

This submitted manuscript has been authored by a contractor of the U. S. Government under contract No. W-31-109-ENG-38. Accordingly, the U. S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U. S. Government purposes.

LS Note No. 190

Development of General Purpose Data Acquisition Shell (GPDAS)

Y. Chung and K. Kim

Advanced Photon Source
Argonne National Laboratory
Argonne, IL 60439

Table of Contents

1.	Introduction	1
2.	Configuration	2
3.	Starting and Programming in GPDAS	2
4.	Constants and Variables	4
5.	System Global Variables	6
6.	Alias Definition	6
7.	History Mechanism	7
8.	Show and Save	7
9.	Input and Output	8
10.	Operators and Functions	8
11.	Subroutines	8
12.	Command Flow Control	9
13.	Graphics	10
14.	Windows	11
15.	GPIB and Device Drivers	11
16.	Help Facility	12

Appendices

A.	GPDAS.CFG (GPDAS Configuration File)	14
B.	GPDAS Shell Command Syntax	16
C.	Special Characters and System Global Variables	21
D.	I/O Command Syntax	22
E.	Operator Syntax	23
F.	Graphics Command Syntax	26
G.	Window Command Syntax	27
H.	GPIB Command Syntax	28
I.	Device Driver Command Syntax	29
J.	Script Example SORTCAP.SCR	32
K.	Script Example RADTRIG.SCR	41

1. Introduction

GPDAS (General Purpose Data Acquisition Shell) has grown out of a need for a command interpreter and a programming environment designed to handle streamlined data acquisition and analysis in a flexible laboratory measurement setting. Although most operating systems on various platforms provide a command interpreter and certain levels of programmability, there are none specifically designed for a laboratory environment dedicated to data acquisition, analysis, and device control that requires command flow control, communication with external devices, and storage and archiving of acquired data.

Usually a dedicated, stand-alone application, though limited in its scope of capabilities, would be developed for such purposes. The drawback of this approach, however, is that such applications provide little flexibility to adapt to different laboratory environments or different configurations of devices, especially in the R&D phase which calls for frequent changes in the measurement procedures.

An interpreter-based shell can be a very convenient tool for device control and data acquisition through the user command input as well as via script files containing several commands. By incorporating the native operating system, such as UNIX or MS-DOS, it can take advantage of existing utility programs for file manipulation and data analysis. Hardware-level debugging and script programming can also be facilitated by utilizing interactive mode, aliases, variables, and history mechanisms. In consideration of the interpreter's relatively slow execution time, some routines requiring high speed can be separated as module programs, or utilities, and called from scripts. Modularization of an application into several small utilities is also a good programming strategy since they are easy to debug, don't take up much memory space, and can be used in other contexts.

The current version of GPDAS runs on MS-DOS-based microcomputers. The software package consists of a shell, utilities, and script programs. The computer system for magnet measurement at APS, for instance, currently uses GPDAS and consists of a Compaq 386/20e microcomputer equipped with 4 MB RAM, a 40 MB hard drive, a 44 MB removable hard drive, a 5.25" 1.2 MB floppy drive, a 3.5" 1.44 MB floppy drive, and a VGA monitor. The system is linked to other computers through Ethernet for data transfer and communication. Text and PostScript graphics files are downloaded to laser printers using an AppleTalk connection.

The shell can be directly interfaced with PC boards in order to control the data acquisition devices. In conjunction with the magnet measurement system, software drivers for IEEE-488, a multifunction board, and an ADC/DAC are built into the shell. The shell can be interfaced to other types of boards through source level modification. Developed with such possibilities in mind, the internal structure of the shell is highly modularized to facilitate such changes.

This note is intended as an abbreviated introduction to the concept and the structure of GPDAS and assumes the reader has a certain level of familiarity with programming in general. The structure of the following sections consists of brief explanations of the concepts and commands of GPDAS, followed by several examples. Some of these are tabulated in the appendices at the end of this note.

2 Configuration

When started, GPDAS reads in several setup parameters from the configuration file. The default configuration file has the same name and is in the same directory as the shell application, with the file extension ".CFG". The syntax of this file and the meaning of each parameter are shown in Appendix A. Default values are used for unspecified parameters. If the file cannot be located in the present working directory, default values are used for the parameters that are essential in starting up the shell. The configuration parameters for the device drivers, e.g., GPIB and ATMIO, are read in the first time those drivers are called.

3 Starting and Programming in GPDAS

The shell is started by typing in the name of the shell program with extension ".EXE" at the DOS prompt. It takes the configuration file name as the only optional argument preceded by "-h".

Ex. 3.1: Starting up the GPDAS shell at DOS prompt.

```
C:\> mm          ! name of the shell program is mm.exe and the
                  ! configuration file name is mm.cfg.

C:\> diag -h gpdas.cfg  ! name of the shell program is
                        ! diag.exe and the configuration
                        ! file name is gpdas.cfg.
```

The shell can be programmed to perform desired tasks either interactively or non-interactively. In the interactive mode commands are entered through the keyboard, while in non-interactive mode the shell reads commands from a script and executes them. The user interface of the interactive mode is very similar to that of the native operating system, MS-DOS.

The syntax of the commands must conform to DASL (Data Acquisition Shell Language), a high-level programming language developed for GPDAS. Commands not understood by the shell are passed to MS-DOS for further processing. These shell commands are listed in Appendix B. Script programming is easier than developing new applications; therefore, debugging of hardware and testing of measurement procedures are simplified.

Ex. 3.2: Sample commands at the shell prompt level.

```
GP-C:\DAS_1> dir/w
      .
      .
GP-C:\DAS_2> for #i = 0, 10, 1 {
              #j = sin (#i * 3.14 / 10);
              write line to screen #i #j;
              }
      .
      .
GP-C:\DAS_3>
```

A script should begin with the "procedure" command followed by the script name and a left curly bracket. This signifies the beginning of the script. The left curly bracket must be matched by a right bracket at the end of the script, which is the implicit end of the script. The "end" command explicitly terminates the script execution. Each command in a script must end with a semicolon (;). A line that begins with an exclamation mark (!) is a comment line and is ignored. If a line starts with an exclamation mark and an asterisk (!*), all the following characters up to "!" are ignored. An example script is shown in Ex. 3.3.

Ex. 3.3: A sample script.

```
procedure sample.scr
{
!*  A sample script displaying numbers from 0 to N.
```

```

*   Syntax:
*       run sample;
*!

define ws write line to screen;
define NN 10;

!   Begin the loop.

for #i = 0, NN, 1 {
    ws "The number is" #i;
}
end;
}

```

Design objectives of the shell include:

- Interactive mode for user command input
- Non-interactive mode for script execution
- Definition of command aliases
- History of commands
- Assignment of variables and arrays (integer, long, float, double, and string) in RAM and virtual memory
- Backup of aliases, history, and variables in script format
- Mathematical, logical, and text string operations
- Input and output
- Command flow control
- Device interface
- Windowing capability
- Graphics capability

Other utility applications have also been developed as integral parts of the software package. These include: PostScript graphics, graphical monitoring, file management, background printing, and device control.

4. Constants and Variables

There are five types of constants and variables: integer, long integer, float, double, and string, as shown in Table 4.1. Integer constants range from $-32,768$ (-2^{15}) to $32,767$ ($2^{15} - 1$), while long integers range from $-2,147,483,648$ (-2^{31}) to $2,147,483,647$ ($2^{31} - 1$). Double type constants are distinguished from the float type constants by the presence of the exponent ('e') character.

The string constants are surrounded by two double quotes ("). They are character arrays of user-definable length (256 by default) and are terminated by a null character (hexadecimal 0). Some special characters, such as the double quote (") and carriage return, can be embedded in a string constant when prefaced by the back-single-quote (`) character. These special characters are listed in Appendix C. String constants can contain any ASCII codes (0x00 - 0xFF) preceded by the tilde (~) character. Examples are shown in Ex. 4.2.

Explicit declaration of variables is optional, and the types of variables that appear without prior declarations are determined as follows: variables that start with characters 'i' through 'n' are long integers; all others are doubles by default. The declaration of variables can be placed anywhere, not only at the beginning of the shell or scripts.

Variables should be preceded by a '#' character, and assignment of values to variables is done through the equal (=) sign. Arrays of variables can be declared by specifying the dimensions in square brackets ([]). The contents of variables may be stored either in RAM (random access memory) or in the virtual memory file on disk to save main memory (i.e. RAM) space, especially when allocating arrays of large size. The default location is specified in the configuration file. To override the default, either "/v" (for virtual memory) or "/m" (for main memory) may be specified in the allocation command. Binary data files can be loaded into variables by including a "/l" option in the allocation command as shown in Ex. 4.1.

Ex. 4.1: Declaration and initialization of variables.

```
float #k, #x = 100.0;
float/m #coeff[5] = {0.1, 0.2, 0.3};           ! allocated in RAM
float/v #data[1000];                          ! allocated in
                                                ! virtual memory
float/l/v #olddata = olddata.vmf;            ! load the data in
                                                ! a binary file
```

Type	Size (bytes)	Examples
Integer	2	0, 1, -10, 32767, -32768
Long integer	4	1, -10, 2147483647
Float	4	1.0, 3.5, -100.2
Double	8	1e+20, -3.4e-15
String	user-defined	"This is a string.", "\"ABC\"", "A\tB\tC"

Table 4.1: Valid constant types in GPDAS.

Ex. 4.2: String constants.

```
write line to screen "~07";           ! sound a beep.
write line to screen "~C4~C4~C4~C4~C4~C4~C4~C4~C4~C4";
                                           ! draw a line.
```

5. System Global Variables

System global variables are declared at the beginning of the shell startup and become accessible to the shell and all scripts. Currently, there are 10 such variables, as listed in Appendix C.

Ex. 5.1: Use of the system global variables.

```
gpib read file from hpna to #vmdir || hpdat.vmf;
float/l/v #hpdat = hpdat.vmf;
```

6. Alias Definition

Commands and parts thereof may be defined as single-word aliases to simplify the interactive command input and script programming. When a predefined alias is encountered in a command string, the interpreter expands the alias and substitutes the arguments, if there are any. No space is allowed in the alias in the "define" command. In case of multiple commands, the aliases must be surrounded by curly brackets ({}), which are stripped away before being stored in the alias file.

Ex. 6.1: Defining aliases.

```
define nn 100;
define ws write line to screen from;
define grna gpib write line to hpna from;
define ad(a,b) ((a)+(b));
define bpptest(file,mseq) run bpm -f file -m mseq;
define forloop {
    for #i = 0, nn, 1 {
        write line to screen #i;
    }
}
```

Command aliases can also be assigned to the functions keys (F1 - F10) by using the "defkey" command. Pressing one of the functions keys will display the command string and prompt for execution or further modification.

Ex. 6.2: Function key assignment.

```
defkey f1 dir/w;  
defkey f2 {copy *.c b:\; copy *.h b:\; copy *.obj b:\}
```

7. History Mechanism

All user commands entered at prompts are recorded in series in a history file for future retrieval and navigation through the commands. These commands are retrieved by using the up key (↑), the down key (↓), and the exclamation mark (!). Pressing the up key displays the command previous to the current one, and pressing the down key displays the command after the current one in a cyclic manner. The exclamation mark (!) is used to recall the n-th command recorded in the history file.

Ex. 7.1: Retrieving a past command.

```
!10<cr>    ! will display the 10-th command since startup.
```

8. Show and Save

Variables, aliases, and history can be displayed and saved to files for later use with the "show" and "save" commands. Wild characters '*' and '?' are used to specify several variables, aliases, and previous commands. Data can be saved to binary files using the "/b" option. When saving variables, it is important that the name of the variable to be saved is at the end of the command.

Ex. 8.1: Showing and saving variables, aliases and history.

```
int #i, #k;  
double #x[100], #y[100], #cx[100], #cy[100];  
show variable #k, #x[*];  
save variable to var.scr #k #x;  
save /b variable to cx.vmf #cx;  
save /b variable to cy.vmf #cy;  
show variable #c*;           ! show variables starting with #c.  
save variable to varc.scr #c*; ! save variables starting  
                             ! with #c. .  
show define do*;           ! show aliases starting with do.  
save define to aliases.scr *; ! save all aliases.  
show history 1?;          ! show commands numbered between 10 and  
                          ! 19.
```

9. Input and Output

The native operating system, such as MS-DOS, provides input/output (I/O) capabilities to peripheral devices and the built-in device drivers provide I/O capabilities to external devices as explained in Section 15. In addition to these, several commands can be used to create, open, read from, and write to disk files and the console. The relevant commands are: "open", "close", "read", "write", and "file". The command "file" has two subcommands: "find" and "go". The syntax for these commands are tabulated in Appendix D.

Ex. 9.1: I/O to disk file and the console.

```
open/n test.txt;
write line to test.txt "This is a test string.";
close test.txt;
write line to screen from "This is a test string.";
```

10. Operators and Functions

There are several operators and functions that are built into the shell. These operate on numerical and logical entities and strings. The list of built-in operators and functions is given in Appendix E.

Ex. 10.1: Operators and functions.

```
#x = 1.0;
#y = sin (#x + sin (#x));
int #ia[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
#std = arrstdev (#ia);
```

11. Subroutines

Subroutines are separated from the main procedure of the script by the "subroutine" heading, and the "call" command invokes the execution of a subroutine. A subroutine begins with a left curly bracket ({} and ends with a right curly bracket (}). Since the aliases and variables are used globally among the main procedure and the subroutines, no arguments are passed between the calling routine and the one called. The "return" command simply makes the script execution resume at the point where

the subroutine was called. If there is no return command, the subroutine returns when the end bracket is encountered.

Ex. 11.1: Calling a subroutine.

```
call test;
      .
      .

subroutine test
{
    write line to screen "This is a test.";
    return;
}
```

12. Command Flow Control

There are seven commands for command flow control (if, elseif, else, do, for, while, and goto). These commands must be immediately followed by a curly bracket ({} and a matching bracket (}) at the end. They can also be nested within each other to an indefinite degree.

If, elseif, and else are conditional flow control commands. If the expression in the parentheses is non-zero, the commands surrounded by the curly brackets are executed; otherwise, the following command is executed.

Ex. 12.1: Conditional flow control commands: if, elseif and else

```
#i = 100;
if (#i .lt. 10) {
    write line to screen "#i is less than 10.";
}
elseif (#i .lt. 100) {
    write line to screen "#i is less than 100.";
}
else {
    write line to screen "#i is greater than or equal to
100.";
}
```

For an unconditional jump to a labeled location in a script, the "goto" command can be used. A label is identified by a colon (:) after the label name. Script execution cannot jump across nested loops using the "goto" command.

Ex. 12.2: Unconditional flow control command: goto

```
if (#i) {
    goto exit;
}
.
.
exit :
return;
```

Other useful flow control commands are: do, while, and for. The "do" command is matched by a "while" condition statement at the end of the loop. After the commands within the curly brackets are executed, the "while" condition is tested. If it is true, the script execution restarts at the beginning of the loop. Otherwise, it continues with the next command. Therefore, the "do" loop is executed at least once. For the "while" command, the condition is tested first, and the execution continues with the loop or skips the entire loop depending on the result. In case of the "for" command, the initial and final values and the increment are given at the beginning of the loop. When the value of the variable goes outside the range specified by the initial and final values, the loop execution terminates.

Ex. 12.3: Flow control commands: do, while and for

```
#i = 0;
do {
    write line to screen #i;
    inc #i;
} while (#i .lt. 10);

while (#i .gt. 0) {
    write line to screen #i;
    dec #i;
}

for #i = 0, 10, 1 {
    write line to screen #i;
}
```

13. Graphics

In order to accomplish simple graphical tasks, a set of graphics commands was included in the shell. Once the shell is put in the graphics mode, the "graf" command can draw various kinds of graphical entities on the screen. This command is ignored if

the shell is in the text mode. The "graf" command is followed by various subcommands. The syntax for the graphics commands is listed in Appendix F.

Ex. 13.1: Graphics-related commands.

```
graf setorigin 0 0;  
graf moveto 100 100;  
graf lineto 200 200;
```

14. Windows

Text strings, including all printable ASCII characters, can be drawn on a window using the window-related commands. The colors of the window background and the text can be assigned for each output. The "create" subcommand creates a window but does not display it on the screen. The window is displayed with the "show" subcommand. Each window has a unique title which must be specified for each command to direct the text output to the proper destination window. The syntax for the window-related commands is listed in Appendix G.

Ex. 14.1: Window-related commands.

```
window create win1 0 1 14 30 10 80 20;  
window write win1 2 2 50 3 "This is win1.";  
window show win1;  
.  
window delete *;      ! deletes all windows.
```

15. GPIB and Device Drivers

For communication with external devices, there are four drivers built in the shell: GPIB, ATMIO, ATDIO and NEAT. The corresponding commands are "gpib", "atmio", "atdio", and "neat", respectively. GPIB (General Purpose Interface Bus) is a popular parallel bus architecture for data acquisition, ATMIO is a multifunction board with A/D and D/A converters with analog I/O, and ATDIO provides high-speed 32-bit parallel digital I/O interface. These boards are manufactured by National Instruments. NEAT is a stepper motor driver board manufactured by New England Affiliated Technologies.

The driver commands are followed by various subcommands, such as "read" and "write", as tabulated in Appendices H and I. Device drivers may be added to or deleted from the shell with minor modifications to the initialization routines in the source code.

Even though the device drivers are integral parts of the shell, each has a separate and independent command syntax, as defined by its programmer. In some cases, it is desirable not to evaluate the variables used in the device command, e.g, when reading a string from an external device and storing it in a string variable. By putting an ampersand (&) in front of the variable, the interpreter does not evaluate the variable when parsing the command.

The "gpib" command is used for I/O and device control for external GPIB devices. Several I/O options allow downloading and uploading data either to a string variable ("line") or to a file ("file" and "ffile"). The command syntax and the functions of subcommands are tabulated in Appendix H.

Ex. 15.1: GPIB commands.

```
define gwna      gpib write line to hpna;
define grna      gpib read line from hpna;
define gfna      gpib read file from hpna;
gwna "idn?";
grna #str;      ! #str is a variable of string type.
gwna "form5";
gwna "outpform";
gfna d:\hpdat.vmf upto 0;
```

16. Help Facility

Extensive on-line help is available to reference the command syntax and other features of GPDAS. The help information, contained in a text file with a user-definable name ("GPDAS.HLP" by default), must conform to the following format. A main topic, usually a command, is preceded by a dollar sign (\$). A subtopic is then preceded by a double dollar sign (\$\$), a subsubtopic is preceded by a triple dollar sign (\$\$\$), and so on. This facilitates updating the on-line help information.

Ex. 16.1: Help file format.

```
$ DEFINE alias[(arg1, arg2, ... , argn)] expr
  Defines alias to string 'expr'.
$$ Examples
  define cp copy;
  define cpall {
```

```

        copy *.c b:\;
        copy *.h b:\;
    }
define add(a,b) ((a)+(b));

$ GPIB
    Invokes the GPIB function.
$$ abort
    gpib abort dev
    Make a GPIB device stop listening.
$$$ Example
    gpib abort hpna;

```

From Ex. 16.1, the command "help define" will display everything from "DEFINE" up to the double dollar sign followed by a list of available subtopics. Typing in "help gpib abort" will display everything from "abort" up to the triple dollar sign, followed by a list of available subsubtopics.

Listings of sample scripts SORTCAP.SCR and RADTRIG.SCR are in Appendices J and K, respectively.

Appendix A. GPDAS.CFG (GPDAS Configuration File)

```
prompt = "GP-{pwd}_{mem}_{cnum}> " ! prompt
                                     ! {pwd} : present working directory
                                     ! {mem} : memory available
                                     ! {cnum}: command number
login = "c:\das\mms\login.scr" ! name of the login file
logout = "" ! name of the logout file
help = "c:\das\mms\gpdas.hlp" ! name of the help file
history = "gpdas.hst" ! name of the history file
vmmdir = "f:\dasvm" ! directory for virtual files
vmfile = "f:\dasvm\vm.dat" ! virtual file for variables
alfile = "f:\dasvm\al.dat" ! virtual file for aliases
dos = "c:\command /e:512 /c " ! dos command
memory = "virtual" ! default memory for variables

string = 256 ! length of strings in bytes
buffer = 1024 ! length of the command buffer
command = 512 ! maximum length of a command
stack = 4 ! size of the operand stack
shell_alias = 128 ! size of the shell alias block
scr_alias = 32 ! size of the script alias block
variable = 32 ! size of the variable block
nhistory = 64 ! number of commands in history
vmsize = 512000 ! size of the variables virtual file
alsize = 512000 ! size of the aliases virtual file
window = 8 ! maximum number of windows

cflow = 8 ! depth of the command flow control
label = 32 ! size of the label allocation block
subroutine = 16 ! size of the subroutine allocation
block

board = gpib

    device = gpib0
    addr = 0
    eos = 10
    timeout = 10

    device = tesla
    addr = 1
    eos = 10 ! = '\n'. Do not use '\r'.
    timeout = 10

board = atmio

    channel = 2
    voltmax = 10.
    voltmin = -10.
    scalemax = 1000.
```

```
scalemin = 900.
```

```
channel = 3
```

```
voltmax = 10.
```

```
voltmin = -10.
```

```
scalemax = 1000.
```

```
scalemin = 500.
```

```
board = atdio
```

```
device = PSCON
```

```
addr = 0x14
```

```
device = ADCON
```

```
addr = 0x18
```

Appendix B.

GPDAS Shell Command Syntax

Commands	Syntax	Function
?	? command	Substitute all applicable aliases and evaluate all expressions, but do not execute the command.
:	label :	Designates a label position for use with 'goto' command.
=	variable[index] = value	Sets the value of a variable[index].
add	add variable [by] expr	add 'expr' to 'variable'.
atdio	atdio subcommand ...	atdio device command. See Appendix I.
atmio	atmio subcommand ...	atmio device command. See Appendix I.
bkp	bkp	Set a breakpoint in a script file for debugging.
break	break	Break out of the loop while executing the script.
call	call subroutine	Call a subroutine in the current script.
close	close filename	Close a file already open for I/O.
command	command expr	Invokes the GPDAS command interpreter. 'expr' is evaluated before execution.
continue	continue	Go to the beginning of the loop and continue.
countl	countl [on off]	Turn line counting on or off. If the line counting is on, screen output will pause after each screenful of output.
deassign	deassign [define defkey variable] arg1, arg2, ... , argn	Remove currently allocated aliases or variables.
debug	debug [on off]	Turn on or off the feature of debugging the internal stack operation.
dec	dec variable	Decrease the value of a 'variable' by 1.
define	define alias[(arg1, arg2, ... , argn)] expr	Defines alias to string 'expr'. 'expr' may be multiple commands, in which case 'expr' must be surrounded by curly brackets ({}).
defkey	defkey fkey expr	Assign a command to a function key (F1 - F10).

Commands	Syntax	Function
div	div variable [by] expr	Divide 'variable' with 'expr'.
do	do { ... } while (expr)	Execute the commands in { ... } while 'expr' is non-zero.
dos	dos expr	Invokes the DOS command interpreter. 'expr' is evaluated before execution.
double	double [/v/m] variable [dimen] [= { ... }]; double/l[/v/m] variable = virtual_filename	Allocate double variables. All variables are initialized to 0 unless specified. If the variable is an array, only the unspecified elements will be initialized to 0.
dup	dup [on off]	Turn the duplication mode on/off. The duplicated command shows the interpreted command on the screen.
if, elseif, else	if (expr1) { .. } elseif (expr2) { .. } else (expr3) { .. }	Conditional command flow control. The commands in { ... } will be executed if the conditional statement is true.
end	end	Terminate the execution of the script and return to the calling script or the shell.
file	file subcommand	Operate on file position pointers. See Appendix D.
float	float [/v/m] variable [dimen] [= { ... }]; float/l[/v/m] variable = virtual_filename	Allocate float variables. All variables are initialized to 0 unless specified. If the variable is an array, only the unspecified elements will be initialized to 0.
for	for var = init_expr, fin_expr, inc_expr { ... }	Execute the commands in { ... } at an incremental step of 'inc_expr' for 'var', starting with 'init_expr'.
goto	goto label	Unconditional command flow control. Will jump to the specified label unconditionally.
gpib	gpib subcommand	Invokes the GPIB function. See Appendix H.
graf	graf subcommand	Invokes several graphics-related commands. See Appendix F.
help	help [command_name function_name]	Display help information about commands or built-in functions.
inc	inc variable	Increase the value of 'variable' by 1.

Commands	Syntax	Function
int	int [/v/m] variable [dimen] [= { ... }]; int/l[/v/m] variable = virtual_filename	Allocate integer variables. All variables are initialized to 0 unless specified. If the variable is an array, only the unspecified elements will be initialized to 0.
long	long [/v/m] variable [dimen] [= { ... }]; long/l[/v/m] variable = virtual_filename	Allocate long variables. All variables are initialized to 0 unless specified. If the variable is an array, only the unspecified elements will be initialized to 0.
monitor	monitor [on off]	Turn the device monitor on or off.
mul	mul variable [by] expr	Multiply 'expr' to 'variable'.
neat	neat subcommand	Invokes the NEAT function. See Appendix I.
open	open [/a/b/n] filename	Open a file for I/O. See Appendix D.
pause	pause [option]	Pause execution of a script. With option = 1 or none a message will appear on the screen for prompt.
pickopt	pickopt opt dest_var [accept_var]	Pick up the option string from the run command. The optional variable accept_var is set to 1 if opt was in the command.
procedure	procedure proc_name	Signifies the beginning of the script.
read	read [word line] from file_name [to] variable	Read a word or a line from the console or a file. See Appendix D.
return	return	Return from the current subroutine to the calling subroutine.
run	run [/i/d/o] script	Execute a script. The script file has an extension ".SCR" by default. /i: Include the script in the current script or the shell. The aliases and the variables are shared between the calling and the called scripts. /d: Run in the debugging mode. /o: Run the compiled script.

Commands	Syntax	Function
save	save [/a/b/c/s/t] [define history variable] [to] dest srce	<p>Save the currently defined aliases, history, or variables ('srce') to the file 'dest'.</p> <p>Options:</p> <ul style="list-style-type: none"> /a: The output will be appended to an existing file. If the file does not exist, a new one will be created. /b: The output will be in binary form. Applies when saving variables. /c: No linefeed character will be added after each element when saving variables. /s: Displays only the names of aliases, variables, and function keys assigned. /t: The output will be in text form. Applies when saving variables.
show	show [/s] [define defkey variable] [v]	<p>Shows the content of 'v'.</p> <p>Option:</p> <ul style="list-style-type: none"> /s: Displays only the names of aliases, variables, and function keys assigned.
string	string [/v/m] variable [dimen] [= { ... }]; string/l[/v/m] variable = virtual_filename	<p>Allocate string variables. All variables are initialized to null unless specified. If the variable is an array, only the unspecified elements will be initialized to null.</p>
sub	sub variable [by] expr	Subtract 'expr' from 'variable'.
subroutine	subroutine sub_name	Signifies the beginning of a subroutine.
video	video [graphics text]	Switch between the graphics and the text mode for screen output.
wait	wait [until for to] time	<p>Pause the execution of a script and wait</p> <p>Options:</p> <ul style="list-style-type: none"> until: will wait until the specified datetime. 'time' is in the form "h:m:s m/d/y". for: wait for the specified time in seconds starting now. to: wait for the specified time in seconds, starting from the beginning of the script.

Commands	Syntax	Function
while	while (expr) { ... }	Execute the commands in { ... } while 'expr' is non-zero.
window	window subcommand ...	Invokes several window-related commands. See Appendix G.
write	write [char word line file] [to] dest [from] srce	Output the string in srce to 'dest', which is the destination file name. See Appendix D.

Table B.1: List of the GPDAS shell commands.

Appendix C. Special Characters and System Global Variables

Character	Contents
<code>`0</code>	Null character (NULL)
<code>`b</code>	Backspace (BS)
<code>`f</code>	Form feed (FF).
<code>`n</code>	Carriage return (CR).
<code>`r</code>	Line feed (LF).
<code>`t</code>	Horizontal tab (TAB)
<code>`x</code>	same as x. x is any character other than above.

Table C.1: List of special characters.

Name	Type	Contents
<code>#\$alfile</code>	string	Name of the file containing aliases definitions.
<code>#\$helpfile</code>	string	Name of the help file.
<code>#\$historyfile</code>	string	Name of the history file.
<code>#\$loginfile</code>	string	Name of the login file.
<code>#\$logoutfile</code>	string	Name of the logout file.
<code>#\$maxstrsize</code>	integer	Size of string variables and constants.
<code>#\$maxwindow</code>	integer	Maximum number of windows that can be created.
<code>#\$memvirtual</code>	integer	Default location of variables and aliases. 1 if disk, 0 if RAM.
<code>#\$vmmdir</code>	string	Directory name holding virtual memory files with extensions ".VMX", (X = I, L,F,D,S).
<code>#\$vmfile</code>	string	Name of the file containing virtual variables.

Table C.2: List of system global variables.

Appendix D. I/O Command Syntax

Commands	Syntax	Function
open	open [/a/b/n] filename	Open a file for I/O. Options: /a: append mode. The subsequent I/O will be directed to the end of the file. /b: binary mode. The file will open in binary mode. /n: new file. If a file of the same name already exists, it will be deleted and a new file will be created.
read	read [word line] from file_name [to] variable	Read a word or a line from the console or a file. 'file_name' can be "screen" or the name of a file or the string variable for file name. It is important that the name of the variable is at the end of the command.
write	write [char word line file] [to] dest [from] srce	Output the string in file 'srce' to 'dest', which is the destination file name. If 'dest' is "screen", the output will be directed to the standard I/O. Otherwise, 'dest' must be the name of the file that has been already opened by the 'open' command.
file find	file find dest [in] srce	Find string 'dest' in a file 'srce' and reposition the file pointer.
file go	file go to [beginning current end] [off foff] [in] srce	Go to the specified offset location in file 'srce'.

Table D.1: The commands for I/O to disk files and the console. The file name for the console is "screen". The characters in square brackets are optional.

Appendix E. Operator Syntax

Operator	Type	Function and Example
a+b	-	arithmetic addition; #x = #y + 2
a-b	-	arithmetic subtraction; #x = #y - 2
a*b	-	arithmetic multiplication; #x = #y * 2
a/b	-	arithmetic division; #x = #y / 2
a%b	-	modulus (remainder after division); #x = #y % 2
a^b	-	power; #x = #y ^ 0.5
.not. a	integer	logical negation; if (.not. #i)
a .lt. b	integer	true if a is less than b; if (#x .lt. 2)
a .le. b	integer	true if a is less than or equal to b; if (#x .le. 2)
a .gt. b	integer	true if a is greater than b; if (#x .gt. 2)
a .ge. b	integer	true if a is greater than or equal to b; if (#x .ge. 2)
a .eq. b	integer	true if a is equal to b; if (#s .eq. "this")
a .ne. b	integer	true if a is not equal to b; if (#s .ne. "this")
a .and. b	integer	true if both a and b are true; if (#i .and. #j)
a .or. b	integer	true if either a or b is true; if (#i .or. #j)
int (a)	integer	type cast to integer; #x = int (2.4);
long (a)	long	type cast to long integer; #x = long (2.4);
float (a)	float	type cast to float; #x = float (2);
double (a)	double	type cast to double float; #x = double (2);
string (a)	string	type cast to string; #x = string (2);
capital (a)	string	change to upper case; if (capital (#x) .eq. "YES")
btoi (a)	integer	convert a binary string to integer; #x = btoi ("0b1010");
itob (a)	string	convert an integer to a binary string; #x = itob (10);
htoi (a)	integer	convert a hexadecimal string to integer; #x = htoi ("0x0a");
itoh (a)	string	convert an integer to a hexadecimal string; #x = itoh (10);
picknum (s,n)	-	pick the n-th number in the string s; #x = picknum (#s, 2);
pickchar (s,n)	string	pick the n-th character in the string s; #x = pickchar (#s, 2);
pickword (s,n)	string	pick the n-th word in the string s; #x = pickword (#s, 2);
format (s, a1,a2, ..., an)	string	format a string using the format specification 's' with arguments a1, a2, ..., an.

Operator	Type	Function and Example
strsize (s)	integer	length of the string s in bytes; #l = strsize (#s);
fexist (a)	integer	true if a file of name "a" exists in the current directory; if (fexist ("data.dat"))
fsize (a)	long	the size of a file "a" in bytes.; #flen = fsize ("data.dat");
diskspace (a)	long	the available disk space in the drive a in bytes.; #dSPACE = diskspace ("c");
datetime (a)	string	a = 0; the current time and date
	long	a = 1; number of seconds since startup in long integer
	double	a = 2; number of seconds since startup in double
	long	a = 3; number of milliseconds since startup in long integer
memory (a)	long	a = 0; the size of available memory in bytes
min (a,b)	-	the smaller of a and b; #x = min (#y, #z);
max (a,b)	-	the larger of a and b; #x = max (#y, #z);
abs (a)	-	the absolute value of a; #x = abs (-10);
sqrt (a)	double	the square root of a; #x = sqrt (10);
sin (a)	double	the sine value of a in radian; #x = sin (2.0);
cos (a)	double	the cosine value of a in radian; #x = cos (2.0);
tan (a)	double	the tangent value of a in radian; #x = tan (2.0);
asin (a)	double	the arc sine value of a; #x = asin (0.5);
acos (a)	double	the arc cosine value of a; #x = acos (0.5);
atan (a)	double	the arc tangent value of a; #x = atan (0.5);
atan2 (a,b)	double	same as atan (a/b); #x = atan (-1, 2);
log (a)	double	the natural logarithm of a; #x = log (2);
exp (a)	double	the exponential of a; #x = exp (2);
rand (a)	double	the random number generator of max a; #x = rand (1);
sign (a)	integer	the sign of the number a; #x = sign (-100);
arrdim (a)	long	the dimension of the array a; #x = arrdim (#y);
arrmax (a)	-	the maximum of the array a; #x = arrmax (#y);
arrmin (a)	-	the minimum of the array a; #x = arrmin (#y);
arrmean (a)	double	the mean value of the array a; #x = arrmean (#y);
arrstdev (a)	double	the standard deviation of the array a; #x = arrstdev (#y);
arrsum (a)	double	the sum of the elements of the array a; #x = arrsum (#y);

Operator	Type	Function and Example
arrsum2 (a)	double	the sum of the square of the elements; #x = arrsum2 (#y);
arrdot (a,b)	double	the dot product of the arrays a and b; #x = arrdot (#y,#z);

Table E.1: List of operators and functions built into the shell. The type of the return value depends upon the arguments if it is not specified.

Appendix F.

Graphics Command Syntax

Commands	Syntax	Function
arc	graf arc radius angle1 angle2	Draw an arc with radius from angle1 to angle2.
clear	graf clear	Clear the graphics screen.
lineto	graf lineto h v	Draw a line to (h,v).
moveto	graf moveto h v	Move to (h,v).
putpixel	graf putpixel h v color	Put a pixel of color at (h,v).
restore	graf restore	Restore the graphics status.
rmoveto	graf rmoveto dh dv	Move by (dh,dv).
rlineto	graf rlineto dh dv	Draw a line (dh,dv).
save	graf save	Save the graphics status.
setbkcolor	graf setbkcolor color	Set the background color.
setcolor	graf setcolor color	Set the color for drawing.
setfont	graf setfont fontid	Set the font for text.
setfontsize	graf setfontsize fontsize	Set the font size for text.
setorigin	graf setorigin h v	Set the origin of coordinate to (h,v).
setlinestyle	graf setlinestyle linestyle	Set the line style.
setlinewidth	graf setlinewidth linewidth	Set the line width.
setscale	graf setscale scale	Set the scale for graphics object and text.
show	graf show string	Draw a text string left adjusted.
showcenter	graf showcenter string	Draw a text string centered.
showright	graf showright string	Draw a text string right adjusted.
vshowdown	graf vshowdown string	Draw a text string top to bottom.
vshowup	graf vshowup string	Draw a text string bottom to top.

Table F.1: List of the graphics related commands.

Appendix G. Window Command Syntax

Commands	Syntax	Function
box	window box wtitle blink bc tc l t r b	Draw a rectangular box (l, t, r, b) around a window.
create	window create wtitle blink bc tc l t r b	Create a window.
cwrite	window cwrite wtitle blink bc tc l t r b text	Draw a colored text in a rectangle (l, t, r, b) in a window.
debox	window debox wtitle l t r b	Remove the rectangular box (l, t, r, b) in a window.
delete	window delete [wtitle all]	Delete a window or all windows.
fill	window fill wtitle blink bc tc l t r b	Fill the rectangle (l, t, r, b) with bc. tc is the color of text in the rectangle.
hide	window hide [wtitle all]	Hide a window or all windows.
move	window move wtitle l t	Move a window to a new location.
resize	window resize wtitle dw dh	Resize the window to dw x dh.
show	window show wtitle	Show a window.
write	window write wtitle l t r b text	Draw a text in a rectangle (l, t, r, b) in a window.

Table G.1: List of the window related commands. wtitle = window title, bc = background color, tc = text color, l = left, t = top, r = right, b = bottom.

Appendix H. GPIB Command Syntax

Commands	Syntax	Function
clear	gpib clear dev	Clear a GPIB device.
command	gpib command [board brd] cmd cnt	Output a GPIB command to a board.
control	gpib control [to] dev	Pass the GPIB control to a device.
ifclear	gpib ifclear dev	Interface clear a GPIB device.
llock	gpib llock [board brd] dev	Disable the local mode of a GPIB device.
local	gpib local dev	Make a GPIB device local.
open	gpib open dev [timeout tmo]	Open a GPIB device.
read	gpib read [line file ffile] [from] dev [to] dest [upto eos]	Read a word, a line, or a file from a GPIB device.
remote	gpib remote [board brd] dev [on off]	Enable/disable REN (remote enable) line.
trigger	gpib trigger dev	Trigger a GPIB device.
write	gpib write [line file] [to] dev [from] srce [upto eos]	Write a word, a line, or a file from a GPIB device.

Table H.1: The commands for communication to external devices via the GPIB interface. The characters in square brackets are optional.

Appendix I. Device Driver Command Syntax

Commands	Syntax	Function
read	atmio read [line file] [all chan chan1 chan2] var	Read A/D conversion result(s) and store to 'dest'. 'dest' is either a variable name or a file name.
write	atmio write line [all dac0 dac1] expr	Output D/A conversion result. 'expr' is the desired voltage.

Table I.1: The commands for the ATMIO device driver. The characters in square brackets are optional.

Commands	Syntax	Function
read	neat read [line file] [from] srce [to] dest	Read from a NEAT board and store to 'dest'. 'dest' is either a variable name or a file name.
write	neat write [line file] [to] dest [from] srce	Write to a NEAT board. 'srce' is test string or a file name.

Table I.2: The commands for the NEAT device driver. The characters in square brackets are optional.

Commands	Syntax	Function
device	atdio device integrator [a b c d] [readstat & var readcnt & var1 & var2 readncnt n & var1 & var2 reset trigger]	Read integrator status byte. Integrate 1 time and read the counts. Integrate n times and read the counts. Reset the integrator. Trigger the integrator.
	atdio device timebase [setup int1 int2 start stop]	Set up, start, or stop the timebase.
	atdio device pscontrol [ramp currf irate read & var1 & var2 record nsample interval]	Set the current and the ramp rate. Read the current and the ripple. Record the current as a function of time.
	atdio device trigger [a b ... h] [setup delay interval start stop]	Set up, start, or stop the gated trigger unit.

Commands	Syntax	Function
read	atdio read [integer long] from [[porta portb portc portd] [group1 group2]] & var	Read from a port or a group of ports and store to 'var'. Group1 is porta and portb, and group2 is portc and portd.
write	atdio write [[char [porta portb portc portd]] [integer [cfg3 group1 group2]]] expr	Write 'expr' to a port or a group of ports. Group1 is porta and portb, and group2 is portc and portd. 'cfg3' is the board configuration port.

Table I.3: The commands for the ATDIO device driver. The characters in square brackets are optional.

Appendix J. Script Example SORTCAP.SCR

```
procedure sortcap.scr

!*   This is a script for measurement of BPM button capacitance
*   for sorting before mounting on the vacuum chamber.  The whole
*   procedure is automated and requires little intervention by
*   the operator.
*!

{
    call initialize;
    call start;
    call setup_tek;
    call measure_cap;
    call finish;
end;
}

subroutine show_usage

{
    TOSCRNL "Script for measuring the button capacitance.";
    TOSCRNL "Syntax:";
    TOSCRNL "    run sortcap [-n | -a] sumfile";
    TOSCRNL "    -n: name of the new summary file.";
    TOSCRNL "    -a: name of the summary file to append to.";
    return;
}

subroutine finish

{
    window delete star;

    open/n #logfile;
    TOLOGL "last_session_operator =" #opr_name;
    TOLOGL "last_session_startime =" #starttime;
    TOLOGL "last_buttonid =" #but_id;
    TOLOGL "last_meas_date =" #meastime;
    close #logfile;

    close #sumfile;

    return;
}
```

```

subroutine get_data
{
!   Initialize the data display.

GWT "autos star";
GWT "chml offset:" || #tekoffset || ",sensi:" || #teksensi;
GWT "mainpos " || #tektbase;
GWT "tbmain len:" || #itlen || ",time:" || #tektdiv;
GWT "navg " || #teknavg;
GWT "tral des:'avg(m1)'" ;
wait for #tekwait;

!   Get the data for normalization to obtain the saturation
!   level.

GWT "curve?";
GRFTB norm.dat upto 0;

!   Turn off averaging for subsequent data.

GWT "chml offset:" || #tekoffset || ",sensi:" || #teksensi;
GWT "tral des:'m1'" ;
GWT "mainpos" #tektbase;
GWT "tbmain time:" #tektdiv;
wait for #tekwait;

!   Write the header data.

GWT "mainpos?";
GRT #line1;
#str_len = strsize (#line1);
GWT "tbmain?";
GRT #line2;
add #str_len by strsize (#line2) + 2;

TOFILEL "Header Data";
TOFILEL format ("Size = %d", #str_len);
TOFILEL #line1;
TOFILEL #line2;

!   Write the normalization data.

TOFILEL "Normalization Data";
TOFILEL format ("Size = %d", fsize (norm.dat));
TOFILEL norm.dat;

for #i = 1, #nmeas, 1 {
    #il = #i - 1;

    GWT "curve?";
    GRFTB wave.dat upto 0;

```

```

open/b/n curv.dat;

GWT "mainpos?";
GRT #line;

write line to curv.dat from #line;

GWT "tbmain?";
GRT #line;
write line to curv.dat from #line;

write file to curv.dat from norm.dat;
write file to curv.dat from wave.dat;

close curv.dat;
dos #vmdir || \cap -i curv.dat -o out.dat
    -x #init #ifit #loff -m 0.0 -a #cavg;

open out.dat;
read line from out.dat #line;
read line from out.dat #line;
close out.dat;

#capac[#i1] = picknum (#line, 5);
TOSCRNL format ("%8d\t%15.5f", #i, #capac[#i1]);
TOFILEL;
TOFILEL format ("Case = %d", #i);
TOFILEL format ("Capacitance = %15.5f pF", #capac[#i1]);
TOFILEL format ("Size = %d", fsize (wave.dat));
TOFILEL wave.dat;
}

TOSCRNL "-----";
TOSCRNL format ("Average = `t%15.5f", arrmean (#capac));
TOSCRNL format ("Stdev   = `t%15.5f", arrstdev (#capac));
TOSCRNL;

file go to beginning off 0 in #datfile;
file find "    Case`tButton Capacitance (pF)" in #datfile;
FRFILEL #line;
call write_results;

! Write the data into the summary file.

write char to #sumfile from
    format ("%04d\t%7.5f\t%7.5f\t%s\t%s", #but_id,
        arrmean (#capac), arrstdev (#capac),
        #opr_name, #m_date, #m_time);
for #i = 0, #nmeas - 1, 1 {
    write char to #sumfile from
        format ("`t%7.5f", #capac[#i]);
}
write line to #sumfile from "";

```

```

return;
}

subroutine initialize
{
    int #itlen = 1024;      ! number of data points.
    int #nmeas = 5;        ! number of measurements per button
    int #done = 0;         ! true if the measurement is done.

    int #init;             ! the pixel where to begin fitting.
                           ! offset from the minimum point.
    int #ifit;             ! the number of pixels for fitting.
    int #ioff;             ! the number of pixels for calculating
                           ! the offset (saturation value).

    #init = 0.02 * #itlen;
    #ifit = 0.35 * #itlen;
    #ioff = 0.2 * #itlen;

    int #teknavg = 16;     ! number of averaging done on
                           ! the scope.
    float #tekoffset = -0.20; ! vertical offset.
    float #teksensi = 0.05; ! vertical sensitivity.
    float #tektbase = 70.28e-9; ! time base.
    float #tektdiv = 2.e-10; ! time per division.
    float #tekwait = 6.0;   ! time in sec to wait for the
                           ! averaging done.
    float #cavg;            ! interval of averaging to
                           ! eliminate the 12 GHz ripple.

    #cavg = #itlen / (10.24 * #tektdiv * 12.e9);

    string #tekprec = "optional";

    double #capac[#nmeas]; ! capacitance array.
    double #capavg;         ! average capacitance.
    double #capstdev;       ! standard deviation.

    int #but_id;           ! button id.
    string #datfile;       ! name of the data file.
                           ! One data file will be
                           ! assigned to each button.
    string #logfile;       ! name of the session log file.
    string #sumfile;       ! name of the summary file.

    string #dt_dir;        ! directory for data.
    string #bu_dir;        ! backup directory for data.

    string #opr_name;      ! name of the operator.
    string #starttime;     ! datetime when the session started.
    string #meastime;      ! time when the measurement was made.
    string #m_date;        ! date when the measurement was made.

```

```

string #m_time;          ! time when the measurement was made.

! Miscellaneous variables.

int #str_len;
string #line, #answer, #string;

! Aliases.

define FRSCRNL read line from screen to;

define TOSCRNC write char to screen from;
define TOSCRNW write word to screen from;
define TOSCRNL write line to screen from;

define FRFILEL read line from #datfile to;

define TOFILEC write char to #datfile from;
define TOFILEW write word to #datfile from;
define TOFILEL write line to #datfile from;
define TOFILEF write file to #datfile from;

define TOLOGL write line to #logfile from;
define FRLOGL read line from #logfile to;

define BACKBYTE(A) file go to current off -1 in A;

define GWT      gpib write line to tek from;
define GRT      gpib read line from tek to;
define GRFT     gpib read file from tek to;
define GRFTB    gpib read /b file from tek to;

define QUERY(A,B) {
    write word to screen from A;
    read line from screen to B;
    window hide star;
}
define QQUERY(A,B) {
    do {
        TOSCRNW A;
        FRSCRNL B;
        TOSCRNL B || ", correct? (y or n)";
        FRSCRNL #t_answer;
    } while (capital (#t_answer) .ne. "Y");
    window hide star;
}

define ALERTW(A,B,C) {
    write char to screen "~07";
    window write star 3 2 45 2 A;
    window write star 3 3 45 3 B;
    window write star 3 4 45 4 C;
    window show star;
    pause 0;
}

```

```

        window hide star;
    }

    #dt_dir = d:\butcap;
    #bu_dir = d:\butcapbu;
    #logfile = #dt_dir || \caplog.dat;
    #sumfile = #dt_dir || \capsum.dat;

!*   Read info from the log file.
    The format of the logfile is:
        last_session_operator = last_opr_name
        last_session_starttime = last_starttime
        last_buttonid = last_but_id
        last_meas_datetime = last_m_datetime
*!

    pickopt -n #sumfile #ia_nsum;
    pickopt -a #sumfile #ia_asum;
    if (.not. (#ia_nsum + #ia_asum)) {
        call show_usage;
    }

    if (fexist (#logfile)) {
        TOSCRNL "The last session log:";
        TOSCRNL;
        dos type #logfile;
        TOSCRNL;
    }

    return;
}

subroutine measure_cap
{
    do {
        QQUERY ("Type in the button ID, please. Numerics only.
", #but_id);
        ALERTW ("Please connect the button to the cable",
                "for measurement.",
                "Hit return when ready.");
        #datfile = format ("%s`c%04d.dat", #dt_dir, #but_id);
        open/b/n #datfile;

        call write_header;
        call get_data;

        close #datfile;
        copy #datfile #bu_dir;

        QQUERY ("Do you have more buttons to measure? (y or n)
", #answer);
    }
}

```

```

        #done = capital (#answer) .eq. "N";
    } while (.not. #done);
}

subroutine setup_tek

{
    ALERTW ("Connect an SMA cable to channel 1 and",
           "connect a button to the cable.",
           "Hit return when ready.");

!   Display adjustment.

    GWT "init";
    GWT "disp mod:vec";
    GWT "enc set:ascii,wav:bin";
    GWT "trig mode:normal,source:int";
    GWT "gra refa:2.5e-1,xun:sec,yun:rho";

!   Select the trace and the channel.

    GWT "rem tra1";
    GWT "rem tra2";
    GWT "rem tra3";
    GWT "rem tra4";
    GWT "sel tra1";
    GWT "out tra1";
    GWT "tbmain len:" #itlen;

!   GWT "tra1 des:'smooth(m1,20)'" ;

    GWT "tra1 des:'m1'";
    GWT "chm1 tdrs:on";
    GWT "chm2 tdrs:off";
    GWT "chm3 tdrs:off";
    GWT "chm4 tdrs:off";

!   Adjust the display offset and sensitivity.

    GWT "chm1 offset:" || #tekoffset || ",sensi:" || #teksensi;
    GWT "tra1 des:'m1'";
    GWT "mainpos" #tektbase;
    GWT "tbmain time:" #tektdiv;

    ALERTW ("Adjust the time base with the knob until",
           "the dip on the trace is 10% from the left.",
           "Hit return when ready.");

    GWT "mainpos?";
    GRT #line;          !   format of #line is "MAINPOS %e".
    #tektbase = picknum (#line, 1);

    GWT "tbmain?";

```

```

GRT #line;      !      format of #line is
                !      "TBMAIN LENGTH:%d,TIME:%e,XINCR:%e".
#tektdiv = picknum (#line, 2);

return;
}

subroutine start
{
    window create star 0 0 15 18 5 63 9;
    window box star 0 0 14 1 1 80 80;

!    Check if a new summary file is to be created.

    if (#ia_nsum) {
        #line = format ("You specified a new summary file
`"%s`". Are you sure? (y or n) ", #sumfile);
        if (capital (#answer) .eq. "Y") {
            open/n #sumfile;
            write line to #sumfile from
                format ("Button ID`tC_p`tStdev`tOperator`tDate
`tTime`tCase 1`tCase 2`tCase 3`tCase 4`tCase 5");
        }
    }
    else {
        TOSCRNL format
        ("Appending to the summary file `"%s`".", #sumfile);
        open/a #sumfile;
    }

    copy #dt_dir || \cap.exe #vmdir;
    QQUERY ("Type in your name, please. ", #opr_name);
    #starttime = datetime (0);
}

subroutine write_header
{
    #meastime = datetime (0);
    #m_time = pickword (#meastime, 1);
    #m_date = pickword (#meastime, 2);

    TOFILEL format ("Operator name:           %s", #opr_name);
    TOFILEL format ("Session start datetime:      %s", #starttime);
    TOFILEL;
    TOFILEL format ("Button ID:                %04d", #but_id);
    TOFILEL format ("Measurement datetime:       %s", #meastime);
    TOFILEL;
    TOFILEL format ("      Case`tButton Capacitance (pF)");

    TOSCRNL;
}

```

```

TOSCRNL format ("    Case`tButton Capacitance (pF)");
TOSCRNL;

!   Make room for the capacitance data in the file.

    call write_results;
}

subroutine write_results
{
  TOFILEL;
  for #i = 1, #nmeas, 1 {
    TOFILEL format ("%8d`t%15.5f", #i, #capac[#i-1]);
  }
  TOFILEL "-----";
  TOFILEL format ("Average = `t%15.5f", arrmean (#capac));
  TOFILEL format ("Stdev   = `t%15.5f", arrstdev (#capac));
  TOFILEL;
}

```

Appendix K. Script Example RADTRIG.SCR

```
procedure radtrig.scr
{
  define INCH_PULSE 5080;
  define CM_PULSE 2000;
  define GMT gpib write line to motor;
  define TOSCRNL write line to screen;
  define TOFILEL write line to #fname;

  int #ndata, #npulse, #nlength, #ndelay, #nver, #interact;
  float #xp, #xpm, #xpen, #xbndin, #xbndout, #signx;
  double #gain, #cntvsec;
  string #answer, #line1;
  string #fname, #fgname, #frname, #ffname, #fxname;

  call get_input;

  #xp = -1.0 * #xbndout;
  call initcoil;
  TOSCRNL " Current Position =" #xpen;

  window write star 3 3 45 3
    " Now moving PC coil in forward direction";
  window write star 3 4 45 4
    " Hit return to continue...";
  window show star;
  if (#interact) {
    pause 0;
  }
  window hide star;

  #xp = #xbndout;
  call movecoil;
  #npulse = 2 * #xbndin * INCH_PULSE / #ndata;
  #nlength = #npulse * #ndata + 1;
  #ndelay = abs (#xpen) * INCH_PULSE - #npulse * #ndata / 2;
  atdio device trig b setup 1 #npulse;
  atdio device trig a setup #ndelay #nlength;
  atdio device trig b start;
  atdio device trig a start;
  GMT start;
  integ_ab -a -f -n #ndata;
  atdio device trig b stop;
  atdio device trig a stop;
  gpib read line from motor #line1;
  #xpen = picknum (#line1, 1) * #signx;
  TOSCRNL " Current Position =" #xpen;

  window write star 3 3 45 3
```

```

    " Now moving PC coil in reverse direction";
window write star 3 4 45 4
    " Hit return to continue...";
window show star;
if (#interact) {
    pause 0;
}
window hide star;
#xp = -1.0 * #xbndout;
call movecoil;
#npulse = 2 * #xbndin * INCH_PULSE / #ndata;
#nlength = #npulse * #ndata + 1;
#ndelay = abs (#xpen) * INCH_PULSE - #npulse * #ndata / 2;
atdio device trig b setup 1 #npulse;
atdio device trig a setup #ndelay #nlength;
atdio device trig b start;
atdio device trig a start;
GMT start;
integ_ab -a -r -n #ndata;
atdio device trig b stop;
atdio device trig a stop;
gpib read line from motor #line1;
#xpen = picknum (#line1, 1) * #signx;
TOSCRNL " Current Position =" #xpen;

call mvorig;
window write star 3 3 45 3 "";
window write star 3 4 45 4
    " Now Doing some calculation....";
window write star 3 5 45 5 "";
window show star;

float/l #frdt = frwd_a.vmf;
float/l #bkdt = bkwd_a.vmf;
float #avrg[#ndata];
float #avrg_add[#ndata + 1];
float #avrg_xxx[#ndata + 1];
#avrg_add[0] = 0;
#avrg_xxx[0] = #npulse * (0.0 - #ndata / 2.0) / INCH_PULSE;
for #i = 1, #ndata, 1 {
    #avrg[#i - 1] =
        #cntvsec * 0.5 * (#frdt[#i - 1] - #bkdt [#i - 1]);
    #avrg_add [#i] =
        #avrg_add [#i - 1] + #avrg[#i - 1];
    #avrg_xxx [#i] =
        #npulse * (#i - #ndata / 2.0) / INCH_PULSE;
}
save/b var to #frname #avrg;
save/b var to #fxname #avrg_xxx;
save/b var to #ffname #avrg_add;

window delete star;

if (#interact) {

```

```

        xyplot -i cmove.inp -x #fxname -y #ffname -o #fgname
        -t "PC Coil Measurement #" || #nver;
    }
    else {
        xyplot -i cmove.inp -x #fxname -y #ffname -o #fgname -b
        -t "PC Coil Measurement #" || #nver;
    }
    tprint pnetq /ps- #fgname;
}

```

```

subroutine get_input

```

```

{
    #gain = 100.0;
    #cntvsec = 5.0 / 250000.0 / #gain;
    #xbndout = 1.4;

    #signx = 1.0;

    pickopt -b #line #tf;
    if (#tf) {
        #interact = 0;
    }
    else {
        #interact = 1;
    }
    pickopt -v #line #tf;
    if (#tf) {
        #nver = picknum (#line, 1);
    }
    else {
        call show_usage;
    }

    pickopt -x #line #tf;
    if (#tf) {
        #xbndin = picknum (#line, 1);
    }
    else {
        #xbndin = 1.0;
    }
    pickopt -n #line #tf;
    if (#tf) {
        #ndata = picknum (#line, 1);
    }
    else {
        #ndata = (#xbndin * INCH_PULSE * 2) / 127;
    }
    ! #fname = "CMOVE" || #nver || ".REC";
    #fgname = "CMOVE" || #nver || ".PSF";
    #frname = "CMOVR" || #nver || ".VMF";
    #ffname = "CMOVF" || #nver || ".VMF";
    #fxname = "CMOVX" || #nver || ".VMF";
}

```

```

TOSCRNL " ";
}

subroutine initcoil
{
    window create star 0 1 15 18 5 63 10;
    window box star 0 1 14 1 1 80 80;
    window write star 3 3 45 3
        " Now moving PC coil to starting position.";
    window write star 3 4 45 4
        " Hit return to continue...";
    window show star;
    if (#interact) {
        pause 0;
    }
    window hide star;

    gpib clear motor;
    wait for 0.2;
    GMT select stepinit;
    wait for 0.2;
    GMT load;
    wait for 0.2;
    GMT "mode e_abs e_abs * *";
    GMT "enco mres 25000 25000 * *";
    GMT "enco eres 508 508 * *";
    GMT "unit pos 5080.0000 5080.0000 * *";
    GMT "vel 100000 100000 * *";
    GMT "accel 80000 80000 * *";
    GMT "decel 80000 80000 * *";
    GMT "slimit e_cw 1.6 1.6 * *";
    GMT "slimit e_ccw -1.6 -1.6 * *";
    GMT "pdef 0 0 0 *";
    GMT "*";
    wait for 1.0;
    gpib read line from motor #answer;
    wait for 0.5;
    GMT start;
    wait for 0.5;

    do {
        call movecoil;
        GMT start;
        wait for 5.0;
        gpib read line from motor #line1;
        write line to screen #line1;
        #xpen = picknum (#line1, 1) * #signx;
    } while (#xpen .ne. #xp);
}

subroutine movecoil

```

```

{
    #xpm = #xp * #signx;

    gpib clear motor;
    wait for 0.2;
    GMT select stepmv;
    wait for 0.2;
    GMT load;
    wait for 0.2;
    GMT "mode e_abs e_abs * *";
    GMT "enco mres 25000 25000 * *";
    GMT "enco eres 508 508 * *";
    GMT "unit pos 5080.0000 5080.0000 * *";
    GMT "vel 100000 100000 * *";
    GMT "accel 80000 80000 * *";
    GMT "decel 80000 80000 * *";
    GMT "slimit e_cw 1.6 1.6 * *";
    GMT "slimit e_ccw -1.6 -1.6 * *";
    GMT "wait for 5.000 seconds";
    GMT "move" #xpm #xpm "* *";
    GMT "in q2 = position of axis2 eabs";
    GMT "out port3 ^ X =^ q2";
    GMT "*";
    wait for 0.5;
    gpib read line from motor #answer;
    wait for 0.2;
}

```

subroutine mvorig

```

{
    window write star 3 3 45 3
        " Now moving PC coil to the origin.";
    window write star 3 4 45 4
        " Hit return to continue...";
    window show star;
    if (#interact) {
        pause 0;
    }
    window hide star;

    do {
        #xp = 0.0;
        call movecoil;
        GMT start;
        wait for 5.0;
        gpib read line from motor #line1;
        #xpen = picknum (#line1, 1) * #signx;
        if (#xpen .ne. 0.0) {
            #xp = 0.1;
            call movecoil;
            GMT start;
        }
    }
}

```

```

        wait for 5.0;
        gpib read line from motor #line1;
        #again = 1;
    }
    else {
        #again = 0;
    }
} while (#again);

window write star 3 3 45 3 " Now PC coil is at origin";
window write star 3 4 45 4 "      x =" #xpen "inch";
window write star 3 4 45 5 " Hit return to continue...";
window show star;
if (#interact) {
    pause 0;
}
window hide star;
}

subroutine show_usage
{
    TOSCRNL
    "Usage: RUN RADTRIG [-b] -v #num [-x #xin] [-n #ndata]";
    TOSCRNL "-b: Operate in batch mode (default : interactive)";
    TOSCRNL "-v #num      : Measurement number(99 < #num < 1000)";
    TOSCRNL "-x #xin       : Integration boundary";
    TOSCRNL "          (-#xin < integration region < #xin)";
    TOSCRNL "-n #ndata    : Number of intervals";
    TOSCRNL "          (default : (#xin * 5080 * 2) / 127)";
    end;
}

```