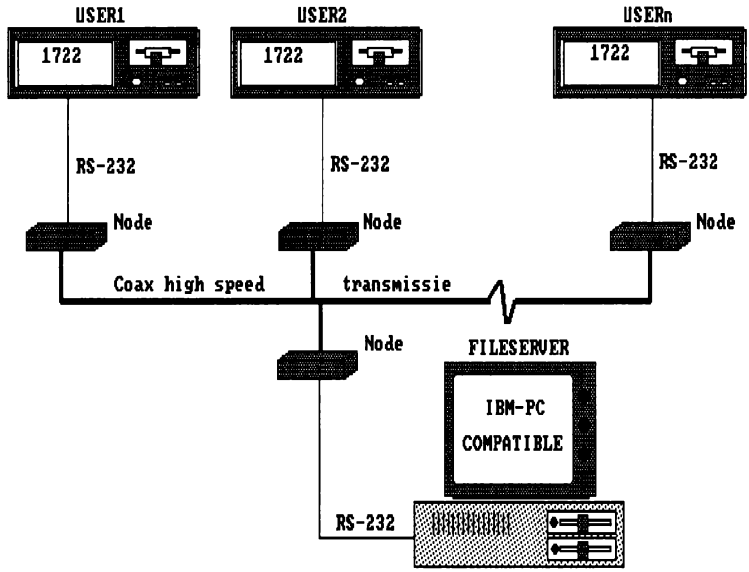


Figure 1

The network itself consists of several intelligent "Clearway" nodes, each with an RS-232 interface to a computer, and a fast, coaxial connection between each node. Maximum transmission speed between two nodes is 98000 Baud. Transmission speed maximum of the RS232 interfaces is 9600 Baud. Each node is known to another node by it's individual address. Connections can be made from one node to another, where the node's software is taking care for buffering of data sent back and forth.

The network consists of several users (PTL systems) and one IBM-compatible network fileserver, connected together through the nodes, as figure 1 demonstrates.

3.2 Software

As far as software concerns, the extensions required to support the network will appear completely transparent in PTL, by means of an additional set of devices PC0..PC9. Each device has a dedicated use in the fileserver, as the table in the following page will show.

PC0:	LPT1:	Printer 1
PC1:	LPT2:	Printer 2
PC2:	COM2:	Serial device 2
PC3:	A:	Floppy A:
PC4:	B:	Floppy B:
PC5:	C:	Winchester C:
PC6:	D:	Substituted drive
PC7:	E:	Substituted drive
PC8:	F:	Substituted drive
PC9:	G:	Substituted drive

PTL Devices versus MSDOS drives table

On the display screen on the MS-DOS compatible computer, the file-server, several status pages will allow an operator to view the status of any of the connected (PTL-) users, as well as the status of the system itself. From this console, some modifications to the networking characteristics are allowed as well.

4 INSTALLATION

4.1 Nodes

Each Clearway node may be configured in software for it's (interface) transmission characteristics. The following protocol is required for the use with the fileserver software:

9600 Baud interface speed
8 databits, No parity, 1 stopbit

Modifications of the protocol characteristics can be done in any terminal emulating software package, such as TTY.FD2 (1722), or PROCOMM.EXE (MS_DOS). When setup correctly, the protocol characteristics may be saved in non-volatile memory.

All users have to be set to a unique address. This address is included in the setup string as follows:

```
ss-9XB9XuuD.POLAR-SLAVE-uu
```

where ss means server (always 99)
uu means user (01..98)

Example: user 03: setupstring =

```
99-9XB9X03D.POLAR-SLAVE-03
```

The fileserver's node should be set to the following configuration:

```
00-9XB9X99Z@POLAR-MASTER-99
```

The commands to setup the node are as follows :

TTY<CR>	Start the terminal emulation
^D	Go to control mode of the node
N	New setup string
No	Answer to prompt
<New characters>	Type until all changes done
Y	Answer to prompt

4.2 1722 setup

Port is KB1:, baudrate = 9600, 8 bit, No parity, 1 stopbit, Stall in- and output disabled, 2 sec timeout.

The startup commandfile should in the 1722 slave should setup the 1722A port for 9600 baud 2 sec timeout, then login in the network with the utility program LOGIN USER01 and then run the MASTER program to select the PTL test programs. At the end of the day, the command LOGOUT will close all open files.

4.3 Cabling

Connection Clearway network to 1722A with rs232 modem cable (straight 1 to 1) and to Server computer to COM1 port via 9 pin to 25 pin (modem) cable.

4.4 Fileserver

The network fileserver should be installed once from the installation floppy disk onto

the winchester disk C: as follows:

After system startup, insert the installation floppy in drive A:, and type at the DOS-prompt the following command:

```
A:INSTALL<CR>
```

Automatically the following directories/ files are being created/ copied:

C:\FSERVER	Network directory
FSERVER.EXE	Network fileserver software
FSERVER.CNF	Configuration file
NETSTART.BAT	Startup batchfile
C:\FSERVER\USER01	User 01 subdirectory
C:\FSERVER\USER02	User 02 subdirectory

.

C:\FSERVER\USER09	User 09 subdirectory

The installation procedure will take a few moments. When the DOS-prompt reappears, installation is finished. At this time, the CONFIG.SYS file in the root-directory has to be edited, to correct the number of allowed open files. The next commands may be added, edited:

FILES=nn
BUFFERS=mm

nn	Number of open files (20)
mm	Number of file-buffers (20)

The number of open files in the file-server is set by file-server startup at the commandline as follows:

```
FSERVER /F:xx /U:y
```

where xx is the total maximum number of files for all users together to be opened at the same time, and y is the maximum number of files for one user to open at a time.

When the commandline argument /F:xx is altered, the configuration file CONFIG.SYS should be edited as well.

NOTE: Whenever the CONFIG.SYS file is edited, the computer should be rebooted to take the changes into effect.

POLAR SYSTEMS

OPTION PTL-NET

PTL Network Support

5 OPERATION FILESERVER

5.1 Startup

A startup batchfile is provided with the system, to automate the startup procedure of the network. The following tasks are executed:

Each networkdevice PC0..PC9 is initialized to a port, device, or path. For instance, the following example may be used:

PTL	Initialization	Device/Path
PC0:	MODE LPT1:132	Printer 1
PC1:	MODE LPT2:80	Printer 2
PC2:	MODE COM2:9600,N,8,1,P	Serial dev 2
PC3:		Floppy A:
PC4:		Floppy B:
PC5:		Winchester C:
PC6:	SUBST D: C:\FSERVER	Sub. drive D:
PC7:	SUBST E: C:\DOS	Sub. drive E:
PC8:	SUBST F: C:\	Sub. drive F:
PC9:	SUBST G: A:\TESTDATA	Sub. drive G:

To activate the bachfile, type at the commandline prompt the following command:

```
NETSTART<CR>
```

and the above mentioned initializations will take place.

The at delivery time included file NETSTART.BAT may be altered to the specific

needs of the application; for instance, the substituted drives may be altered etc. Any ASCII text-editor such as EDLIN.COM can alter the contents of NETSTART.BAT.

After the startup has finished, the main window of the fileserver is presented on the fileserver console. Three possible windows may be accessed from here :

- System status window
  This window contains information about the status of the fileserver;

- User status windows
  These windows contain information about each individual user;

- Configuration window
  Network interface characteristics may be altered in this window.

All windows are updated with data from the PTL networking systems instantaneously, e.g. while selecting the status window of user 1, the window is updated with live data etc.

5.2 System status window

This window may be selected from the main pulldown menu, and contains the following items :

- Transmit Queue Size
  The RS-232 outbound space will be displayed here. Enough space should be available to process commands from the users (>2048 bytes);

- Transmit Status

The last occurred transmit error will be displayed here; this item should be normally 0 (no error). A serial error may occur at small transmit space or any hardware errors.

- Transmit Errors

  This item displays the count of transmit errors. At each error, this item is incremented by one, and should therefore be normally 0.

- Receive Queue Size

  Analogue with the transmit queue space, the receive queue space displays the inbound space. During receival of messages from the PTL systems, the receive space will temporarily decrease, and should be large enough to process a complete message from a user.

- Receive Status

  Whenever a protocol error occurs at receival of messages from a user, this item will contain a non-zero value.

- Receive Errors

  Every time an error occurs at data receival, this errorcount will be incremented by one.

- Last OpCode

  The last received operation code will be displayed in this field.

- Last User

  The user number of the last received message will be displayed in this field.

- Sequence number

  For statistical overview, every time a command is received, this

sequencenumber is incremented by one.

- Usage

  This number represents the number of open filehandles of the fileserver, and therefore represents a usage- indication of the fileserver.

- User messages

  Messages as received from a user are displayed on a binary base in this field.

The system status window may be exited at any time by pressing a key on the keyboard. No interferance will take place with the receival and handling of user-messages because of the low priority of this keyboard task.

5.3 User status window

The user status window represents the status of a specific user. The following fields are available:

- Status

  The status field in the user window represents the login state : either 'Logged In' or 'Logged Out'.

- OpCode

  The last issued command from a user is being displayed in this field.

- Count

  Each time a user is sending a message to the fileserver, this count is

incremented by one. The count is reset at each login command.

- Directory

  The current directory, which a specific PTL-system may use, will be displayed in this field.

- Userdata

  Each command, as received from a user, will be displayed in this field. Since the protocol in use is a binary one, some non-printable ascii characters will be displayed as well as normal 'printable' ascii.

- Replydata

  Each command from a user will imply a specific response from the fileserver to the user in question. This reply message will be displayed here, and since the message is also using a binary protocol, some non-printable ascii characters will be displayed in this field.

- Filename

  At each file-related command, the name of the corresponding file will be displayed in this field.

- Type

  With each file, a filetype is defined as well, such as ASCII, Device LPT1:, Device LPT2: etc. This filetype is displayed in the field: 'Type'.

- Access

  A file's access rights are displayed in this field. The access rights are: 'R' for read only, 'W' for write only, or 'A' for append.

- Handle

Each file-related operation has a corresponding filehandle number, which is displayed in this field.

A user status window may be exited at any time by pressing a key on the keyboard. No interferance will take place with the receival and handling of user-messages because of the low priority of this keyboard task. At this point, another selection may take place.

5.4 Configuration window

In the configuration window, the following fields may be altered:

- Networkport (COM1/COM2)
- Server Network id. (1-99)
- Baudrate serial port
- Parity (N/E/O/S/M)
- Databits (5/6/7/8)
- Stopbits (1/2)
- Transmit buffer (2..60000)
- Receive buffer (2..60000)

Each parameter is affecting the transmission characteristics of the fileserver's node, and mey be altered during operation. The current parameters are displayed in the corresponding fields. By entering the <Ctrl> and <CR>-key at the same time, the menu is exited, and the new parameters will be actualized instantly.

Warning: All messages currently in the receive-queue of the fileserver, will be lost when parameters are modified, so that either one of the parameters should be modified at system startup only.

6 NETWORK UTILITY PROGRAMS

For the communication with the network server a number of 1722 FDOS utility programs can be used.

These programs can be used in a startup commandfile to make contakt with the network server and to setup the right environment (sub directory).

- LOGIN

To log into the network server and to set the work directory to G:USERxx, where xx is the user number (node number).

- CD G:<PATH>

To change the work directory on the network server to G:<PATH>.

- DIR G:<PATH>

To give a directory of the directory G:<PATH>. This directory listing can be as usual in the IBM world with size and date or with a /Q flag in condensed format with only the file names.

- TYPE <NAME>

To see the file <NAME> on the screen. Note that "LIST.PTL" and "PC3:LIST.PTL" are both on the network directory, and "ED0:LIST.PTL" is on the local E-Disk.

- PRINT <NAME>

To print the file <NAME> on the network printer (LPT1:). Note that "LIST.PTL" and "PC5:LIST.PTL" are both on the network directory, and "ED0:LIST.PTL" is on the local E-Disk.

- COPY <NAME1> [<NAME2>]

Will copy <NAME1> to <NAME2>. Note that the commands COPY PC8:LIST.PTL ED0:LIST.PTL and COPY LIST.PTL will both copy the file LIST.PTL from the server to the local E-Disk. (If the second name is not given, the local E-Disk is assumed.)

- DEL <NAME>

Will delete the file <NAME> from the server working directory.

- LOGOUT

Will close the path to the server.

7 WARNING MESSAGES

Each command, or message from a user may introduce a possible error- or warning message.

For PTL-systems, some error or warning messages will not be returned, since PTL itself takes care for the file-server protocol.

The following messages may be returned to indicate an error:

- Bad open argument
The file could not be opened due to a wrong open mode (Not a 'R', 'A', or 'W

- Bad file number, filehandle
The filehandle as sent to the fileserver, is currently not attached to a file. Open the file before usage.

- Bad file type
The file specified has a non-valid type. Legal types are: ASCII, Device LPT1:, Device LPT2: or Device COM2:

- Bad open mode
The file related to the command has a wrong open-mode, such as an attempt to write to a file, opened for read only etc.

- Bad directory
The directory as specified in a command does not exist.

- Device file could not be opened
The device (spool-)file as specified (LPT1:, LPT2: or COM2:) could not be opened,

due to a harddisk error.

- EOF
An attempt was made to read from a file, after it has encountered the end of file mark.

- Error writing data, Write error
A write to a file could not be executed, due to file-error(s).

- Error reading from file, Read error
A read from file could not be performed, due to file- error(s).

- Error in logout procedure
An error occurred during the logout procedure, at which time all opened files are closed.

- Error while printing
An MS-DOS error occurred during printing to a specified print-device, such as LPT1:, LPT2:, COM2:. Check the printer connected.

- File not found
The file(s) searched for in a directory related command could not be found.

- File could not be closed
The file specified could not be closed. Possibly the file was not open, or a DOS error occurred during closing.

- File could not be deleted
The file specified could not be deleted. The file could be write-protected, or not-existing.

- File could not be opened

The file in question could not be opened. Several causes can create this message, such as disk full, bad filename etc.

- No more free filehandles left

An attempt was made to open more files than the file-server can handle. See installation chapter on the number of open files.

- Too many files opened

A specific user is not allowed to open more files than specified at startup. (See installation). An attempt was made to open more files at a time.

- Unable to create TMP file

A tempory filename could not be opened, so a spool-file could not be created. This error should normally not occur.

- User path allowed with G: only !

The command Changedirectory (CD), or Login are used without the fixed prefix "G:".