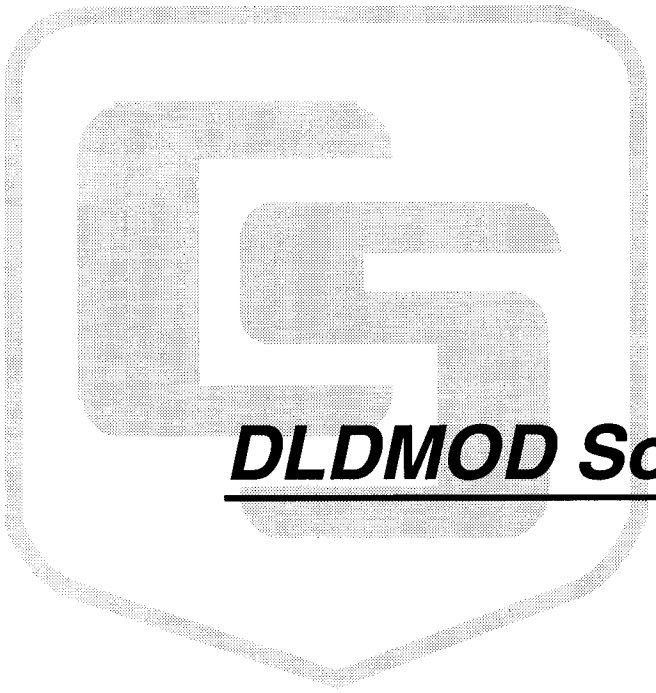
INSTRUCTION MANUAL



DLDMOD Software Manual

Campbell Scientific, Inc.

CAMPBELL SCIENTIFIC, INC.

DLDMOD
PRELIMINARY MANUAL

February 8, 1996

INTRODUCTION TO THE DLDMOD MANUAL

This manual is supplied with the preliminary release of DLDMOD.

There are two parts to this manual. The first part is intended to provide an overview of DLDMOD. The second part describes the procedures required to develop a working DLDMOD application. All of the available commands are listed with several examples of how to use them. As with most computer languages, much can be learned by studying programs previously written by software developers familiar with the language. For this reason, some sample programs have been included on the floppy disk supplied to you.

Throughout this manual, "developer" is used to describe the person that will write the DLDMOD application. This is most likely the person reading this manual. The word "user" describes the person running the application created by the "developer".

It is important that you read the agreement included with the package you received and that you comply with the provisions stated.

As a developer, you are responsible for supporting your DLDMOD applications and the datalogger programs created with your applications. Campbell Scientific, Inc. cannot assume any responsibility for the support of applications you create. You should not distribute your application (or allow your application to be distributed) to users that you will not support.

Comments and suggestions are welcome and should be relayed to the person that supplied your copy of DLDMOD.

LIMITED WARRANTY

Campbell Scientific, Inc. warrants that the magnetic diskette on which the accompanying computer software is recorded and the documentation provided with it are free from physical defects in materials and workmanship under normal use. Campbell Scientific, Inc. warrants the computer software itself will perform substantially in accordance with the specifications set forth in the Operator's Manual published by Campbell Scientific, Inc. Campbell Scientific, Inc. warrants the software is compatible with IBM PC/XT/AT and PS/2 microcomputers and 100% compatible computers only. Campbell Scientific, Inc. is not responsible for incompatibility of this software running under any operating system other than those specified in accompanying data sheets or operator's manuals.

The above warranties are made for ninety (90) days from the date of original shipment.

Campbell Scientific, Inc. will replace any magnetic diskette or documentation which proves defective in materials or workmanship without charge.

Campbell Scientific, Inc. will either replace or correct any software that does not perform substantially according to the specifications set forth in the Operator's Manual with a corrected copy of the software or corrective code. In the case of a significant error in the documentation, Campbell Scientific, Inc. will correct errors in the documentation without charge by providing addenda or substitute pages.

If Campbell Scientific, Inc. is unable to replace defective documentation or a defective diskette, or if Campbell Scientific, Inc. is unable to provide corrected software or corrected documentation within a reasonable time, Campbell Scientific, Inc. will either replace the software with a functionally similar program or refund the purchase price paid for the software.

Campbell Scientific, Inc. does not warrant that the software will meet licensee's requirements or that the software or documentation are error free or that the operation of the software will be uninterrupted. The warranty does not cover any diskette or documentation which has been damaged or abused. The software warranty does not cover any software which has been altered or changed in any way by anyone other than Campbell Scientific, Inc. Campbell Scientific, Inc. is not responsible for problems caused by

computer hardware, computer operating systems or the use of Campbell Scientific, Inc.'s software with non-Campbell Scientific, Inc. software.

ALL WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED AND EXCLUDED. CAMPBELL SCIENTIFIC, INC. SHALL NOT IN ANY CASE BE LIABLE FOR SPECIAL, INCIDENTAL, CONSEQUENTIAL, INDIRECT, OR OTHER SIMILAR DAMAGES EVEN IF CAMPBELL SCIENTIFIC, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Campbell Scientific, Inc. is not responsible for any costs incurred as a result of lost profits or revenue, loss of use of the software, loss of data, cost of re-creating lost data, the cost of any substitute program, claims by any party other than the licensee, or for other similar costs.

LICENSEE'S SOLE AND EXCLUSIVE REMEDY IS SET FORTH IN THIS LIMITED WARRANTY. CAMPBELL SCIENTIFIC, INC.'S AGGREGATE LIABILITY ARISING FROM OR RELATING TO THIS AGREEMENT OR THE SOFTWARE OR DOCUMENTATION (REGARDLESS OF THE FORM OF ACTION - E.G. CONTRACT, TORT, COMPUTER MALPRACTICE, FRAUD AND/OR OTHERWISE) IS LIMITED TO THE PURCHASE PRICE PAID BY THE LICENSEE.

LICENSE FOR USE

This software is protected by both United States copyright law and international copyright treaty provisions. You may copy it onto a computer to be used and you may make archival copies of the software for the sole purpose of backing-up Campbell Scientific, Inc. software and protecting your investment from loss. All copyright notices and labeling must be left intact.

This software may be used by any number of people, and may be freely moved from one computer location to another, so long as there is no possibility of it being used at one location while it's being used at another. The software, under the terms of this license, cannot be used by two different people in two different places at the same time.

RELATIONSHIP

Campbell Scientific, Inc. hereby grants license to use DLDMOD in accordance with license statement above.

No ownership in Campbell Scientific, Inc. patents, copyright, trade secrets, trademarks, or trade names is transferred by this Agreement.

Developer may create as many different applications as desired and freely distribute them. Campbell Scientific, Inc. expects no royalties or any other compensation outside of the DLDMOD purchase price.

Developer is responsible for supporting DLDMOD applications created by the developer.

RESPONSIBILITIES OF DEVELOPER

The developer agrees:

To provide a competent programmer familiar with Campbell Scientific, Inc. datalogger programming to write the DLDMOD applications.

Not to sell or distribute "COMPILE.EXE" or "DLDMOD.EXE" in any form.

Not to freely distribute any other Campbell Scientific, Inc. Software (i.e. PC208) in any form.

Applications developed with DLDMOD will be solely for the support of Campbell Scientific, Inc. dataloggers. No attempt will be made to support non-Campbell Scientific, Inc. dataloggers with DLDMOD applications.

To assure that each application developed with DLDMOD clearly states the name of the person or entity that developed the application. This information should appear on the first window the user will see.

WARRANTY

There is no written or implied warranty provided with DLDMOD software other than as stated herein.

TERMINATION

Any license violation or breach of Agreement will result in immediate termination of the developer rights herein and the recovery of all DLDMOD materials supplied by Campbell Scientific, Inc.

MISCELLANEOUS

Notices required hereunder shall be in writing and shall be given by telegram, telex, or similar communication or by certified or registered air mail, return receipt requested. Such notice shall be deemed given in the case of telegrams or similar communication when sent and in the case of certified or registered mail on the date of receipt.

This Agreement shall be governed and construed in accordance with the laws of the State of Utah, U.S.A. Any dispute resulting from this Agreement will be settled in arbitration.

This Agreement sets forth the entire understanding of the parties and supersedes all prior agreements, arrangements and communications, whether oral or written pertaining to the subject matter hereof. This Agreement shall not be modified or amended except by the mutual written agreement of the parties. The failure of either party to enforce any of the previsions of this Agreement shall not be construed as a waiver of such provisions or of the right of such party thereafter to enforce each and every provision contained herein. If any term, clause, or provision contained in this Agreement is declared or held invalid by a court of competent jurisdiction, such declaration or holding shall not affect the validity of any other term, clause, or provision herein contained. Neither the rights nor the obligations arising under this Agreement are assignable or transferable.

OVERVIEW OF DLDMOD

USER

The user has measurements to make with a datalogger and a DLDMOD application created by a developer for the user's application or a similar application. The user does not need any knowledge of datalogger programming. DLDMOD provides an easy way to set up the datalogger for the supported application. The user runs the DLDMOD application, fills in the blanks, and selects options from menus. There may also be appropriate help prompts for each blank and option. As the answers are entered, they are checked against a list of acceptable answers. The user is asked to reenter any values that fail. When this is done, a FILENAME.DLD file has been created and is ready to download to the datalogger. This manual has been written for the developer rather than the user.

DEVELOPER

The developer creates the DLDMOD application the user will run. The developer has knowledge of the types of measurement and control the user wants to do. While the developer may not know each user's exact datalogger configuration, the developer does understand what variations the users might have. The developer also knows how to program a datalogger. The developer could program the FILENAME.DLD file for the user directly if it was known exactly what the user needed to do.

DLDMOD is a programming language designed to allow the developer to ask the user questions and thereby change or create a FILENAME.DLD file based on the user's answers. DLDMOD programming consists of two parts: getting information from the user, and creating a data-logger program for the user. As part of creating a datalogger program, wiring diagrams can be generated, input location usage shown, etc. DLDMOD does not do these things, but DLDMOD provides the tools for the developer to do them.

INSTALLATION

You should make a backup copy of the original DLDMOD disk. To use DLDMOD, simply copy the files from the backup copy of the original disk to the directory of your choice. DLDMOD works best from a hard disk.

The following example shows the DOS commands to install the DLDMOD software in a subdirectory named C:\DLDMOD on the C: drive. It assumes the floppy containing DLDMOD is in the A: drive.

```
CD\
MD DLDMOD
CD DLDMOD
COPY A:*.*
```

SAMPLE PROGRAMS

This section will help explain the ten sample overview programs (OVERVW1.FMT through OVERVW9.FMT and DEMO.FMT) included with DLDMOD. The steps to compile and run these .FMT source files are described so you can try them. For example, to run OVERVW1.FMT, you would type the following at a DOS prompt (after changing to the sub directory where DLDMOD was installed) :

```
Compile overvw1
Makeexe overvw1
overvw1
```

Repeat these steps, with the appropriate file name, to run the other examples. While running an example, press F1 for a list of keys that can be used. Go ahead and try some of the examples now.

When you run any example the second time, you will see a screen asking if you want to use the default answers or those you selected last time you ran the application. Experiment with both options. DLDMOD saves all of the variables in a file when program execution moves to the Compile section. When a DLDMOD application is run, it checks for the file containing the variables and will optionally use the values as the defaults. This is built into DLDMOD. You can select the default answers without being asked by typing a "D" after the example name. For example, to run the first example you could type:

```
overvw1 d
```

This would use the default answers.

All these examples are DLDMOD programs that allow the end-user to make a measurement with one of three sensors: SENSOR A, SENSOR B, and SENSOR C. A NO SENSOR selection is also provided. The user's selection will be used to create a datalogger program named OVERVIEW.DLD. Note that the sensors, datalogger instructions, and the OVERVIEW.DLD program are examples only. All of the examples create a file named OVERVIEW.DLD, overwriting any existing files with the same name. The example datalogger instructions needed to measure each of the three sensors are as follows:

```
Sensor A
01:P1
01:1
02:2
03:3
04:4

Sensor B
01:P2
01:3
02:2
03:1

Sensor C
01:P3
01:4
02:5
03:6
04:7
05:8
```

All DLDMOD programs have two main sections; Declarations and Compile. In the Declaration section, the developer sets up what the user will see and do. In the Compile section the developer actually makes changes to the .DLD file. When a DLDMOD program is run, it starts by displaying the first windows, as declared in the Declarations section.

From this point, the user navigates through the windows with the cursor keys or the mouse, making selections (Menu options or options) and filling in prompts (variables). DLDMOD provides two ways to get information from the user: variables and menu options. Variables provide a prompt and the user types in the appropriate information. Try OVERVW1 to see an example of a variable. Menu options allow the user to press ENTER and select an option after moving the highlight cursor to the option he wants. Try OVERVW2 to see an example using options.

The Declaration section describes how the windows are displayed and what options and variables are used. When the user has finished all the declared windows or when the user presses the F4 key, DLDMOD program flow moves to the Compile section. Usually at this point, the user's answers are committed and are used to actually create the FILENAME.DLD file (OVERVIEW.DLD in these examples).

Prior to moving to the Compile section, it is good practice to allow the user to change answers and responses as needed. Program execution before moving to the Compile section is not sequential, but is event driven in nature. Windows are displayed and removed based on the user's selections and key presses. The sequence of the main windows and the response to each menu option is described in the Declarations section, but within these constraints the user is free to navigate as desired.

Program execution in the Compile section is sequential and it is easy to trace program flow. The following is the Compile section from the first few examples. Remember this is the code that actually changes OVERVIEW.DLD.

```
COMPILE
DLDFILE=New `overview',`CR10'
CHANGETO 10 at 1
IF (sensor = `A') THEN
  INSERT 1:1,2,3,4 at 1:1
ENDIF
IF (Sensor = `B') THEN
  INSERT 2:3,2,1 at 1:1
ENDIF
IF (Sensor = `C' ) THEN
  INSERT 3:4,5,6,7,8 at 1:1
endif
INSERT 28 at 10:1
INSERT 64 at 10:2
RENUM
SAVE
END
```

If the user selected SENSOR B the created OVERVIEW.DLD file would be:

```
}CR10
;overview
MODE 1
SCAN RATE 10
1:P2
```

```
1:3
2:2
3:1

MODE 10
1:28
2:64

^E^E
```

Compile and run OVERVW1 if you have not already done so. Notice how OVERVW1 ends as soon as you type a correct response? When you type enough characters to fill a variable's box, DLDMOD automatically advances to the next variable or option. If there are no more on the current window, DLDMOD will advance to the next window. If there are no more windows, DLDMOD advances to the Compile section. Run OVERVW1 again. Notice how typing ENTER accepts the default answer. Try an incorrect response to see how DLDMOD checks the answers. One disadvantage of variables is the user can always type in an incorrect response. DLDMOD will check the answers, but it can be frustrating for a user.

Now compile and run OVERVW2. OVERVW1 and OVERVW2 get the same information from the user and create the same OVERVIEW.DLD file. This gives a comparison of variables and menu options. Menu options provide a simple way to let a user see all the choices and select one without typing. Notice that OVERVW2 also ends when a selection is made. This is similar to what OVERVW1 did but for a different reason. The following fragment is from the Declaration section. Notice the Clearback attribute on the main window declaration:

```
WINDOWS
 Main
  full
  dialogbox, clearback, frame
  Nothing
```

The Clearback attribute causes a window to be removed after the first menu option is selected. If it were not present, the window would remain until the user pressed ESC or F4. Compile and run OVERVW3. Notice it has no Clearback declaration so the window is not removed until you select the DONE option. The following is the window declaration from OVERVW3:

```
WINDOWS
 Main
  full
  dialogbox, Frame
  readonly(sensor)+
```

The other difference is the addition of a READONLY command. This allows you to display a variable without allowing the user to move the cursor to it or edit it. Remember the last line of a window declaration is a single command (using multiple command will be discussed later) executed just before the window is displayed. This command marks the SENSOR variable as READONLY. We can still change the variable's value under program control, but a user cannot edit it while it is READONLY. READONLY variables are a handy way a to give users feedback when they make a selection using menu options. Any variable can be marked READONLY, and they are entered in the TEXT section just like any other variable.

The other change of interest is the DONE menu option. Looking at the Declaration shows how it works:

```
Done
 removewin
 `Done making selections'
```

When DONE is selected, the current window is removed. Since there are no more windows DLDMOD moves through the Compile section and the program ends.

Window declarations allow a single executable statement, executed just before the window is displayed. Menu options also allow a single executable statement. If you require more than one command, there are two ways to accomplish it. You could put multiple commands in a subroutine and then call the subroutine as the single statement. Subroutines are most useful if there is a need to execute the same code in multiple places. DLDMOD treats an entire IF THEN ELSE ENDIF block as one statement. You can group multiple statements into a single IF block and it will compile as a single statement. OVERVW4 is the same as OVERVW3 except it changes the display color of the READONLY variable. It uses the IF block technique to group two statements into the window declaration as follows:

```
WINDOWS
 Main
  full
  dialogbox, Frame
  if (1=1) then ; Use If statement to combine
   readonly(sensor)+ ;multiple statements where
```

```
  rocolor(light cyan) ; only one is expected!
 endif
```

Notice that IF (1=1) will always be true, so the two statements are always executed. Also notice the comments at the end of the line. Everything after the semicolon is ignored. Compile and run OVERVW4.

OVERVW5 introduces several new concepts. Compile and run it. OVERVW5 uses a subwindow. Subwindows are displayed only when the DISPLAY command is used. A subwindow is removed by:

- Selecting an option when the Clearback attribute is set.
- Pressing the ESC key.
- Filling in a variable box when that variable is the last item on the window.
- Execution of the Removewin command.

When a subwindow is removed, the previously displayed window is restored. Notice the use of the SELECT LOGGER menu option to display the subwindow. Subwindows are useful for grouping and reducing complexity.

The new information about what datalogger is being used is put to use in the first line of the Compile section:

```
DLDFILE=New `overview', dtype
```

This differs from our previous DLDFILE instruction by using the new variable DTYPE instead of the a constant type CR10 as in the previous examples:

```
DLDFILE=New `overview',`CR10'
```

Run OVERVW5 and select the 21X datalogger type. Type the OVERVIEW.DLD file and see the difference.

When you run OVERVW5, notice it is somewhat awkward to enter in the datalogger type. Also notice once you leave the LOGGERS subwindow, you have no indication of which datalogger you selected. OVERVW6 fixes both of these problems. No new techniques are used, just menu options and READONLY variables. Compile and run OVERVW6. Even with a small DLDMOD program, it is important to give visual indication of what a user has selected.

The DLDMOD License agreement requires that you indicate who created the DLDMOD application. OVERVW7 adds an introduction window to do just that. Notice the difference between windows and subwindows. Windows are displayed in sequence until they have all been displayed. Subwindows are only displayed with the DISPLAY command. Once the INTRO window has been displayed there is no way to return to it. The LOGGERS subwindow can be displayed as many times a needed. Notice how the INTRO window uses the DONE option to move to the next window. If INTRO had no variables or menu options the user would have to press RETURN to continue. Also notice that the DONE menu option is used in two different places. Menu options and variables can be used on multiple windows.

Compile and run OVERVW7 to get a feel for windows and subwindows. While running OVERVW7, press F1 and look at the help screen. DLDMOD provides this as a default help screen, but you can create your own. Compile and run OVERVW8. Notice how pressing the F1 key displays a customized HELP window. The HELP attribute, as part of window declaration, makes that window become the HELP window. The following are the window declarations from OVERVW8:

```
WINDOWS
 Intro
 full
 frame
 nothing
 Main
  full
  dialogbox, Frame
  if (1=1) then;  Use If statement to combine
   readonly(sensor)+ ; multiple statements
   readonly(dtype)+ ; where only one is expected
   rocolor(light cyan)
  endif
SUBWINDOWS
loggers
 5,5,30,12
  dialogbox,clearback,frame
nothing
myhelp
 7,7,45,15
 help,frame
 nothing
```

Two rules apply when creating your own HELP window. Only one window should have the HELP declaration. If two or more are declared, only one of them will be displayed. Also, do not display the HELP window with a DISPLAY command. This might violate the more general

rule of never allowing a window to be displayed twice at the same time.

Compile and run OVERVW9. OVERVW9 adds an additional subwindow to organize things a little better.

Also included on the DLDMOD disk is DEMO.FMT. Compile and run this program. It is an example using many of the techniques described here. Examine the source code for explanations on what was done. Many editors allow the special line drawing characters used in the DEMO.FMT file. To add these characters, press and hold the ALT key while you type the three digit code for the character you wish to draw. Consult an ASCII table (found in many DOS manuals) for the appropriate codes.

When modifying these examples or writing your own .FMT files, be sure to use an ASCII text editor. Most word processors do not store in plain ASCII, although some will optionally export to ASCII. The EDIT program that comes with DOS 5 or DOS 6 is a good ASCII editor.

PROGRAMMING HINTS

- It is important to observe the difference between an opening and a closing single quote when writing a FILENAME.FMT file. The opening quote is an accent mark, usually found on the upper left corner of the computer keyboard. The closing quote is an apostrophe.
- DLDMOD is not case sensitive A = a.
- Everything must be declared before you use it. For example, a variable must be declared before it can appear in a SET command. A window must be declared before it can be used in a DISPLAY command. You can rearrange the order of declarations to help accomplish this (i.e., subroutines can be after the variable declarations).
- Never allow a window to be displayed twice at the same time.
- The following lists the maximum number of variables, menu options, and windows that can be used:

Windows	90
Menu Options	140
Variables	320

CREATING AN APPLICATION

The process of getting the final questions and FILENAME.DLD file to the user requires four operations by the Developer.

1. Writing a FILENAME.BSE file - A .BSE is a FILENAME.DLD file that will be modified to fit the user's answers, if needed. This is optional since DLDMOD is also capable of generating a new FILENAME.DLD file.
2. Writing a FILENAME.FMT file - The .FMT file contains all the information the user will see and the actions to be taken depending on the answers.
3. Compiling the FILENAME.FMT file to a FILENAME.MEN file - A debugged .FMT file filled with numbers and parameters. Creating the FILENAME.MEN file involves compiling the FILENAME.FMT file.
4. Once the .MEN file is created, it is combined with DLDMOD.EXE to make a stand-alone FILENAME.EXE file to distribute to the customer. A batch file, MAKEEXE.BAT, does this. You shouldn't distribute the .MEN file, COMPILE.EXE, or DLDMOD.EXE. Only the FILENAME.EXE file and the .BSE file (if one is necessary) should be distributed.

For example, if the application created was to be named SAMPLE, the steps would be as follows:

1. Use EDLOG to create the base file. EDLOG creates SAMPLE.DLD which is copied and renamed to SAMPLE.BSE. (.BSE files are optional depending on the application)
2. Use a text editor to create the SAMPLE.FMT file based on the DLDMOD syntax as described in this manual.
3. Compile the .FMT file to a .MEN file with the following COMPILE SAMPLE. If there are no errors, SAMPLE.MEN will be created.
4. Use the MAKEEXE.BAT file to create SAMPLE.EXE by typing MAKEEXE SAMPLE. The resulting SAMPLE.EXE and SAMPLE.BSE (created with EDLOG) are distributed to the customer. No other files should be distributed. The MAKEEXE.BAT does the following:

   ```
   COPY /B DLDMOD.EXE + SAMPLE.MEN
   SAMPLE.EXE
   ```

When the customer runs SAMPLE.EXE it will use the responses and the SAMPLE.BSE file to create SAMPLE.DLD. This is used to program the datalogger.

THE FILENAME.FMT FILE

The structure of a FILENAME.FMT file is broken down into seven parts:

1. Subroutines used in the program
2. Variables declaration
3. Menus declaration
4. Windows Attributes
5. Text to go in each window
6. Compile section
7. Comments

1. SUBROUTINES act as they do with other programming languages. When the subroutine is called, program control is transferred to the subroutine. When the subroutine is finished, control is transferred back to the instruction following the call.
2. All variables must be declared in the FILENAME.FMT program before they are actually used. The declaration includes a variety of information, including what will appear in the dialog box, and what responses are acceptable for this option.
3. The MENUS declaration includes parameters to be put in the dialog box and what action is to be taken when this selection is chosen.
4. The WINDOWS attributes describe the basic layout of the window, such as where to place it, whether to frame it, etc.
5. The Window TEXT is the exact text that will appear in the window when the program is run. It also describes where the menu options, variables, and the dialog box are to be placed.
6. COMPILE SECTION uses the user's answers and generates or modifies the FILENAME.DLD code. DLDMOD can:

   a) change the Program Table Interval of Modes.
   b) change any parameter in any FILENAME.DLD program.
   c) change, insert or delete any program at any given entry number.
   d) change and insert labels.

7. COMMENT makes reading a FILENAME .FMT file a little easier. They are spread throughout the program between window text definitions and following a semicolon in declarations and executable instructions.

1. SUBROUTINES

SYNTAX:

SUBROUTINE name
instruction
END name

DESCRIPTION:

To call the subroutine, use the command GOSUB Name. After the subroutine is executed, control will be switched to the line after the line that called it.

EXAMPLE:

```
SUBROUTINE Colors
        TextColor (yellow)
        TextBackground (blue)
END Colors;
```

2. VARIABLE DECLARATIONS

SYNTAX:

VARS
name
        vartype
        size of box
        choices
        dialog box message
        default

name2

        vartype
        ...

DESCRIPTION:

Variable Name The name must be a unique string of letters ['A'..'z']. It can be up to eight characters long. Anything after that is ignored.

Vartype May be one of the following:

CHR .....{single character}
STR.....{string}
INT .....{-32768..32767}
REAL .....{1.5e-45..3.4e38}

When the program is actually run, and if the user types in something that doesn't correlate with the variable type expected, an error message is shown describing what expression is expected. The user is asked to input the answer again. If there is a default given, then it is displayed.

Size of Box — Must be in the range of 1 Max Size. This is the size of the box where the user is expected to input an answer.

Choices — A list of choices the user must choose from. As with the vartype, if the user doesn't enter something from this list, he is shown an error message. Enter "ALL" to allow any response of the correct type.

Dialog Box Message — When the program gets to this variable, the message on this line will appear in the dialog box. The message will automatically be word-wrapped to fit in the box. Enter the message on one line. To put a hard return in the message, break the two lines into two string literals separated by commas. Two commas in a row means a double return. See example AM32 below.

Default — The answer automatically shown on screen. If the FILENAME.FMT file was used before, the user is asked if the responses that were last given would like be used, or begin again with the default answers the FILENAME.FMT program supplied. The word "none" on this line means the box will originally show up blank.

```
VARS
DataloggerType
        STR
        6
        `21x', `cr10', `cr7'
        `TMS supports either a 21x, CR10, or CR7
        datalogger'
        none
AM32
        INT
        6
        1..4
        `Enter the number of AM32 multiplexors to be
        used in this test.',,`No more than four AM32s
        multiplexors can be used.'
        none
DSP4
        CHR
        6
        `Y', `N'
        `TMS will support a DSP4 with any datalogger.'
        `Y'
```

3. MENU DECLARATION

SYNTAX:

OPTIONS
option_name
instruction to execute if selected
dialog box message
option2
instruction to execute if selected
...

DESCRIPTION:

Name of Option — Write in the exact text you want highlighted if the user is ready to select this option.

Executable Instruction — Instruction to execute when menu is selected. Same as compiling instruction. Usually either DISPLAY or GOSUB instruction.

Dialog Box Message — Same as variable dialog box message. Appears in dialog box when user highlights this option.

EXAMPLE:

```
OPTIONS
Select Hardware
DISPLAY Hardware_Selection
`First thing you must do to run TMS.'

Show Measurements
Display Measurements
`Second thing you must do when running TMS.'

Show Parameters
DISPLAY Test_Parameters
`Third thing you must do when running TMS.'

Create ReportsDISPLAY Reports
`Fourth thing you must do when running TMS.'

sensor A
Display SensorA
`'

sensor B
display SensorB
```

4. WINDOW ATTRIBUTES

SYNTAX:

```
Windows
name
size
attribute
executable instr

name_2
size
...

Subwindows
name_3
size
attribute
executable instr

name_4
size
...
```

DESCRIPTION:

The SIZE of the window may be declared as "full" (takes up the whole screen) or in the following format:

left marg, top marg, right marg, bottom marg

Any of the following attributes may be selected by including it on the line.

Frame	A double frame will appear around the window when active.
Dialogbox	Draw a dialog box in this window.
Clearback	Determines how long the window will remain visible. With clearback enabled, the window will be removed after the first menu option is selected, as if the user pressed the ESC key. This option is normally used on subwindows. When clearback is used on a main window, that window is removed and the next window is displayed. If no more main windows exist, execution passes to the compile section. Pressing ESC will show the aborted window on any of the main windows.
Rdonly	If enabled, it will prevent the user from editing any of the variables or selecting any menu bars within the window. This option is useful for summary screens, etc.
Help	This declaration will cause the window to be displayed when the user presses F1. Normally this will be a HELP window. Only one window should use the HELP declaration. A default HELP window is provided if no window uses the HELP declaration. See the section on HELP.
Executable Instruction	Instruction to execute before the window is displayed. Enter "Nothing" if an instruction is not needed.

EXAMPLE:

```
WINDOWS
Start                ;firstwindow
Full
Frame, Clearback
nothing

Main                 ;main window
1,1,80,24
Frame, Dialogbox
GOSUB Colors

SUBWINDOWS
Userhelp             ;User defined help screen
10, 5, 70, 15
Help, Frame
nothing

Summary              ;summary window
1, 1, 80, 24
Clearback, Rdonly
nothing
```

Windows in the window section will be displayed in the order listed. Windows in the subwindow section are not displayed unless a "DISPLAY window_name" command is used. The exception to this is a window declared with the HELP attribute set, which is displayed when F1 is pressed.

5. WINDOW TEXT SECTION

SYNTAX:

TEXT window name
text to appear in window
.STOP

TEXT window name2
text to appear in window
.STOP

DESCRIPTION:

Windows must be declared before the TEXT is declared. All declared windows must have a TEXT section. Within the text, type a carat mark (^) and the name of the variable everywhere you want to use a variable (must be previously declared). Everywhere you want a menu option to appear, type in a tilde mark (~) followed with the name of the menu option and another tilde mark. Mark the two corners of the dialog box with @DB.

EXAMPLE:

```
TEXT Initialize

 * Before going out to collect data, the user defines
   the following:

        1. ~Select Datalogger~
        2. ~Select Hardware~
        3. ~Select Measurements~
        4. ~Defining Reports~
                        HELP
 @DB
                                @DB
.STOP

TEXT Hardware_Selection
                                        HELP
                        @DB
    AM32:   ^AM32

   AM416:   ^AM416

SM192/716: ^SM

    DSP4:   ^DSP4

 SDM-INT8: ^SDM

                                        @DB
 .STOP
```

6. THE COMPILE SECTION

SYNTAX:

Compile
instruction ;comment
instruction1
...
End

DESCRIPTION:

Execution of the instructions in the Compile section begins when the user presses the F4 key or when the last main window is finished. Before these instructions are executed, all of the variables (answers) are stored in a file. On subsequent executions of the application, DLDMOD presents a window giving the user the option of using the last answers as defaults. The user may also choose to use the original default answers. Only the last set of answers are saved and any changes to variables in the Compile section are not saved. The variables are saved in a file with the same name as the .EXE file but using a .ANS extension.

The executable instructions are described on the following pages. The Compiler is case insensitive, so it doesn't matter whether upper case or lower case letters are used. Commands, unless otherwise noted in the following pages, are expected to be one line each. Anything following a semicolon (outside of string literals) on a command line is ignored. The semicolon is optional.

Executable Commands:

Instruction	Structure or Example
File	
DldFile	DLDFILE = `example';
Save	SAVE
Entries/programs	
ChangeTo	CHANGETO 5 AT 1:4:2;
Delete	DELETE 1:3;
Insert	INSERT 2:5,2,1 AT 1:3;
Renum	RENUM;
Labels	
InsLabel	INSLABEL `measure' AT 4;
Label	LABEL lbl1,,lbl2 AT 235;
Comments	
Comment	COMMENT `Version 1' AT 4;
Program Control	
If Then Else	IF (Reps = 1) THEN
	ELSE
	ENDIF
Variables	
Inc	INC (number, step)
Dec	DEC (number, step)
Set	SET: variable = v
Readonly	READONLY (variable)+

Windows
Display.....DISPLAY Window 5
Print.....PRINT`PRN':Hookup1,A

Colors
TextColor.....TextColor (yellow)
TextBackground..TextBackground (blue)
BorderColor.....BorderColor (black)
RespColor.....RespColor(blue)
RespBackGrnd....RespBackGrnd (yellow)
FrameColor .....FrameColor (red)
HelpText.....HelpText (red)
HelpBack.....HelpBack (white)
HelpFrame.....HelpFrame (blue)
MenuColor.....MenuColor (red)
MenuBack.....MenuBack (blue)
VarColor .....VarColor (black)
VarBack.....VarBack (white)
RoColor .....RoColor (cyan)
RoBack.....RoBack (green)

Misc.
ClrWin.....ClrWin
RemoveWin.....RemoveWin
Type.....Type `Please wait'
TypeIn.....TypeIn `system is working'

FILE

DLDFILE defines which FILENAME.DLD file to use. To have DLDMOD edit an already existing FILENAME.DLD file, use the expression:

DLDFILE = filename

To have DLDMOD create a new FILENAME.DLD file, use the expression:

DLDFILE = NEW dldname, datalogger type

EXAMPLE:

```
DldFile = `example';----------{will edit the file example.bse file};

DldFile = new `example', `21x'; --{will create a new file called example.dld for the 21X};

DldFile = new name, dtalggr;-----{will create a new file using the string stored in name with the datalogger type stored in dtalggr};
```

If no datalogger type is specified when creating a new FILENAME.DLD file, it defaults to CR10. Note that it is unnecessary to specify the datalogger type when editing an existing file.

DLDMOD will attempt to load a file with the extension .BSE if no extension is specified.

A runtime error (NOT a compile error) will occur if a file specified for editing (i.e. not as NEW) is not found. For this reason, it is good practice to use the DLDFILE command as the executable instruction for the first window displayed (or as soon as possible). Otherwise, the user may complete the entire windows section only to have the program abort with an error because the specified file was not found.

SAVE When the DLDFILE is finished being editing, use the SAVE command to save the changes.

EXAMPLE:

```
Save; ----------{Saves the file as the name given with the DldFile = command. If no extension was specified with dldfile= then .dld is used.}

Save `Rain2'; ----------{Saves the file as Rain2.Dld. };

Save `Rain2.ddd'; --------{Saves the file as Rain2.ddd};

Save rain2;----------{Saves the file as the name given to the variable rain2};
```

The default extension for saving is .DLD. DLDMOD will overwrite any existing files without prompting when told to do so.

It is good practice to NOT overwrite the original file, but to save the changed file under a different name. This allows the user to start over at any time.

PROGRAMS / ENTRIES

CHANGETO To change a specified mode, parameter, program, or table interval, use the expression:

CHANGETO xxxx AT mode#:entry#:parameter#;

EXAMPLE:

```
Changeto 45 at 1; ----------{changes the first program table interval to 45 seconds};

Changeto 95 at 1:6; ----------{changes the 6th program of table 1 to 95};

Changeto -16 at 1:5:4; ----------{changes the fourth parameter of the fifth entry of table 1 to 16--. If the fourth parameter had not already existed in the fifth entry, then it is inserted at the correct place with the correct parameter.};

Changeto tc1 at wh:wh2:wh3; ---{numeric variables can also be used};
```

DELETE xx:yy : deletes the yyth entry in the xxth mode. No other entry numbers are changed.

EXAMPLE:

```
Delete 1:5;----------{will delete the fifth entry in
                      the first mode};
```

INSERT xx AT y:zz : inserts xx program at the zz entry in mode y.

EXAMPLE:

```
Insert 5 at 1:3;---------- {will insert program 5 at the
                            third entry of table one. If
                            there is already a program in
                            that spot, then DLDMOD will
                            insert it as the 2.001 entry. If
                            there is already a 2.001 entry,
                            then it will insert the program
                            as 2.002 and so forth.};
INSERT 15:2,3 AT 2:36;------- {inserts P15 ( with 2 as the
                               first parameter and 3 as the
                               second) as the 36th spot of
                               table 2.};
INSERT tc1 AT tc2:tc3;-------- {make sure these variables
                               hold numbers};
```

RENUM This is used for cleaning up the FILENAME.DLD program after you're done changing it. It renumbers the entries by integers. Programs don't necessarily need to be renumbered in order to run. Rather, the Renum program makes the program easier to read.

EXAMPLE:

```
Renum;
```

LABELS

INSLABEL Inserts label(s) at a specific location. If there is already a non-blank label at the given location, then it and the non-blank labels following it are moved over until there are enough blank labels to compensate for the inserted label(s).

EXAMPLE:

```
INSLABEL `NewLabel' AT 4;-----{will take the first set of la-
                               bels and change it to the
                               second set of labels};

;:Cntr   :DF2   mV:Vx    mV:Batt  V  :Pulse ch2
;:Pulse ch1:_____:_____:Fixed #1 :Fixed #2

;:Cntr   :DF2   mV:Vx    mV:NewLabel :Batt  V
;:Pulse ch2:Pulse ch1:_____:Fixed #1 :Fixed #2
```

Label follows the same rules as the InsLabel command as far as inserting more than one label at a time, inserting repetitive labels, using variables, and inserting at which locations. The only exception is:

```
Label `Cntr',,`Pulse ch1' at 3;----{will change the 3rd 5th la-
                                   bels, leaving the 4th label
                                   alone};
```

```
InsLabel `Cntr',,`Pulse ch1' at 3; {will NOT work, and will
                                    flash an error when
                                    compiled};
```

Otherwise:

```
InsLabel `Cntr' at 236; ---------- {inserts the label Cntr at
                                    236 and moves the follow-
                                    ing labels over as needed};

InsLabel `Cntr';---------- {inserts Cntr as the first la-
                            bel};

InsLabel `Cntr',varLabel,`' at 5;- {inserts the label Cntr as
                                    the fifth label, value in
                                    varLabel as the sixth, and
                                    a blank label in the
                                    seventh. The eighth label
                                    on down are moved over
                                    as needed};

InsLabel `test' #[5..236] at 10; --- {inserts test#5 to test#236
                                      starting at the 10th
                                      location. All following
                                      labels are moved forward
                                      as needed};
```

LABEL Changes a label or labels at a given location.

EXAMPLE:

```
Label `Cntr' at 236;---------- {changes the 236th label
                                to Cntr};

Label `Cntr';---------- {changes the first label to
                        Cntr};

Label `Cntr',`DF2  mV',,`Vx  mV'; {changes the first,
                                   second, and fourth labels
                                   to labels indicated. The
                                   third label is ignored};

Label `Cntr',varLabel,`' at 5; ---- {changes the fifth label to
                                    Cntr, the sixth label to the
                                    value stored in varLabel,
                                    and deletes the seventh la-
                                    bel, replacing it with a
                                    blank label};

Label `Fixed ' #[1..5], varLabel #[1..10] at 5;  {changes
                                    the first 15 labels, the first 5
                                    being Fixed #1 to Fixed #5
                                    and the next 10 being the
                                    value of varLabel
                                    numbered one to 10};
```

7. COMMENTS

COMMENT Adds comments to the .DLD file. Useful for documentation purposes. Comments within a .DLD file are always preceded by a (;). The COMMENT command also adds a (~) character to differentiate comments it adds from other comments (e.g., labels). When modifying comments, only those with the (~) character are counted or replaced. This instruction will overwrite an existing comment at the specified location.

EXAMPLE:

```
COMMENT `Version 1.2' AT 1
COMMENT Name AT 2  ; {name is a string variable}
```

PROGRAM CONTROL

IF THEN ELSE The If Then Else works similarly to other high-level languages, looking like:

```
IF (boolean expression) THEN
      executable instruction(s)
 ELSE
      executable instruction(s)
 ENDIF
```

EXAMPLE:

```
If (TcTemp = `S') then
  Display SingleEnded;
endif;

If (Temp = `C') then
  set: mult = 1;
  set: offset  = 0;
else
  set: mult = 1.8;
  set: offset = 32;
endif;

If (Reps > 14) then
  Print Hookup1;
else
IF (reps < 5) then
  Print Hookup2;
endif;
endif;
```

VARIABLES

INC and DEC Increment and decrement (respectively) any variable, v, by step counts. Step is an optional parameter, and if not specified, the variable will be incremented or decremented by one.

SYNTAX:

INC (v , step) or INC (v)
DEC (v , step) or DEC (v)

EXAMPLE:

```
inc (number, 2); ---------- {number := number + 2};
inc (number);------------------- {number := number + 1};
dec (number, 3); ---------- {number := number - 3};
dec (number);------------------- {number := number - 1};
```

SET Sets a variable equal to another variable, a constant, or expression.

SET: variable = expression

EXAMPLE:

```
Set: reps = 12;---------------------{sets the variable reps
                                     equal to 12.};
Set: reps = rep2;-------------------- {sets the variable reps
                                     equal to the value stored in
                                     rep2};
Set: name = 'Bob'
```

Math is also allowed in the numeric expressions. Concatenation is allowed with string variables. The following operators are allowed:

+	addition for numeric, concatenation for string.
-	subtraction (unary minus also allowed)
*	multiply
/	divide
%	modulo
^	exponential
()	parentheses (used to alter precedence)

EXAMPLE:

```
Set: loc = loc * reps
Set: temp = offset * (val3 + val2)
Set: lastname = name + 'Smith'
```

READONLY Marks a variable as READONLY or as NOT READONLY. A variable marked as READONLY will be displayed normally on the screen except it can't be highlighted or edited by the user (i.e. no edit box, and the cursor will skip the affected variable when cursor is moved). All variables default to NOT READONLY.

SYNTAX:

Readonly (variable) option

Valid options are:

(+) makes the variable READONLY.
(–) makes the variable NOT READONLY.

The READONLY attribute of variables has no effect on the SET command. Only the window display and editing are affected.

WINDOWS

DISPLAY This command displays a window on the screen for the user. Once the user is finished with the window, the window is removed and the previous window is restored.

SYNTAX:

DISPLAY windowname

EXAMPLE:

```
Display W1;
```

PRINT This command prints windows to either a text file or directly to the printer. The file can

be appended to or newly created. The command structure is:

PRINT `filename' : windowname, option;

If you want to print directly to the printer, type PRN as the file name.

Valid options are:

A	Append to existing file, create file if it does not exist.
O	Overwrite any existing file, create file if it does not exist.

Either option can be used for direct printer output.

EXAMPLE:

```
Print PRN : Intro,A ;----------{prints the windows Intro,
                               SessionA, and Hookup1 to
                               the printer};

Print `HookUp.Prn' : SessionA, O ;   {prints the window
                               SessionA to the text file
                               `hookup.prn' overwriting
                               `hookup.prn if it exist.};

Print CustmrName : SessionA,A;   {appends the
                               window SessionA to a text
                               file named the value stored
                               in variable CustmrName};
```

If the printer (PRN) is selected but is not ready, a run time warning is given and the program continues. Nothing is printed, but otherwise execution is normal.

COLORS

TEXTCOLOR through **HELPFRAME** The writer of the FILENAME.FMT program may change the various colors the user will see via these instructions. Each of these instructions requires a color as its parameter. Possible colors are:

black	dark gray
blue	light blue
green	light green
cyan	light cyan
red	light red
magenta	light magenta
brown	yellow
light gray	white

CLRWIN Clears the active window or clears the screen if no windows are displayed. The active window is not removed; it is only cleared.

REMOVEWIN Removes the active window. If the active window is the last window (not a SUBWINDOW) then the windows section is left as if F4 had been pressed.

TYPE Writes the quoted string to the active window. Writes to the screen if no windows are active. No carriage return is sent at the end of the line so subsequent writes will be on the same line.

TYPELN Same as Type, only it places a carriage return at the end of the line.

HELP

The default pop-up HELP window tells the user about the following keys:

F1	Display help screen.
F3	Leaves the program, abandoning all of the changes that were made unless the program is already in the Compile section. If in the Compile section, changes may or may not be saved. The program asks if the user would like to quit.
F4	Leaves the window section and begins the Compile section.
Cursor Movements	The arrow keys, HOME, END, PAGE UP, and PAGE DOWN keys will move the cursor to the different responses. The ESC key removes the current window.

HELP window can be changed by the developer. If a window is declared with the HELP attribute set, the text in it will be displayed instead of the default message. See the section on window declarations.