

User Manual



**TekTMS
Instrument Front Panel Developers
S3FT100
070-7835-02**

This document supports software version 2.5.

Please check for change information at the rear of this manual.

First Printing: October 1992

Tektronix, Inc., P.O. Box 500, Beaverton, OR 97077

Printed in U.S.A.

Copyright © Tektronix, Inc., 1992. All rights reserved. Tektronix products are covered by U.S. and foreign patents, issued and pending. The following are registered trademarks: TEKTRONIX, TEK, TEKPROBE, TEKTMS, and SCOPE-MOBILE.

Microsoft and MS-DOS are registered trademarks and Windows is a trademark of Microsoft Corporation

IBM and PC AT are registered trademarks of International Business Machines Corporation.

SOFTWARE WARRANTY SUMMARY

Tektronix warrants that its software products will conform to the specifications in the documentation provided with the product, when used properly in the specified operating environment, for a period of three (3) months. The warranty period begins on the date of shipment, except that if the program is installed by Tektronix, the warranty period begins on the date of installation or one month after the date of shipment, whichever is earlier. If the software product does not conform as warranted, Tektronix will provide the remedial services as described in the documentation provided with the product.

For products offered without documentation, Tektronix warrants that the media on which the software product is furnished and the encoding of the programs on the media will be free from defects in materials and workmanship for a period of three (3) months from the date of shipment. If any such medium or encoding proves defective during the warranty period, Tektronix will provide a replacement in exchange for the defective medium. Except as to the media on which the software product is furnished, the software product is provided "as is" without warranty of any kind, either express or implied.

Tektronix does not warrant that the functions contained in any software product will meet Customer's requirements or that the operation of the programs will be uninterrupted or error-free.

In order to obtain service under this warranty, Customer must notify Tektronix of the defect before the expiration of the warranty period and, for warranted products, make suitable arrangements for such service in accordance with the instructions received from Tektronix. If Tektronix is unable, within a reasonable time after receipt of such notice, to provide remedial service for warranted products or, for "as is" products, to provide a replacement that is free from defects in materials and workmanship, Customer may terminate the license for the software product and return the software product and any associated materials for credit or refund.

The above warranties shall not apply to any software product that has been modified or altered by Customer. Tektronix shall not be obligated to furnish service under this warranty with respect to any software product a) that is used in an operating environment other than that specified or in a manner inconsistent with the User Manual and documentation; or b) when the software product has been integrated with other software if the result of such integration increases the time or difficulty of analyzing or servicing the software product or the problems ascribed in the software product.

THE ABOVE WARRANTIES ARE GIVEN BY TEKTRONIX WITH RESPECT TO THE LISTED PRODUCTS IN LIEU OF ANY OTHER WARRANTIES, EXPRESS OR IMPLIED. TEKTRONIX AND ITS VENDORS DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. TEKTRONIX' RESPONSIBILITY TO PROVIDE REMEDIAL SERVICE WHEN SPECIFIED, REPLACE DEFECTIVE MEDIA, OR REFUND CUSTOMER'S PAYMENT, AS APPLICABLE, IS THE SOLE AND EXCLUSIVE REMEDY PROVIDED TO CUSTOMER FOR BREACH OF EITHER WARRANTY. TEKTRONIX AND ITS VENDORS WILL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IRRESPECTIVE OF WHETHER TEKTRONIX OR THE VENDOR HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Preface

The objective of this manual is to explain the development of instrument driver (.ISD files) or scripts, used to define front panel displays for the Interactive Procedure Generator (IPG). These front panel displays show the controls you want to manipulate during the running of an IPG test procedure. This manual also gives instructions for adding help messages, using the IPG Help Editor, for use with the scripts.

Instrument Script Language (ISL) is used to develop the scripts. Any word processing editor that will produce ASCII output can be used to generate script files containing the script statements.

About this Manual

This manual contains the following sections and appendices:

- Section 1, General Information — Contains an overview of the topics covered in this manual.
- Section 2, Instrument Script Language Description — Describes the different parts of scripts, and what they contain and do.
- Section 3, Writing Scripts — Gives a tutorial on writing scripts.
- Section 4, Using the Help Editor — Gives tutorials on using the Help Editor.
- Appendix A, *TDM5120.ISD* Script Printout — Gives a printout of the *TDM5120.ISD* script file used in the tutorials.

Related IPG Documentation

The Interactive Procedure Generator Users Manual, provided with the IPG package, provides information about developing procedures using scripts.

Contents

Preface	
About this Manual	
Related IPG Documentation	
Contents	

Section 1 General Information

General Information	1-1
User Requirements	1-1
Overview	1-1
Interactive Views	1-2
Program Generation Views	1-2
Front Panel View Optimization	1-3
Setting, Learn, and Measurement Control Functions	1-3
Actual vs Composite Controls	1-4
Updating Features	1-5
Front Panel Appearance	1-5
Front Panel Helps	1-5

Section 2 Instrument Script Language Descriptions

Instrument Script Language Descriptions	2-1
Script Basics	2-1
The Framework	2-2
General	2-2
Script Block	2-4
BUSNOTE Block	2-4
GPIB Parameters	2-5
RS232 Parameters	2-6
VX5520 Bus Parameters	2-6
VXI Internal Bus Parameters	2-7
MXI Bus Parameters	2-7
CDS 53 Series Parameters	2-7
LEARN Block	2-8
VIEW Block	2-9
Multiple Views	2-10
Display Types	2-10

View Layout	2-10
Coordinate System	2-11
Text or Connect Statement	2-11
CONTROLGROUP Block	2-12
Control Block	2-13
Control ID	2-13
CONTROLGROUP Types	2-15
Textbox	2-17
Editbox	2-18
Listbox	2-19
Pushbutton	2-20
Checkbox	2-21
Radiobutton	2-23
WaveformDisplay	2-25
Control Type Summary	2-33
SETTING and MEASUREMENT Blocks	2-35
Dialogs	2-35
Statements	2-35
IF Conditional Structure	2-36
Condition	2-36
Dialogs	2-37
Statements	2-40
Functions	2-45
Chr	2-45
TimeDelay	2-45
Display	2-46
FastDCWrite	2-46
FastDCRead	2-48
Readlength	2-49
Variable Types	2-50
Integers	2-50
Floats	2-50
Strings	2-50
Waveforms	2-50
Variable Formats	2-51
Controller Dialog Formats	2-51
Instrument Dialog Formats	2-54
Common Problems when Using Instrument Dialog Formatting	2-55
ISL Keywords	2-57

Section 3 Writing Scripts

Writing Scripts	3-1
Introduction	3-1
The Editor	3-1
References	3-1
The Script Model	3-2
Script Planning	3-3
Charting A Front Panel	3-3
Determining Control Types	3-3
Interchangeability	3-5
Script Development Procedures	3-6
Script and Description Presentation	3-6
Many-To-One Control Groups	3-19
Testing Scripts	3-27
Initializing the Script	3-27
Activating Controls	3-28
A Rejected Script	3-28
Generating Test Procedures	3-29
Modifying Scripts	3-29
Binary ISD Files	3-31
Using Binary	3-31
Using Binary ISD Files	3-33

Section 4 Help Editor

Help Editor	4-1
Learning to Use the Help Editor	4-1
Starting the Help Editor Program	4-2
Adding and Linking a Message	4-9
Linking	4-10
Creating a New Message	4-12
Deleting a Message	4-14
Ending an Edit Session	4-15
Making a New ISD Help Message File	4-15
Message Link Map	4-17
Message Form	4-19
Printing a *.HLP File	4-20

Appendices

Appendix A: TDM5120.ISD Script Listing	A-1
Appendix B: Software Performance Report	A-9

Index

Change Information

Section 1

General Information

General Information

This manual describes and explains the Instrument Script Language, its structure, syntax, contents and use for developing scripts. A script performs two basic functions:

- Defines the graphical appearance of a logical instrument window for an instrument. This capability enables user interaction.
- Defines how each control in the logical instrument view interacts with the hardware (i.e., instruments on the bus, and the bus itself).

Also included in this manual are instructions for using the Help Editor (an IPG utility program). The Help Editor allows the user to incorporate help messages within the script.

User Requirements

Developing scripts does not require considerable programming skills, but some knowledge of programming is desirable. You should also have knowledge of PC DOS and a text editor or word processing program that exports ASCII text.

It will be helpful to refer to the *Interactive Procedure Generator Users Manual* for instructions on how the scripts you write will be used to develop test procedures.

Overview

Scripts serve two purposes: First, they provide an interactive, graphical, instrument front panel view for user interaction with the instrument while using the Interactive Procedure Generator (IPG); second, scripts control the flow of information between a test program and the instrument when running an IPG test procedure.

A script may contain only a subset of the instrument functionality. This approach is useful if you are using only some of the instrument functions, or if you desire a simplified approach. Also, any number of identical instruments can use (copies of) the same script. With IPG you can call scripts for use as you generate test procedures.

Front panel views provided by scripts have two general uses: Interactive and Program Generation (see Figure 1-1). We can further classify front panel views into two general implementations: Fully Functional and Application Oriented.

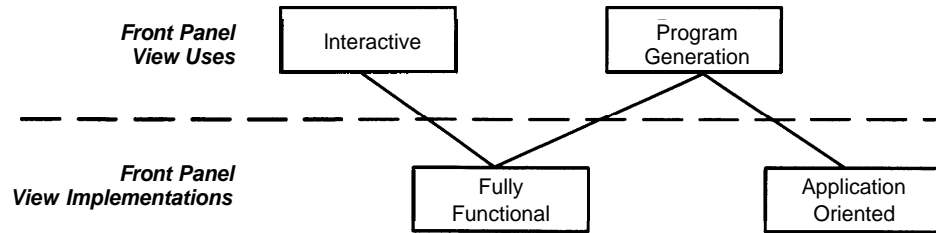


Figure 1-1: Categorization of Front Panel Views

Interactive Views

Interactive front panel views are generally intended to be used for real-time testing under full-time operator control, using fully interfaced, on-line instruments. These front panel views normally would be fully functional implementations, and contain a complete, or nearly complete, set of instrument controls and/or functions.

Program Generation Views

Program Generation front panel views are generally intended to be used to create automated test procedures that may be run unattended in real-time applications when started by an operator, or automatically at a time specified by program control. These front panel views may be fully functional or application oriented implementations depending upon the specific requirements of the test.

Fully functional view implementations, as previously indicated, support the full range of an instrument's control functions.

Application oriented implementations can be simply defined as tailored, customized, or optimized scripts, which would contain only those controls needed for a particular test situation. These scripts also may contain composite controls that combine several instrument functions into one application-oriented, high-level control function that does not correspond to a single instrument control.

There are several advantages to using an application oriented front panel:

- With fewer controls, it is easier for the program developer or test operator to find and select a control for a particular application.
- With only relevant controls would reduce the overall size of the view. (using less memory space allows more drivers to be stored on the system).
- With only a subset of controls, it may be easier to make instrument substitutions. Instruments would be more likely to share a common subset of controls than they would total functionality.

Front Panel View Optimization

Instrument front panel drivers should be optimized whether they are used interactively or for program generation.

As indicated earlier, interactive front panels should contain controls that correspond to actual instrument front panel controls, or in the case of VXI instruments, to specific instrument functions. This criteria closely fits the definition of a fully-functional front panel, which is most likely to contain a complete set of the instrument's most commonly used controls.

In other cases, controls that are implemented on an interactive front panel may not be needed on a front panel optimized for program generation use. For example, it may be possible to adjust a signal generator frequency by pressing an INCRement or DECRement pushbutton on the instrument's front panel. While this function might be useful on an interactive front panel, it is not likely to be used in program generation, thus it probably should not be implemented on a program generation front panel.

If both extensive interactive control and program generation are required, two instrument front panel views should be created, with each optimized for its primary use. For instruments with hardware front panels, a program generation front panel view or set of program generation front panel views may be sufficient. For instruments, such as VXI instruments that do not have hardware front panels, it may be desirable to have both program generation and interactive front panels. It also may be desirable to have an interactive front panel available whenever a test operator is expected to use an instrument to troubleshoot a failed unit-under-test (UUT).

Setting, Learn, and Measurement Control Functions

NOTE

In IPG, the keywords QUERY, MEASURE, and MEASUREMENT are synonymous and fully interchangeable. Any one of them can be used in a Learn or Measurement block to query an instrument control for its current setting or measured value.

An interactive front panel driver should implement the LEARN block for each instrument, and the SETTING and MEASUREMENT blocks for every control. The LEARN block allows the user to query the instrument for its current settings, store them in a procedure step, then recall them on demand to restore the instrument to those settings. The inclusion of the MEASUREMENT block provides a means for querying the instrument for a control's current setting or measured value, which can be used to update the interactive front panel display (Also see Updating Features, following). On instruments that do not support a QUERY function (LEARN or MEASUREMENT block) for their controls, a current status query cannot be implemented for updating the front panel display.

NOTE

The instrument help message should tell the user whether or not the instrument has a Learn block.

In a program generation front panel view, query functions should be implemented only to acquire a measured value. By eliminating all other QUERY (or MEASURE/MEASUREMENT) blocks, the driver can be smaller in size, thus minimizing overhead at execution time and allowing more drivers to be stored in memory.

Actual vs Composite Controls

On an interactive front panel view, a control is more likely to correspond directly to an instrument control or command than it is on a program generation front panel. For example, a digital multimeter will typically have controls for measuring voltage and current. On an interactive front panel, these controls would be implemented as two separate displays, one for voltage and one for current. Each display would correspond to an actual instrument function.

However, if test procedures are to be developed for an application-oriented program generation front panel where a power measurement is required, the view should contain a control for displaying power rather than two displays for current and voltage. The script for this control would query the instrument for its current and voltage readings, calculate the power, and display it on a one front panel power control. In this case, there would be only one control for displaying power, which would not correspond to an actual instrument control or function.

A program generation front panel will often contain controls that do not correspond to controls on the actual instrument front panel. If a function that involves several instrument controls is used frequently in a test situation, this function could be created as a single multifunction control on the front panel. As stated earlier, calculations could be incorporated into the control functions. For example, several readings might be taken and averaged with only the averaged result displayed in a control on the front panel. In this case, only one control needs to be selected by the test procedure step to obtain the averaged result.

Updating Features

An interactive front panel may be optimized to make use of the front panel display updating features such as automatic refresh, driver control update lists, and the front panel menu Update! command.

Front Panel Appearance

The appearance of front panel displays may be enhanced by using various optimizing techniques. For example, a data display might be enhanced by converting its reading from engineering notation (5E-3) to standard suffix units notation (5 mV). This technique should be use only on interactive front panel views.

On a program generation front panel, interactive features, display enhancements, and the total number of controls may be minimized to make the panel smaller and easier to use, and to provide the minimum overhead at execution time.

Front Panel Helps

Each front panel should have a help file. This file will contain useful hints about the instrument front panel and about each control. See Section 4, Learning To Use the Help Editor for more information.

Section 2

Instrument Script

Language Descriptions

Instrument Script Language Descriptions

Script Basics

This section provides a detailed description of the block structure of the Instrument Script Language (ISL) shown in Figure 2-1. ISL is used in developing scripts. This section explains the interaction between blocks, the constructs used in blocks, construct syntax, and the ISL special functions.

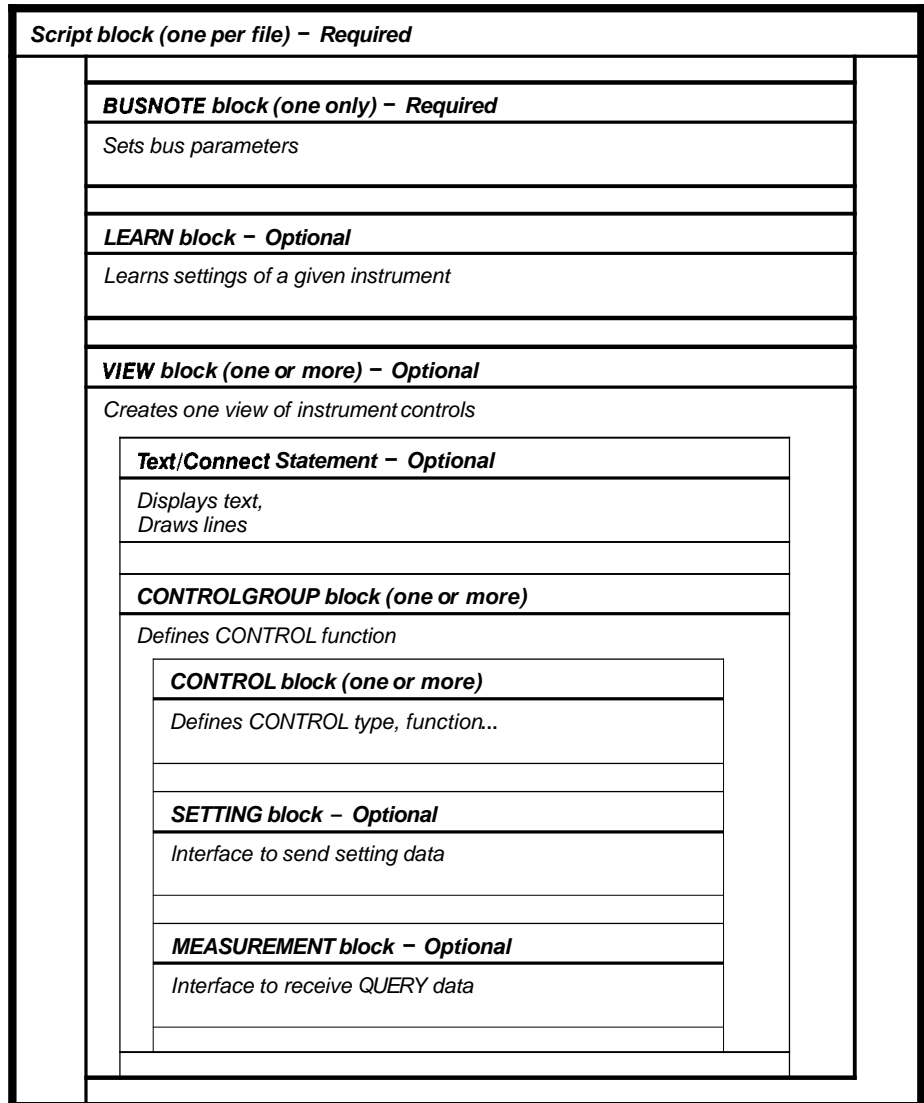


Figure 2-1: ISL Block Structure

The Framework

A script is a text file that can be created using a text editor or the front panel editor. The syntax is block structured: all blocks are identified by keywords. The termination of every block is marked with the keyword **END**. Related information is encapsulated in blocks, as shown in Figure 2-1 on the previous page. All items in a script, (e.g., keywords, variable names, etc.) are case insensitive; therefore, uppercase and lowercase may be mixed within an entry as desired (e.g., `script` is the same as `SCRIPT`).

General

A line beginning with the keyword **REM** or **REMARK** is treated as a comment. Each remark line must have REM or REMARK at its beginning.

A section may be indented and spaced in any style you want to improve readability. The syntax of the script is free format; i.e., any number of contiguous spaces, tabs, or new-lines are equivalent to a single space.

The following is a simple example of a script: it defines only one CONTROL in one view. A script can have many controls and many views, but, for introducing a complete script, this example is sufficient.

```
REM THE FOLLOWING IS AN EXAMPLE OF A SIMPLE SCRIPT TO
REM ILLUSTRATE THE VARIOUS BLOCKS USED IN A SCRIPT.
REM NOT ALL BLOCKS ARE SHOWN IN THIS EXAMPLE.
REM THE TEXT, CONNECT, AND LEARN BLOCKS ARE NOT USED IN
REM THIS EXAMPLE. THEY WILL BE EXPLAINED IN OTHER
REM EXAMPLES.

REM YOU BEGIN WITH A 'SCRIPT' BLOCK. IT CONTAINS ALL
REM SCRIPTDATA.

SCRIPT "EXAMPLE1"

REM NEXT IS THE BUSNOTE BLOCK. IT TELLS WHICH
REM COMMUNICATIONS BUS TO USE AND SETS VARIOUS
REM COMMUNICATIONS PARAMETERS.

GPIB
  EOM = 10;
END

REM IF A SCRIPT HAS ANY CONTROLS, A 'VIEW' BLOCK IS
REM REQUIRED.

VIEW "CHANNELA"

REM EACH CONTROL IS IN A 'CONTROLGROUP' BLOCK. YOU CAN
REM PLACE MORE THAN ONE CONTROL IN A 'CONTROLGROUP'
REM BLOCK. YOU MUST HAVE AT LEAST ONE 'CONTROLGROUP'
REM IN A VIEW.
```

```

CONTROLGROUP
  REM  A CONTROL (THE GRAPHICAL PART) IS DEFINED IN A
  REM  'CONTROL' BLOCK. YOU ALSO DEFINE THE NAME OF THE
  REM  CONTROL VARIABLE, ITS DATA TYPE, AND ITS LOCATION
  REM  IN THE 'CONTROL' BLOCK.

CONTROL FREQUENCYA:FLOAT EDITBOX @1,2

  REM  THE SIZE, TITLE, AND DEFAULT VALUES OF A CONTROL
  REM  ARE DEFINED WITHIN THE 'CONTROL' BLOCK.

  NUMROWS 1;
  NUMCOLS 12;
  CONTROLTITLE "FREQUENCY A";
  STRING "10000";
END

  REM  A 'SETTING' BLOCK TAKES INFORMATION FROM THE
  REM  FRONT PANEL OR TEST PROGRAM FOR ALL OF THE C
  REM  ONTROL VARIABLES DEFINED IN A 'CONTROLGROUP'
  REM  BLOCK.

SETTING
  CONT -> "FREQA: ", FREQUENCYA, ";";
END

  REM  A 'MEASUREMENT' BLOCK TAKES INFORMATION FROM THE
  REM  CONTROL VARIABLES DEFINED IN A 'CONTROLGROUP'
  REM  BLOCK AND RETURNS IT TO THE FRONT PANEL OR TEST
  REM  PROGRAM.

MEASUREMENT
  CONT -> "FREQA?";
  INST -> "FREQA: ", FREQUENCYA, ";";
END
END
END
END

```

The example shows some of the syntax points. All lines beginning with the keyword REM are comments.

Script Block

The SCRIPT block is the complete script (or it can be defined as the outermost block containing a script). The keywords for this block are **SCRIPT** and **END**. SCRIPT is followed by the script identifier. The script identifier is a character string enclosed within double quotes.

The SCRIPT block, in its entirety, communicates with the instrument as a driver. A script block consists of a number of internal blocks: the *BUSNOTE* block, the *LEARN* block, and the *VIEW* block. The *LEARN* block and *VIEW* block are optional, as shown in the following example script:

```
SCRIPT "SCRIPTID1"  
  
GPIB  
  EOM = 13,10;  
  EOI = 1;  
  TIMEOUT = 1000;  
END  
END
```

This example defines a complete script. There are no controls defined, but a simple script such as this one can be used to establish a communications path between a test program and an instrument for TRANSFER DATA steps in an IPG test procedure (i.e., VARIABLE TO INSTRUMENT or INSTRUMENT TO FILE transfers). Refer to Section 5, Advanced Applications, in the Interactive Procedure Generator Users Manual, under the Communicating with an Instrument which does not have a Driver (.ISD) File topic for additional information. The example script can also be used to gain access to the TALW LISTEN function of a front panel. Refer to the *Talk/Listen* description in Section 4, Menus, of the Interactive Procedure Generator User Manual, under the Instrument Front Panel Menu description, for further information.

BUSNOTE Block

The purpose of the BUSNOTE block is to select the bus type and advise the script interpreter of interface setting parameters for the corresponding bus. This subsection describes the parameters and the corresponding settings that can be made within the BUSNOTE block. Only one bus (GPIB, VXI, etc.) can be used. The valid keywords for a BUSNOTE block and their definitions are as follows:

- **GPIB** — National Instruments GPIB interface for the PC
- **VXIINTERNAL** — Tektronix and Colorado Data Systems embedded VXI controllers
- **VX5520** — Tektronix VX5520 GPIB interface and Resource Manager for VXI
- **RS232** — COM1 or COM2 ports on the PC
- **MXI** — National Instruments MXI PC to VXI interface.
- **CDSBUS** — Colorado Data System 53 Series embedded controllers.

The general syntax for a BUSNOTE block is shown in the following example:

```
GPIB

EOM = number, number;
EOI = 1 (YES) | 0 (NO);
TIMEOUT = number;

END
```

Note that the equal sign (=) and the semicolon (;) are required in the syntax. There is a different set of parameters for each bus type. The parameters for each bus type and their description follows.

GPIB Parameters

EOM (End Of Message) — This parameter specifies a character or characters automatically appended to the end of a device-dependent message on output. The characters to be appended are the ASCII equivalent of the numbers specified in the argument. EOM characters found in the input from an instrument will terminate the Read. The End of Message syntax is as follows:

```
EOM = n, m;           0 <= n <= 255
                      0 <= m <= 255
```

The following are examples of EOM:

```
EOM = 0;              (turns off EOM function)
EOM = 10;             (LF termination)
EOM = 13, 10;        (CR/LF termination)
```

The above examples illustrate three possibilities: zero (OFF); one character specified; and two characters specified.

EOI (End Of Input) — This parameter determines if EOI is asserted on the last byte of messages sent by the controller to the instrument, or by the instrument to the controller. The following examples show the use of EOI:

```
EOI = 1;              (EOI is asserted)
EOI = 0;              (EOI is not asserted)
```

TIMEOUT — This parameter is specified in milliseconds. It determines the wait period before a timeout is reported. Since some GPIB control software does not support a continuous range, the script interpreter will use the closest value that is at least as great as the specified value. The following example shows the use of TIMEOUT:

```
TIMEOUT = 1E4;        (TIMEOUT value of 10 seconds)
```

Default SETTINGS — The default settings for the parameters EOM, EOI and TIMEOUT are as follows:

```
EOM = 0;  
EOI = 1;  
TIMEOUT = 10000;
```

RS232 Parameters

```
RS232  
EOM = 0;  
BAUD = number; (300,600, 1200 ... 9600,19200)  
PARITY = 0 (EVEN) | 1 (ODD) | 2 (NONE) ;  
FLOWCONTROL = 0 (NONE) | 1 (XONIXOFF)) | 2 (DTR/DSR —  
                hardware) | 3 (RTS/CTS);  
STOPBITS = 1 | 2 | 3 (is 1.5);  
DATABITS = 4 | 5 | 6 | 7 | 8;  
END
```

Default SETTINGS — The default settings for the RS232 parameters are as follows:

```
BAUD = 1200;  
PARITY = 2; (NONE)  
FLOWCONTROL = 1; (XON/XOFF)  
STOPBITS = 1;  
DATABITS = 7;  
EOM = 0;
```

VX5520 Bus Parameters

The parameters for the VX5520 Interface are the same as GPIB (substitute VX5520 for GPIB) as in the following example script:

```
VX5520  
EOM = 13,10;  
EOI = 1; (1 = YES, 0 = NO)  
TIMEOUT = 1E4; (TIMEOUT value of 10 seconds)  
END
```

Default SETTINGS — The default settings for the VX5520 Bus parameters are as follows:

```
EOM = 0;  
EOI = 1;  
TIMEOUT = 10000;
```

NOTE

It is important to use a *BUSNOTE* type of *VX5520* if you are using a *VX5520* as the *VXI* Resource Manager and a *GPiB* interface to *VXI* instruments.

VXI Internal Bus Parameters

The parameters for VXI internal bus are shown in the following example:

```
VXIINTERNAL
  EOM = 0;
END
```

Default SETTINGS — The default settings for the VXI Internal Bus parameters are as follows:

```
EOM = 0;
```

MXI Bus Parameters

The parameters for MXI bus are shown in the following example:

```
MXI
  EOM = 10;
  TIMEOUT = 10000;
END
```

Default SETTINGS — The default settings for the MXI Bus parameters are as follows:

```
EOM = 0;
TIMEOUT = 10000;
```

CDS 53 Series Parameters

The parameters for CDS 53 Series parameters are shown in the following example:

```
CDSBUS
  EOM = 13,10;
  TIMEOUT = 5000;
END
```

Default SETTINGS — The default settings for the CDS 53 Series bus parameters are as follows:

```
EOM = 0;  
TIMEOUT = 10000;
```

LEARN Block

The LEARN block provides a way to create IPG test procedure steps that will reset an instrument to a state you specify on subsequent executions of the test procedure. You do this simply by creating a LEARN step that contains a SETTING block and a MEASUREMENT block.

The SETTING block is executed when a test procedure with a LEARNED SETTING step is executed.

The MEASUREMENT block is executed when requested by the user during test program creation in IPG.

The LEARN SETTING block does not describe a graphical, interactive control, but is used to store the instrument state in a test program for later retrieval.

The following examples show the use of the LEARN block:

```
REM THIS EXAMPLE SHOWS AN ASCII LEARN SETTING BLOCK FOR  
REM A TEKTRONIX DM5010 DIGITAL MULTIMETER. THERE IS NO  
REM GRAPHICAL DESCRIPTION, AS THIS CONTROL DOES NOT  
REM APPEAR ON THE SCREEN.
```

```
LEARN LEARNSTR:STR ASCII  
SETTING  
    REM SEND THE ENTIRE LEARNED STATE STRING TO THE  
    REM INSTRUMENT.  
    CONT -> LEARNSTR;  
END  
MEASUREMENT  
    REM QUERY THE INSTRUMENT FOR ITS ENTIRE SETTING STATE  
    REM AND STORE THE DATA IN THE CONTROL VARIABLE  
    REM LEARNSTR.  
    CONT -> "SET?";  
    INST -> LEARNSTR;  
END  
END
```

```
REM THIS EXAMPLE SHOWS A BINARY LEARN SETTING BLOCK FOR  
REM A TEKTRONIX DM5010 DIGITAL MULTIMETER. THERE IS NO  
REM GRAPHICAL DESCRIPTION, AS THIS CONTROL DOES NOT  
REM APPEAR ON THE SCREEN.
```

```
LEARN LEARNSTR:STR BINARY  
SETTING  
    REM SEND THE ENTIRE LEARNED STATE STRING TO THE  
    REM INSTRUMENT.
```

```

CONT -> LEARNSTR ;
END
MEASUREMENT
REM QUERY THE INSTRUMENT FOR ITS ENTIRE SETTING STATE
REM AS A BINARY STRING (THIS IS SHORTER THAN AN ASCII
REM STRING) AND STORE THE DATA IN THE CONTROL VARIABLE
REM LEARNSTR .
CONT -> "LLSET?" ;
INST -> LEARNSTR ;
END
END

```

The difference between an ASCII LEARN block and a BINARY LEARN block, is that you can edit the contents of the ASCII string when creating a LEARNED SETTING step, but you cannot edit the contents of a BINARY string.

When one of the example scripts is used, the MEASUREMENT block gets a measurement from an active instrument and places it into the variable specified in the script. When a LEARN SETTING step is created in the IPG test procedure and the test procedure is run, only the SETTING block is executed, allowing you to set the instrument.

The LEARN block is used with the **Learn SETTING...** menu item from the **Step** menu selection in the IPG Instruments Front Panel (for further information, refer to the *Instrument Front Panel Menu* description in *Section 4, Menus*, of the IPG Users Manual).

VIEW Block

The keywords for a VIEW block are **VIEW** and **END**. The keyword VIEW is followed by the *VIEW identifier*. The VIEW identifier is a character string enclosed within double quotes. In a script, there may be any number of VIEW blocks. One VIEW block corresponds to one logical instrument view.

The VIEW block consists of a number of statements and blocks. The first entries may be **TEXT** or **CONNECT** statements (or both). The blocks in the VIEW block are **CONTROLGROUP** blocks.

A logical instrument view contains graphical controls such as **PUSHBUTTONs**, **CHECK BOXES**, etc. These graphical controls allow interaction with the hardware and the bus.

There are several approaches to assigning functions to different views (The issue of fully functional vs. application-oriented front panels is explained in *Section 1, General Information*):

- Don't put too many controls into one View as it creates the same problems as when actual hardware front panels have too many controls in a small space. Keep the View simple.
- A fully functional or general purpose approach might parcel views by the actual partitioned functionality of the instrument front panel (e.g., a DSO might have a Channel 1 view, a Channel 2 view, a Timebase view, etc.).

An application-oriented approach might put all the functions needed to perform a particular action in one view, and those for another action in another view. (e.g., in a DSO, all the functions required to acquire a waveform on channel 1 might be presented in one view, and all those needed to acquire a waveform on channel 2 might be presented in another view. The time base and trigger functions would be included as required and duplicated on the two views).

- On an interactive front panel, a group of measurements might be placed in a different view from control settings so that when the measurements are updated, the controls are not updated.
- Implementation of a set of controls might be simplified by using several views. For example, if the DM5120 had one range control that controlled the ranges for all functions, it might be included in a view for each function (DCV, ACV, OHMS, etc.), and be given an appropriate label corresponding to the function associated with the view.
- Don't build a view that requires the entire display screen (maximized).

Multiple Views

Multiple views in the same script should use approximately the same surface area for the view layout to preclude the user resizing the window to see all of the controls each time a new view is selected. When views are not the same size, the first view (default) should be the largest so the user doesn't accidentally miss controls that might be outside the window of the initial display area. Also any controls that are duplicated from view to view should be laid out in the same relative position.

Display Types

If an instrument front panel has been developed for an EGA display and it fills a large part of the display, it may not be completely displayed on a CGA display. Also, a VGA display provides a smaller amount of display area than a comparable EGA display. The user should be aware that different graphics adapters provide different display resolutions, and if a front panel is to be used on systems with different graphics adapters, it should be designed for the system with the lowest resolution (smallest display area).

View Layout

All controls must fit within a 27 character high by 80 character wide area (Instrument Script Language character coordinates). This assures that all controls are visible on the smallest available window display, which is the 640 x 480 VGA mode that provides 640 x 410 pixels in the white space of a maximized front panel window. A 640 x 480 VGA mode displays characters in 8W x 15H cells, which reduces the number of rows of characters that can be displayed. Controls should be spaced far enough apart on a front panel so that they are perceived as separate controls. Each control should be surrounded by several characters of space to promote "readabil-

ity" of the panel. You may find it easier to layout the front panel to scale on graph paper before implementing it in a script. The units for front panel layout are the number of characters from the top left corner of the display.

Coordinate System

The IPG software uses character spaces to set coordinates. Starting from the top left corner of your display, the coordinates are 0,0. They move right horizontally to the maximum number of columns; and down vertically to the maximum number of rows. Coordinates can be entered into the script, or can be function values. The coordinates can also be decimal numbers. For example, you can place a control at 3.3,4.5, if desired. You can use several digits of precision if you desire, but tenths of character sizes are usually adequate.

TEXT or Connect Statement

The **TEXT** or **CONNECT** statement (either is started with a separate keyword) can appear in a VIEW block. A TEXT or CONNECT statement is used to display text or to draw lines on the screen of the logical instrument front panel view. The body of each statement may contain one or more sets of TEXT and CONNECT points. TEXT or CONNECT statements must come before CONTROLGROUP blocks.

Following are examples of TEXT and CONNECT statements:

```
TEXT 'text to be displayed' @ 5,5;  
CONNECT (5,10),(10,10);
```

This example defines text to be displayed beginning at character spaces (5, 5); and specifies drawing a line between display coordinates (character space points) 5, 10 and 10, 10.

More than one TEXT or CONNECT statement can be included in a VIEW block.

NOTE

Notice that these statements require a terminating semicolon.

TEXT — The following syntax displays one line of text:

```
TEXT "stringof text" @ X,y;
```

As an example:

```
TEXT "THIS IS TEXT" @ 5, 10;
```

This example causes THIS IS TEXT to be displayed, beginning at character space location 5, 10.

Connect — The following syntax draws a line between end points:

```
CONNECT (X1, Y1), (X2, Y2) [, (X3, Y3), (X4, Y4) [, ...]];
```

where the x and y values of the points are given in character spaces. They are relative to the upper left hand corner of the user window, where x = 0, and y = 0.

Examples:

```
CONNECT (5,10), (10,10);
```

This example draws a line between the two points: (5, 10) and (10, 10).

```
CONNECT (3,3), (10,3), (10,10), (3,10), (3,3);
```

This example causes a line to be drawn from the point (3, 3) to (10, 3) then to (10, 10) to (3, 10) and then back to (3, 3). The result is a rectangle.

CONTROLGROUP Block

A CONTROLGROUP block is a collection of at least one CONTROL block and, optionally, one SETTING and one MEASUREMENT block that define the control functions in an instrument view. The CONTROLGROUP must contain at least one CONTROL block, but it may have more than one. A CONTROLGROUP block without either a SETTING or MEASUREMENT block will not be able to send data to or receive data from an instrument. Each CONTROLGROUP block specified has the ability to define a SETTING and/or a MEASUREMENT block to perform when the control is manipulated interactively or used in a test program. In a SETTING block, data is taken from the front panel or test procedure and sent to the SETTING block. In a MEASUREMENT block, data is gathered from the instrument and returned to the front panel or test procedure.

It is important to remember that SETTING blocks accept data and MEASUREMENT blocks return data. SETTING blocks do not update the front panel control, while MEASUREMENT blocks do not use the value of the front panel control in the measurement.

Another important consideration about scripts is that they do not execute sequentially. They do not start at the first line and execute to the last line of the script. Each control definition will execute either the SETTING or MEASUREMENT block independently, depending on the level of front panel to instrument interaction, type of instrument step executed, and which controls are manipulated.

The following is an example of a CONTROLGROUP block with CONTROL, SETTING, and MEASUREMENT blocks defined:

```
CONTROLGROUP
CONTROL FREQUENCYA:FLOAT EDITBOX A2,2
CONTROLTITLE "FREQUENCY A";
STRING "10000";
NUMCOLS 12;
NUMROWS 1;
```



```

END
SETTING
  CONT -> "FREQA: ", FREQUENCYA, ";";
END
MEASUREMENT
  CONT -> "FREQA?";
  INST -> "FREQA: ", FREQUENCYA, ";";
END
END

```

In this example, when the front panel is in "full Interaction" mode, and the ENTER keyboard key is press when the input focus is on the EDITBOX control, the SETTING block will send the string `FREQA: 10000;` to the instrument. If the control is saved as a SETTING step in a test procedure, and the value of the control was 10000 when the step was saved, the same string will be sent to the instrument when this step is executed. The value of the CONTROL variable `FREQUENCYA` will be set to 10000 and the SETTING block will be executed.

If the front panel is in "full interaction" mode when the ENTER key is pressed, the SETTING block is executed, and then the MEASUREMENT block is executed which updates the value of the EDITBOX to reflect the value returned from the instrument in response to the `FREQA?` query. If the response from the instrument is `FREQA: 10000.0;`, then the front panel EDITBOX will be updated with the value `10000.0`.

More detailed information about the various blocks and instrument dialogs is presented later in this section. For further information about the front panel window, refer to the Interactive Procedure Generator Users Manual.

Control Block

Since a CONTROL block describes only one control, each control on an instrument front panel display must have a separate CONTROL block. The first line of a CONTROL block contains the keyword CONTROL followed by a CONTROL ID (control variable), a colon (:), a variable type (INTeger (a long), FLOAT (a double), or STRing), the control type (TEXTBOX, EDITBOX, etc.), and the controls reference position (relative X, Y coordinate).

A CONTROL variable name is limited to 15 alphanumeric characters, and must begin with an alpha character. The following example defines a TEXTBOX control at location 2, 2 0 that accepts and shows integer data. `COUNTA` is the name of the CONTROL.:

```
CONTROL COUNTA:INT TEXTBOX @2,20
```

Control ID

Once you declare a Control ID, it becomes a global variable that can be accessed by any subsequent SETTING or QUERY block (MEASUREMENT block) in the script. A common use of this global variable feature is for defining a control that can display an error message for an operator using

the front panel interactively. The following example script makes use of this feature by defining a control to display an error message that is defined and implemented by another control.

```
SCRIPT "SCRIPT1"  
  GPIB  
  END  
  VIEW "VIEW1"  
    REM DEFINE THE ERROR DISPLAY CONTROL.  
    CONTROLGROUP  
      CONTROL ERRMSG:STR TEXTBOX @ 1, 2  
        TITLE "ERROR MESSAGE";  
        NUMROWS 1; NUMCOLS 20;  
      END  
      QUERY  
        REM FORCE AN UPDATE OF THE CONTROL VALUE.  
        ERRMSG = ERRMSG;  
      END  
    END  
    CONTROLGROUP  
      CONTROL TWO:INT CHECKBOX @ 3, 5  
        STRING "CLICK HERE";  
        REM CAUSE THE ERRMSG QUERY BLOCK TO EXECUTE.  
        UPDATELIST ERRMSG;  
      END  
    SET  
      REM GIVE THE ERROR MESSAGE A VALUE.  
      ERRMSG = "ERROR 0";  
    END  
  END  
END
```

In this example, a TEXTBOX control to display an error message is declared using the Control ID ERRMSG and front panel TITLE "ERROR MESSAGE". This error message display control must be the first control defined in the VIEW or script, so that any errors detected in subsequent steps can be detected.

Next, a CHECKBOX control using the Control ID TWO and front panel title "Click Here" is declared to cause the system to locate the MEASUREMENT step for ERRMSG, the measurement to be taken, and execute the assignment of the error message and the updating of the TEXTBOX display.

The QUERY block is required., since **UPDATELIST** only updates blocks that contain a QUERY block. If it is missing, the control will not be updated by the UPDATELIST ERRMSG statement.

Script Design Considerations— There is a restriction that must be considered when one control sets or reads the value of the Control ID of another control. When writing a script, a Control ID can be set only if it has been previously defined with a CONTROL statement earlier in the script.

CONTROLGROUP Types

Each CONTROL block defines a single control, and defines the variable and variable type used for communications with that control.

There are several types of front panel controls. The CONTROL block defines one of the controls and control data types listed in Table 2-1.

Table 2-1: CONTROL Block

CONTROL	CONTROL Data Type
TEXTBOX	INT, STR, FLOAT
EDITBOX	INT, STR, FLOAT
LISTBOX	INT, STR
CHECKBOX	INT, STR
RADIOBUTTON	INT, STR
PUSHBUTTON	INT, STR
WAVEFORMDISPLAY	STR, WAVE

The INTeger value for a CHECKBOX, RADIOBUTTON, or PUSHBUTTON is either 1 (ON) or 0 (OFF). The string value for these controls can be any string necessary for the function of the control SETTING or MEASUREMENT blocks.

Data values for a TEXTBOX or EDITBOX can be any value required by the control function.

The INTeger value for a LISTBOX is the index of the selected LISTBOX item. The indices run from 1 to the number of entries in the LISTBOX. The string value of the LISTBOX is either the string displayed in the LISTBOX, or the corresponding TOSCRIPt string.

CONTROLGROUP Type Construct Definitions — Table 2-2 defines the control constructs used in the following CONTROLGROUP type discussions.

Table 2-2: Construct Definitions

Name	Syntax	Description
CONTROL-TITLE	CONTROLTITLE " <i>titlestring</i> ";	<i>titlestring</i> is the string to be displayed above the TEXTBOX, EDITBOX, or LISTBOX (no default)
DEFAULTPOS	DEFAULTPOS <i>n</i> ;	<i>n</i> is the default selected row in a LISTBOX (default 1)
NUMCOLS	NUMCOLS <i>n</i> ;	<i>n</i> is the number of columns wide of the TEXTBOX, EDITBOX, or LISTBOX (default 10)

Table 2-2: Construct Definitions (Cont.)

Name	Syntax	Description
NUMROWS	NUMROWS <i>n</i> ;	<i>n</i> is the number of rows high of the TEXTBOX, EDITBOX, or LISTBOX (default 1)
ONEOF-GROUP	ONEOFGROUP " <i>groupname</i> "	<i>groupname</i> is the name of the group of controls to which a CHECKBOX or RADIOBUTTON belongs (no default)
SCROLL-HORZ	SCROLLHORZ YES< <i>N</i> > NO;	YES specifies a horizontal scroll bar for a TEXTBOX, EDITBOX, or LISTBOX; NO specifies no horizontal scroll bar (default NO)
SCROLLVERT	SCROLLVERT YES< <i>N</i> > NO;	YES specifies a vertical scroll bar for a TEXTBOX, EDITBOX, or LISTBOX; NO specifies no vertical scroll bar (default NO)
STATE	STATE ON< <i>N</i> > OFF;	The default state of a CHECKBOX or RADIOBUTTON (default OFF)
STRING	STRING " <i>datastring</i> ";	<i>datastring</i> is the string to display in the TEXTBOX, EDITBOX, or LISTBOX; or the string to be displayed as the label of the CHECKBOX, PUSHBUTTON, or RADIOBUTTON (no default)
TITLE	TITLE " <i>titlestring</i> ";	<i>titlestring</i> is the string to be displayed above the TEXTBOX, EDITBOX, or LISTBOX (no default)
TOSCRIPPT	TOSCRIPPT " <i>string</i> ,[" <i>string</i> ",[...]]	<i>string</i> is the string that will be placed in the CONTROL variable (~) in a LISTBOX
TOSCRIPPT-OFF	TOSCRIPPTOFF " <i>offstring</i> ";	<i>offstring</i> is a string that will be placed in the CONTROL variable when the CHECKBOX or RADIOBUTTON is turned OFF or deactivated (no default)
TOSCRIPPTON	TOSCRIPPTON " <i>datastring</i> ";	<i>datastring</i> is a string that will be placed in the CONTROL variable when the PUSHBUTTON is manipulated (no default)
	TOSCRIPPTON " <i>onstring</i> ";	<i>onstring</i> is a string that will be placed in the CONTROL variable when the CHECKBOX or RADIOBUTTON is turned ON or activated (no default)

Table 2-2: Construct Definitions (Cont.)

Name	Syntax	Description
UPDATELIST	UPDATELIST <i>controlid</i> [, <i>controlid</i> [,]];	controlid is the name of another control in this view to update when the control whose script contains the UPDATELIST is manipulated. To update a control, the MEASUREMENT block actions for the control whose script contains the UPDATELIST are executed. (no default)

TEXTBOX

A **TEXTBOX** is a rectangular area in which text is displayed and can be used for measurement displays on a front panel. If you do not need to modify the measurement value reported from the instruments, use a TEXTBOX. The following illustration is an example TEXTBOX graphic.



The following script was used to create the example TEXTBOX graphic:

```

CONTROLGROUP
CONTROL ID:STR TEXTBOX @ 5 , 10
  NUMROWS 1 ;
  NUMCOLS 10 ;
  SCROLLHORZ YES ;
  CONTROLTITLE "ID:" ;
END
MEASUREMENT
  CONT -> "ID?" ;
  INST -> ID ;
END
END

```

In the above example, a CONTROL block with a TEXTBOX is defined with its number of columns and rows. The TEXTBOX has a horizontal scroll bar and a control title ID: ; and is empty. When the front panel is updated, the TEXTBOX will display the contents of the control variable ID, which is the output returned by the MEASUREMENT block. The variable and type are declared with the control type (in the above example CONTROL ID:STR TEXTBOX @ 5 , 10).

The available constructs in a TEXTBOX are:

CONTROLTITLE
NUMCOLS
NUMROWS
SCROLLHORZ
SCROLLVERT
STRING
TITLE

The definitions of these constructs are given in Table 2-2, in this section.

The variable type can be STR, INT, or FLOAT.

EDITBOX

An **EDITBOX** is a rectangular area in which you can enter and edit text and can be used for setting and reporting numeric values for an instrument (i.e., the frequency of a signal generator). The following illustration is an example of the EDITBOX graphic.



The following script was used to create the example EDITBOX graphic:

```
CONTROLGROUP
CONTROL NUM:FLOAT EDIT @ 10, 10
  NUMROWS 1;
  NUMCOLS 23;
  CONTROLTITLE "AVE:";
  SCROLLHORZ YES;
END
SETTING
  CONT ->"AVE", NUM, ",";
END
MEASUREMENT
  CONT ->"AVE?";
  INST ->"AVE", NUM, ",";
END
END
```

The example defines an **EDITBOX** with its number of columns and rows. It has a horizontal scroll bar and the title **AVE:** displayed above the box. Initially, the **EDITBOX** is empty. The variable and type are declared with the control type (in the example, CONTROL NUM:FLOAT EDIT @ 10, 10).

When the SETTING block is evaluated, the number in the EDITBOX is assigned to the CONTROL variable NUM. The number is encoded with the string constant and sent to the instrument. For example, if NUM = 100, then "AVE 100" is sent.

When the MEASUREMENT block is evaluated, the number to be displayed in the EDITBOX is taken from the CONTROL variable NUM. The instrument is queried for the value of AVE and its string in the form "AVE NUM";. The value of NUM is extracted from the instrument response.

The available constructs for an EDITBOX are:

CONTROLTITLE	SCROLLVERT
NUMCOLS	STRING
NUMROWS	TITLE
SCROLLHORZ	UPDATELIST

The definitions of these constructs are given immediately following the RADIOBUTTON description, in this section.

The variable type can be STR, INT, or FLOAT

LISTBOX

A LISTBOX is a scrollable list of strings representing instrument controls or measurements and can be used to give the front panel user a choice of a limited number of settings for an instrument. The following illustration is an example of a LISTBOX graphic giving a choice of operating modes for a digital multimeter:



The following script is used to create the example LISTBOX graphic:

```

CONTROLGROUP
CONTROL PARAM1:STR LISTBOX @ 5, 10
  NUMROWS 3;
  NUMCOLS 8;
  CONTROLTITLE "FUNCTION:";
  SCROLLVERT YES;
  STRING "DCV", "OHMS", "DIODE", "ACV", "ACDC";
END
SETTING
  CONT ->PARAM1;
END
MEASUREMENT
  CONT ->"FUNC?";
  INST ->"FUNC", PARAM1, ",";

```

```
END
END
```

The example defines a LISTBOX with the number of columns, rows, a vertical scroll bar, a title `FUNCTION` and the strings displayed in the LISTBOX. The variable and type are declared with the control type (in the example, `CONTROL PARAM1:STR LISTBOX @ 5, 10`).

When the SETTING block associated with a LISTBOX is evaluated, the string selected in the LISTBOX is assigned to the control variable `PARAM1`.

When the MEASUREMENT block is evaluated, the string to be highlighted will be taken from the control variable `PARAM1`. If the variable does not match any string in the STRING block, no string in the LISTBOX will be highlighted. The match is case sensitive.

The available constructs for a LISTBOX are:

CONTROLTITLE	SCROLLVERT
DEFAULTPOS	STRING
NUMCOLS	TITLE
NUMROWS	UPDATELIST

And, if the control type is STR:

TOSCRIPT

The definitions of these constructs are given immediately following the RADIOBUTTON description, in this section.

The variable type can be STR or INT.

PUSHBUTTON

A Pushbutton is a box that contains a string representing an action and is used for actions that require no response (i.e., the initialization function of an instrument). A Pushbutton has only a SETTING block, and when it is manipulated (pushed) is always in the ON state.

Following is an example of a PUSHBUTTON graphic:



The following script was used to create the example PUSHBUTTON graphic:

```
CONTROLGROUP
CONTROL INIT:STR PUSHBUTTON @5,15
  STRING "INIT";
  TOSCRIPTON "INIT";
END
SETTING
  CONT ->INIT;
END
END
```


This example script defines a PUSHBUTTON that will have the text INIT inside it. The variable and type are declared with the control type (in the example, CONTROL INIT:STR PUSHBUTTON @5,15).

When the SETTING block associated with the PUSHBUTTON is evaluated, the string defined by TOSCRIPTON is assigned to the control variable.

In the following alternative example, the PUSHBUTTON uses a control variable of the type INT.

```
CONTROLGROUP
CONTROL INIT:INT PUSHBUTTON @5,15
  STRING "INIT";
END
SETTING
  CONT ->"INIT"
END
END
```

This example script also defines a PUSHBUTTON that will have the text INIT inside it. Again, the variable and type are declared with the control type (in the example, CONTROL INIT:STR PUSHBUTTON @5,15). This example takes advantage of the fact that when a PUSHBUTTON is manipulated, the value of the control variable is always 1, and simply sends the constant string INIT to the instrument.

A PUSHBUTTON should not have a MEASUREMENT block. If it does, the MEASUREMENT block will not be evaluated.

The available constructs for a PUSHBUTTON are:

```
STRING
UPDATELIST
```

And, if the control type is STR:

```
TOSCRIPTON
```

The definitions of these constructs are given in Table 2-2, in this section.

The variable type can be STR or INT.

CHECKBOX

A **CHECKBOX** is a small square with an identifying character string to the right. When a CHECKBOX is activated, an "x" appears in the square. A CHECKBOX is used to indicate the state of a control, either ON or OFF. A CHECKBOX might be used to indicate whether the output of a signal generator is turned ON or OFF.

Following is an example of a CHECKBOX graphic:



The following script was used to create the example CHECKBOX graphic:

```
CONTROLGROUP
CONTROL LFR:STR CHECKBOX @10,20
  STRING "LFR";
  TOSCRIPTON "LFR ON";
  TOSCRIPTOFF "LFR OFF";
END
SETTING
  CONT ->LFR;
END
MEASUREMENT
  CONT ->"LFR?";
  INST ->LFR, ",";
END
END
```

This example defines a CHECKBOX with the text LFR displayed next to it. The variable and type are declared with the control type (i.e., CONTROL LFR:STR CHECKBOX @ 10,20).

When the SETTING block is evaluated, if the CHECKBOX is activated, the string defined by the key word **TOSCRIPTON** is assigned to its control variable; if the CHECKBOX is not activated, the string defined by the key word **TOSCRIPTOFF** is assigned to the control variable.

When the MEASUREMENT block is evaluated, the control variable will be used to update the state of the CHECKBOX. If the control variable matches the string defined by the key word **TOSCRIPTON**, the CHECKBOX will be activated; if it does not match, the CHECKBOX will not be activated.

The following alternative example of a CHECKBOX script uses a control variable of the type INIT:

```
CONTROLGROUP
CONTROL LFR:INT CHECKBOX @5,15
  STRING "LFR";
END
SETTING
  IF (LFR == 1) THEN
    CONT -> "LFR ON";
  ELSE
    CONT -> "LFR OFF";
  ENDIF
END
MEASUREMENT
  TEMPVAR TEMPSTR:STR;
  CONT -> "LFR?";
  INST -> TEMPSTR, ",";
  IF (TEMPSTR == "LFR ON") THEN
    LFR = 1
  ELSE
    LFR = 0
  ENDIF
```

```
END
END
```

This example also defines a **CHECKBOX** with the text **LFR** displayed next to it. The variable and type are declared with the control type (i.e., `CONTROL LFR:INT CHECKBOX @ 5,15`).

This example takes advantage of the fact that when a **CHECKBOX** is manipulated, the value of the control variable is 1 if the **CHECKBOX** is activated, and 0 if it is not activated. The **SETTING** block checks the value of the control variable and sends the appropriate command to the instrument. The **MEASUREMENT** block checks the value of the query and sets the control variable to either 1 (ON) or 0 (OFF). Notice that the **MEASUREMENT** block uses a temporary string variable to help with the comparison.

The available constructs for a **CHECKBOX** are:

```
ONEOFGROUP
STATE
STRING
UPDATELIST
```

And, if the control type is **STR**:

```
TOSCRIPTOFF
TOSCRIPTON
```

The definitions of these constructs are given immediately in Table 2-2, in this section.

The variable type can be **STR** or **INT**.

RADIOBUTTON

A **RADIOBUTTON** is a small circle with a character string to the right of the circle. When selected, the circle contains a black dot. **RADIOBUTTONS** are used only in groups to represent mutually exclusive selections. A **RADIOBUTTON** is used to indicate the state of a control, either **ON** or **OFF**. A group of **RADIOBUTTONS** show the state of mutually exclusive instrument controls or states. For example, a group of **RADIOBUTTONS** could be used to show which function of a function generator is currently active (i.e., sine, square, triangle, etc.).

Following is an example of the **RADIOBUTTON** graphic:



The following script was used to create the example **RADIOBUTTON** graphic:

```
CONTROLGROUP
CONTROL LFR:INT RADIOBUTTON @10,20
STRING "LFR";
```

```

    REM RADIOBUTTONS MUST CONTAIN A ONEOFGROUP DEFINITION.
    ONEOFGROUP "FREQ";
END
SETTING
  IF (LFR == 1) THEN
    CONT ->"LFR ON";
  ELSE
    CONT ->"LFR OFF";
  ENDIF
END
MEASUREMENT
  TEMPVAR FROMINST:STR;
  CONT ->"LFR?";
  INST ->FROMINST;
  IF (FROMINST == "LFR ON;") THEN
    LFR = 1;
  ELSE
    LFR = 0;
  ENDIF
END
END

```

The example defines a **RADIOBUTTON** with the text `LFR` next to it. The variable and type are declared with the control type (i.e., `CONTROL LFR:INT RADIOBUTTON @ 10, 20`).

When the **SETTING** block is evaluated, if the **RADIOBUTTON** is ON, a 1 is assigned to the control variable; if the **RADIOBUTTON** is OFF, a 0 is assigned to the control variable.

When the **MEASUREMENT** block is evaluated, the control variable is used to update the state of the **RADIOBUTTON**. If the output variable is 1, the **RADIOBUTTON** will be ON; otherwise, the **RADIOBUTTON** will be OFF.

NOTE

The *RADIOBUTTON* being turned OFF has its **SETTING** block executed first; then the **SETTING** block of the button being turned ON is executed.

The available constructs for a **RADIOBUTTON** are:

```

ONEOFGROUP
STATE
STRING
UPDATELIST

```

And, if the control type is STR:

```

TOSCRIPTOFF
TOSCRIPTON

```

The definitions for these constructs are on the following page.

The variable type can be STR or INT.

If a RADIOBUTTON is type STRING (STR) then the TOSCRIPTON and TOSCRIPTOFF constructs must be used.

WAVEFORMDISPLAY

The **WAVEFORMDISPLAY** control shows a variable of the type WAVEFORM and gives you control over how the waveform is acquired and viewed.

The ISD Script Language contains two functions, **WAVEFORMTOADIF** and **WAVEFORMTOVAR**, to convert most instrument specific data formats into a WAVEFORM variable that can be used and displayed by TekTMS. The **WAVEFORMDISPLAY** control does not appear on the screen, but appears as an icon at the bottom of the screen when an instrument view containing a waveform control is shown. The display can be viewed by expanding the icon. If no waveform has been captured the display field is blank. For further information regarding waveform display, refer to the Interface Program Generator Users Manual. The contents of the variable cannot be modified by the display.

The data type of a **WAVEFORMDISPLAY** control is either WAVE or STR. The variable and type are declared with the control type. If you use the data type of WAVE you must use the function **WAVEFORMTOVAR** to convert the instrument specific data. If you use the data type of STR, you must use the function **WAVEFORMTOADIF** to convert the instrument specific data. The control string is the name of the file that contains the acquired waveform data. The preferred type to use is WAVE. The STR type is mainly for compatibility with previous versions of TekTMS. With the WAVE type you can also capture XY or Envelope data from an instrument and put it into the WAVE variable.

Unlike other controls, the position specified for the **WAVEFORMDISPLAY** graphic is relative to the upper left hand corner of the front panel, since the graphic is a separate window rather than part of the front panel.

Figure 2-2 is an example of a **WAVEFORMDISPLAY** graphic.

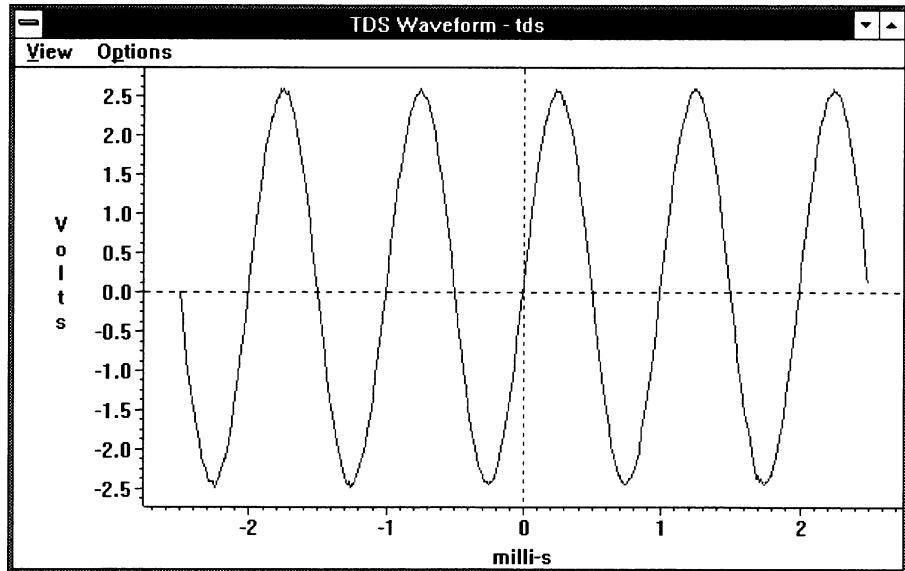


Figure 2-2: Example of a **WAVEFORMDISPLAY** Graphic

The only available construct for a **WAVEFORMDISPLAY** control is **CONTROL-TITLE** or **TITLE**.

The following formulas are used to calculate the real value of the point:

- Implicit dimension: $value[n] = scale * n + offset$ where n goes from 0 to points-1.
- Explicit dimension: $value[n] = scale * data[n] + offset$ where $data[n]$ is n th data point on that dimension.

WAVEFORMTOVAR Function — The following paragraphs describe how to use the **WAVEFORMTOVAR** function in IPG.

The **WAVEFORMTOVAR** function converts raw digitized waveform data into a waveform variable.

WAVEFORMTOVAR takes a number of parameters that determine the actions taken by the routine. The **SCRIPT** invocation is shown in the following example:

```
WAVEFORMTOVAR (INFILE, INFORMAT, WAVEORDER, INTYPE,
               XSCALE, XOFFSET, YSCALE, YOFFSET, POINTS, BITS, BYTES,
               BYTEORDER, READOFFSET, BYTEFORMAT, XLABEL, YLABEL);
```

The Parameters are defined as follows:

- **INFILE** — A string containing the name of the input raw digitized waveform data.

INFORMAT — An integer which defines the input format. The possible formats are:

- 0 ASCII format
- 1 Binary Signed format.
- 2 Binary unsigned format.

■ **WAVEORDER** — An integer indicating the order that data is stored in the input file — this is used only for xy and env inTypes.

- 0 by Tuple (the data is organized so that the data points are listed together i.e. x1y1x2y2)
- 1 by Dimension (the data for the first dimension is given then the next)

■ **INTYPE** — An integer which defines the type of waveform input. The possible formats are:

- 0 yt (normal waveform — explicit data for only one dimension)
- 1 xy (explicit data for both dimensions)
- 2 env (two explicit dimensions with the same implicit dimension — max value first)
- 3 env (two explicit dimensions with the same implicit dimension — min value first)

■ **XSCALE** — A floating point number giving the X scale factor used to convert the raw data into usable information.

■ **XOFFSET** — A floating point number giving the X offset to add to the raw data.

■ **YSCALE** — A floating point number giving the Y scale factor used to convert the raw data into usable information.

■ **YOFFSET** — A floating point number giving the Y offset to add to the raw data.

■ **POINTS** — An integer number giving the number of data points in one dimension.

■ **BITS** — An integer giving the number of bits of data per data point. This is typically the resolution of the instrument (i.e. 8 for an 8 bit digitizer, 10 for a 10 bit digitizer, 16 for Tektronix 11400 Series digitizers).

■ **BYTES** — An integer giving the number of bytes per data point in the input data. The possible values are:

- 1 one byte per data point.
- 2 two bytes per data point.
- 4 four bytes per data point. Either integer or floating point format.
- 8 eight bytes per data point. Floating point format only.

- **BYTEORDER** — An integer that specifies the order of the bytes in the data point when the input format is binary. This field is used not used when there is only one data byte per point. The possible values are:
 - 0 Least Significant Byte first. (Swapped bytes).
 - 1 Least Significant Byte last.
- **READOFFSET** — An integer giving the number of bytes to skip at the beginning of the input before attempting to read the waveform data points. This allows you to skip header information which as 'CURVE %nn' as is common with Tektronix digitizers.
- **BYTEFORMAT** — an integer which specifies the format of the input data if the input format is binary. The possible values are:
 - 4 integer format.
 - 8 floating point.
- **XLABEL** — A string containing the label for the X axis.
- **YLABEL** — A string containing the label for the Y axis.

The following WAVEFORMDISPLAY control script is from a Tektronix TDS540 DSO script:

```

CONTROLGROUP

REM THIS CONTROL IS OUTPUT ONLY, SO IT ONLY HAS A
REM MEASUREMENT BLOCK.

CONTROL WFMDISPL:WAVE WAVEFORMDISPLAY @ 2.43, 2.00
CONTROLTITLE "TDS WAVEFORM";
END

MEASUREMENT

REM DEFINE ALL OF THE TEMPORARY VARIABLES USED BY THE
REM WAVEFORMTOVAR FUNCTION.

TEMPVAR RAWFILE:STR, INMODE:INT, INTYPE:INT, XSCALE:FLOAT,
XOFF:FLOAT, YSCALE:FLOAT, YOFF:FLOAT, YZERO:FLOAT,
NUMPTS:INT, BITS:INT, BYTES:INT, BORDER:INT, ORDER:INT,
READOFF:INT, BFORMAT:INT, TRIGPTS:INT, XLABEL:STR,
YLABEL:STR;

REM SET THE INPUT FILE NAME. DATA IS READ FROM THE
REM TDS540 INTO A DATA FILE AND THE
REM RESULT OF THE WAVEFORMTOVAR FUNCTION
REM IS A WAVEFORM VARIABLE.
REM GIVE THE RAW DATA FILE ANY LEGAL NAME THAT DOES
REM NOT CONFLICT WITH NAMES TO BE USED IN TEKTMS.
REM THE VARIABLE WFMDISPS IS A CONTROL VARIABLE AND
REM CONTAINS THE STRING GIVING THE INPUT SOURCE.
REM THIS HAS BEEN SET BY A SETTING STEP OR
REM FRONT PANEL UPDATE.

```



```

RAWFILE = "~TDS.WAV";

REM SETUP THE TDS AND QUERY THE INSTRUMENT FOR THE DATA
REM REQUIRED TO CREATE THE WAVEFORM.

CONT->"DATA:SOURCE ",WFMDISPS,"";
CONT->"DATA:ENCDG ASCII";
CONT->"WFMPRE:", WFMDISPS, ":XUNIT?";
INST->"\",XLABEL,\"\"";
CONT->"WFMPRE:", WFMDISPS, ":XINCR?";
INST->XSCALE;
CONT->"WFMPRE:", WFMDISPS, ":YUNIT?";
INST->"\",YLABEL,\"\"";
CONT->"WFMPRE:", WFMDISPS, ":YMULT?";
INST->YSCALE;
CONT->"WFMPRE:", WFMDISPS, ":YZERO?";
INST->YZERO;
CONT->"WFMPRE:", WFMDISPS, ":YOFF?";
INST->YOFF;
CONT->"WFMPRE:", WFMDISPS, ":NR_PT?";
INST->NUMPTS;
CONT->"WFMPRE:", WFMDISPS, ":PT_OFF?";
INST->TRIGPTS;

REM INPUT FORMAT IS ASCII,INTYPE IS YT, NUMBER OF BITS
REM IS 8. BORDER AND BFORMAT CAN HAVE ANY VALUE.
INMODE = 0;
INTYPE = 0;
ORDER = 0;
BITS = 8;
BYTES = 1;
BORDER = 1;
BFORMAT = 4;

REM ASCII DATA STARTS AT THE BEGINNING OF THE FILE.
READOFF = 0;

REM CALCULATE THE OFFSETS BASED ON DATA FROM THE TDS.
XOFF=(XSCALE*TRIGPTS)*-1;
YOFF=YZERO-(YSCALE*YOFF);

REM QUERY FOR THE WAVEFORM DATA AND WRITE IT TO A FILE.
CONT->"CURVE?";
INST->RAWFILE:%F;

REM ASSIGN THE RESULT OF THE WAVEFORMTOVAR FUNCTION TO
REM THE CONTROL VARIABLE.

WFMDISPL = WAVEFORMTOVAR(RAWFILE, INMODE, ORDER, INTYPE,
    XSCALE, XOFF, YSCALE, YOFF, NUMPTS, BITS, BYTES, BORDER,
    READOFF, BFORMAT, XLABEL, YLABEL);

END
END

```

Note that the variable type is WAVE and that no ADIF file is written at this time. The Control can not be defined as a STR. To get an ADIF file you need to do a Variable to file transfer step using the waveform variable assigned when a measurement is done using this control.

WAVEFORMTOADIF FUNCTION — The following paragraphs describe how to use the **WAVEFORMTOADIF** function in IPG.

The WAVEFORMTOADIF function converts raw digitized waveform data into ADIF format data. It takes an input waveform file and outputs an ADIF format data file containing the waveform. This release supports ADIF *Version 1.00*.

WAVEFORMTOADIF takes a number of parameters which determine the actions taken by the routine. The script invocation is shown in the following example:

```
WAVEFORMTOADIF (INFILE, INFORMAT, OUTFILE, OUTFORMAT,  
                XSCALE, XOFFSET, YSCALE, YOFFSET, POINTS, BITS, BYTES,  
                BYTEORDER, READOFFSET, BYTEFORMAT, XLABEL, YLABEL) ;
```

The parameters are defined as follows:

- **INFILE** — A string containing the name of the input raw digitized waveform data.
- **INFORMAT** — An integer which defines the input format. The possible formats are:
 - 0 ASCII format.
 - 1 Binary signed format.
 - 2 Binary unsigned format.
- **OUTFILE** — A string containing the name of the output ADIF file.
- **OUTFORMAT** — An integer which defines the output format. The possible formats are:
 - 0 ASCII format.
 - 1 Binary format. All ADIF binary formats are signed.

The parameter OUTFORMAT is not used in Version 2.5. All ADIF files are written in binary format (32 bit floating point). This parameter is included for compatibility with previous versions of TekTMS.

- **XSCALE** — A floating point number giving the X scale factor used to convert the raw data into usable information.
- **XOFFSET** — A floating point number giving the X offset to add to the raw data.
- **YSCALE** — A floating point number giving the Y scale factor used to convert the raw data into usable information.
- **YOFFSET** — A floating point number giving the Y offset to add to the raw data.

- 8 **POINTS** — An integer giving the number of data points in the raw input data file.
- 8 **BITS** — An integer giving the number of bits of data per data point. This is typically the resolution of the instrument (i.e., 8 for an 8 bit digitizer, 10 for a 10 bit digitizer, 16 for the Tektronix 11400 Series digitizers).
- **BYTES** — An integer giving the number of bytes per data point in the input data. The possible values are:
 - 1 one byte per data point.
 - 2 two bytes per data point.
 - 4 four bytes per data point. Either integer or floating point format.
 - 8 eight bytes per data point. Floating point format only.
- **BYTEORDER** — An integer that specifies the order of the bytes in the data point when the input format is binary. The possible values are:
 - 0 Least Significant Byte first. (Swapped bytes.)
 - 1 Least Significant Byte last.
- 8 **READOFFSET** — An integer giving the number of bytes to skip at the beginning of the input file before attempting to read waveform data points. This allows you to skip header information such as "CURVE %NN" as is common with Tektronix digitizers.
- 8 **BYTEFORMAT** — An integer that specifies the format of the input data if the input format is binary. The possible values are:
 - 4 integer format.
 - 8 floating point.
- 8 **XLABEL** — A string containing the label for the X axis.
- **YLABEL** — A string containing the label for the Y axis.

The following WAVEFORMDISPLAY control script is from the Tektronix 2430A DSO script:

```

CONTROLGROUP

    REM THIS CONTROL IS OUTPUT ONLY, SO IT ONLY HAS A
    REM MEASUREMENT BLOCK.

    CONTROL WFM:STR WAVEFORMDISPLAY @2.5,2.0
    CONTROLTITLE "2400 WAVEFORM";
    END

MEASUREMENT

    REM DEFINE ALL OF THE TEMPORARY VARIABLES USED BY THE
    REM WAVEFORMTOADIF FUNCTION.
  
```

```

TEMPVAR RAWFILE:STR,ADIFFILE:DTR,INMODE:INT,
        OUTMODE:INT,XSCALE:FLOAT,XOFF:FLOAT,
        YSCALE:FLOAT,YOFF:FLOAT,NUMPTS:INT,BITS:INT,
        BYTES:INT,BORDER:INT,READOFF:INT,BFORMAT:INT,
        XLABEL:STR,YLABEL:STR;

REM SET THE INPUT AND OUTPUT FILE NAMES. DATA IS READ
REM FROM THE 2430A INTO A DATA FILE AND THE RESULT OF
REM THE WAVEFORMTOADIF FUNCTION IS A DATA FILE. GIVE
REM THE RAW DATA AN EXTENSION OF .WAV AND THE ADIF
REM FILE AN EXTENSION OF .WFD.

RAWFILE = "2400.WAV";
ADIFFILE = "2400.WFD";

REM SET THE INPUT DATA FORMAT FOR THE RAW WAVEFORM
REM DATA AS BINARY SIGNED AND THE OUTPUT AS BINARY

INMODE = 1;
OUTMODE = 1;

REM SET THE INPUT BYTE SIZE, BYTE NUMBER, ORDER, AND
REM FORMAT.

BITS = 8;
BYTES = 1;
BORDER = 1;
BFORMAT = 4;

REM SET THE READ OFFSET TO 3 BYTES (SKIP THE %NN AT
REM THE BEGINNING OF THE DATA FILE).

READOFF = 3;

REM SET THE DATA RETURN FORMAT TO JUST NUMBERS AND
REM THEN QUERY THE INSTRUMENT FOR THE DATA REQUIRED TO
REM CREATE THE ADIF FORMAT DATA.

CONT -> "PATH OFF;LONG OFF;";
CONT -> 'DATA ENC:RIB;WFMPRE? XIN,XUN,PT.0,YUN,YMU,YOF,
        NR.P';

REM READ THE SCALE, OFFSET, NUMBER OF DATA POINTS, AND
REM AXIS LABELS.

INST -> XSCALE,"",XLABEL,"",XOFF,"",YLABEL,"",
        YSCALE,"",YOFF,"",NUMPTS;

REM SCALE THE OFFSET CORRECTLY

```

```

XOFF = XOFF*XSCALE*-1;
YOFF = YSCALE*YOFF*-1;

REM QUERY FOR THE WAVEFORM DATA AND WRITE IT TO A
REM FILE.

CONT -> "CURVE?";
INST -> RAWFILE:%F;

REM TURN THE RAW INSTRUMENT DATA INTO AN ADIF FILE FOR
REM USE BY THE WAVEFORMDISPLAY AND PULSE PARAMETER
REM FUNCTIONS.

WAVEFORMTOADIF (RAWFILE, INMODE, ADIFFILE, OUTMODE, XSCALE,
                XOFF, YSCALE, YOFF, NUMPTS, BITS, BYTES,
                BORDER, READOFF, BFORMAT, XLABEL, YLABEL);
REM ASSIGN THE WAVEFORMDISPLAY CONTROL VARIABLE THE
REM NAME OF THE FILE CONTAINING THE JUST CAPTURED AND
REM CONVERTED WAVEFORM.

WFM = ADIFFILE;

END
END

```

Notice that the variable type is STR and that the variable is assigned the name of the ADIF file. The type of IPG variable associated with this control would be WAVEFORM, but in the script, it is treated as a string.

Control Type Summary

Table 2-3 is a quick reference guide showing the available control type and parameters:

Table 2-3: Control Type Summary

	CHECK BOX	EDIT BOX	LIST BOX	PUSH BUTTON	RADIO BUTTON	TEXT BOX	WAVE FORM
CONTROLTITLE		X	X			X	X
DEFAULTPOS			X				
NUMCOLS		X	X			X	
NUMROWS		X	X			X	
ONEOFGROUP	X				X		
SCROLLHORZ		X				X	
SCROLLVERT		X	X			X	

Table 2-3: Control Type Summary (Cont.)

	CHECK BOX	EDIT BOX	LIST BOX	PUSH BUTTON	RADIO BUTTON	TEXT BOX	WAVE FORM
STATE	X				X		
STRING	X	X	X	X	X	X	
TOSCRIP			X				
TOSCRIP	X			X	X		
TOSCRIP	X			X	X		
UPDATELIST	X	X	X	X	X		X

SETTING and MEASUREMENT Blocks

SETTING blocks and **MEASUREMENT** blocks are alike. Both consist of dialogs and statements. The IPG procedures can only send data to a **SETTING** block and can only receive data from a **MEASUREMENT** block. These blocks must be located at the end of their **CONTROLGROUP** blocks, and the **SETTING** block must precede a **MEASUREMENT** block

A **SETTING** or **MEASUREMENT** block can contain actions that are either dialogs or statements. The following is a list of dialogs and statements:

Dialogs

CONT -> Controller Dialog

INST -> Instrument Dialog

Statements

Assignment	General purpose expression evaluation and variable assignment
IF THEN ENDIF	IF (condition) THEN (actions) ENDIF
IF THEN ELSE ENDIF	IF (condition) THEN (actions) ELSE (actions) ENDIF
WHILE DO END	WHILE (condition) DO (actions) END
GPIB commands	Low level GPIB bus commands
Variable declarations	Definitions of variables local to a SETTING or MEASUREMENT block
Function call	Calls to various ISL functions

More information on dialogs and statements is provided in the following sections of this manual.

The dialogs and statements in a **SETTING** or **MEASUREMENT** block are executed in order, starting at the beginning of the block. In some applications, the sequential execution of all dialogs and statements may not be desired. An **IF-THEN-ELSE** statement can be used within a **SETTING** or **MEASUREMENT** block to control statement execution.

IF Conditional Structure

The syntax for an IF conditional structure is as follows:

```
IF condition THEN dialogs OR statements;  
ELSE dialogs OR statements;  
ENDIF
```

OR

```
IF condition THEN dialogs OR statements;  
ENDIF
```

The IF Statement allows decision processing. When the condition result in an IF statement is TRUE, the dialogs or statements in the initial IF statement are executed. When the condition result is FALSE, the dialogs or statements in the ELSE portion of the construct are executed, or, in the case of no ELSE, the dialogs or statements in the IF construct are bypassed without execution. Therefore, the IF Conditional Structure would be used to choose between alternative actions, based on the result of the condition.

Dialogs and statements are the heart of a script. SETTING and MEASUREMENT blocks use the dialogs and statements to communicate with and execute commands for instrument operation from the controller. Specific instrument address information comes from the **Instrument Select** information entered when an ISD file is selected. Communications parameters come from the BUSNOTE section of the script.

Condition

The syntax for condition could be any one of the following:

```
(EXPRESSION)  
(EXPRESSION RELATIONAL OPERATOR EXPRESSION)  
(STRING VARIABLE RELATIONAL OPERATOR STRING VARIABLE)
```

NOTE

Expressions used in these conditions must evaluate to a zero (FALSE) or a nonzero (TRUE) value.

Where EXPRESSION is a mathematical expression and RELATIONAL OPERATOR is a relational operator (=, >, <, >=, <=, or <>). A STRING VARIABLE is either a string constant or a variable name of type STR.

A condition is TRUE if it evaluates to a non-zero value, and FALSE if it evaluates to zero. A comparison of a string to another string evaluates to TRUE if they are identical and FALSE if they are not. The comparison is case sensitive.

Dialogs

Dialogs are used for controller and instrument instructions. Each dialog type is designed to carry out a different kind of task.

There are two dialog types: ControllerDialog and InstrumentDialog. For each dialog type there is a unique keyword.

CONT -> Controller dialog

INST -> Instrument dialog

A dialog is limited to using CONT and INST.

Controller Dialog — prepares and sends data, or the contents of a file, to the instrument. The string to be sent is built by evaluating each OUTPUT-EXPRESSION and appending the results to form the string. This string is then sent to the instrument. The text for a controller dialog consists of one or more OUTPUT_EXPRESSIONS. The format for the controller dialog is as follows:

```
CONT -> OUTPUT-EXPRESSION [ ,OUTPUT-EXPRESSION [ , . . . ] ] ;
```

Where OUTPUT-EXPRESSION contains:

- **STRING constants** — One or more ASCII characters enclosed in double quotes. "ABC", "VOLTS", and "ID?" are examples of string constants. The double quotes are required around the string. A string without quotes is interpreted as a variable reference.
- **Variables** — FREQUENCYA, CH1VOLTS, and DMMVALUE are examples of variables.
- **Variable:%format** — CH1VOLTS:%6.3G and DMMVALUE:%8.2E are examples of variables with format specifiers.
- **Expressions** — Any valid mathematical calculation, string concatenation, or function call. 2*VOLTS,VOLTSDIV/4.3, and VALUEQUERY & "VOLTS" are examples of expressions.
- The **commas** separating the OUTPUT-EXPRESSIONS and the ending semicolon are required.
- **Function calls** — You can use the SKIP(N) or the CHR(N) functions in a controller dialog.

Following are some examples of controller dialogs:

```
CONT -> "VOLTS?";
```

This example will send the string "VOLTS?" to the specified instrument.

```
CONT -> "CH1 VOLTS: ",VOLTS:%8.2G,"";
```

In this example, if the variable VOLTS contained the number 12.34, the string "CH1 VOLTS: 12.34;" will be sent to the specified instrument. Notice that there are leading spaces before the number 12.34. The 8 in the dialog is the number of spaces in the width of the format, and the 2 is the

number of digits after the decimal point. The format type `G` tells the controller dialog to output a floating point number. The `:%` characters indicate that the following character is a format descriptor.

If the variable `VOLTS` contained the number `12.3456`, the string sent would be `"CH1 VOLTS: 12.35;"`. Notice the leading spaces and the rounding of the number to fit the specified format.

More information on format types and format modifiers is given later in this section.

The following is an example of a function call in a controller dialog:

```
CONT -> "MODE BINARY ", CHR(43), CHR(45);
```

The controller dialog in this example will send the string `"MODE BINARY +-"` to the specified instrument.

Instrument Dialog — reads data from the instrument. The format for an instrument dialog is as follows:

```
INST -> INPUT-EXPRESSION [INPUT-EXPRESSION [, ...]];
```

Where an `INPUT-EXPRESSION` contains:

STRING constants — One or more ASCII characters enclosed in double quotes. `"ABC"`, `"VOLTS"`, and `"ID?"` are examples of string constants. The double quotes are required around the string. A string without quotes is interpreted as a variable reference. String constants in an instrument dialog are used to indicate exact string matches in the response from the instrument. When a string constant is found in an instrument dialog, the string read from the instrument is scanned for a match with the constant. If a match is found, the rest of the dialog takes data starting with the character after the matched string. If a match is not found, an error is reported.

- **Variables** — `FREQUENCYA`, `CH1VOLTS`, and `DMMVALUE` are examples of variables.
- **Variable:%format** — `CH1VOLTS:%6.3G` and `DMMVALUE:%8.2E` are examples of variables with format specifiers.
- **Expressions** — String concatenation. Such as `VALUEQUERY & "VOLTS"` is allowed.
- **Function calls** — You can use the `SKIP(N)` or `CHR(N)` functions in an instrument dialog.

The commas separating the `INPUT_EXPRESSIONS` and the ending semicolon are required.

Following are some examples of instrument dialogs. In each example, the instrument string read is:

```
CH1 VOLTS:12.345;CH1 AMPS:0.2345;CH1 TERM:50;
```

Where `VOLTS` and `AMPS` are `FLOATS` and `TERM` is an `INT`.

```
INST -> "VOLTS:",VOLTS;
```

In this example, the instrument dialog scans the string read from the instrument from the string "VOLTS:", then reads the following characters as a floating point number. The conversion of characters from a string into a number will stop when the ";" character after the number 12.345 is read. The variable VOLTS will contain the value 12.345 after the instrument dialog is executed.

```
INST -> "AMPS:",AMPS:%6.3G;
```

In this example, the instrument dialog will scan the string read from the instrument for the string "AMPS:", then read the following characters as a floating point number with a width of 6 characters and no more than 3 characters after the decimal point. The variable AMPS will contain the value 0.234 after the instrument dialog is executed. The next character that would be processed is the ";", because the width specifies a format of 6 characters. The character 5 is skipped.

```
INST -> "VOLTS:",VOLTS,"AMPS:",AMPS,"TERM:",TERM;
```

In this example, the instrument dialog will scan the string read from the instrument for three different strings and extract the VOLTS, AMPS, and TERM values from the string.

The following is an example of a function call in an instrument dialog:

```
INST -> SKIP(20),DMMVALUE;
```

In this example, the instrument dialog will read a string from the instrument, skipping the first 20 characters, then place the remainder of the data into the variable DMMVALUE using the default format for the data type declared for DMMVALUE.

You can use the **SKIP** characters function in an Instrument dialog to skip over unneeded characters. For example, if an instrument returns an ASCII binary representation of switch closures, then you could skip over unwanted characters to get to the one character that indicated whether or not the switch was open or closed. If an instrument with 16 switches returned a string of the format WORDF:B #B000000000000001;, to indicate that Switch 1 was closed and all other switches were open, you could read just the Switch 1 setting with the MEASUREMENT block using the SKIP characters function in the following example:

```
CONTROLGROUP
CONTROL SWITCH1:INT CHECKBOX @5,3
  STRING "1";
END
SETTING
  CONT -> "CLOSE:F1.A1"
END
MEASUREMENT
  CONT -> "WORDF: BINARY; WORD?";

  REM SKIP OVER THE 'WORDF:B #B' HEADER AND THE FIRST
```

```

REM 15 SWITCH SETTINGS TO GET TO THE DIGIT NEEDED
REM READ IT AS A DECIMAL NUMBER SO THAT 1
REM (CLOSED/TRUE) OR 0 (OPEN/FALSE) IS RETURNED.

INST -> SKIP (25), SWITCH1;%1D;
END
END

```

Statements

There are five types of statements you can use in programming SETTING and MEASUREMENT blocks. These types are:

- Temporary variable declarations
- GPIB commands
- Assignment statements
- IF statements
- WHILE statements

Temporary Variables — are the variables specified in the **TEMPVAR** statement. These variables can be used to store values during the execution of a SETTING or MEASUREMENT block. Variables declared with the **TEMPVAR** statement are local to the SETTING or MEASUREMENT block where they are declared and can not be accessed by another SETTING or MEASUREMENT block. This contrasts with control variables, which are global in a script. Temporary variables are used where it is desirable to temporarily store data without declaring a separate CONTROL block.

TEMPVAR is the temporary variable statement keyword, specifying a list of variables that are made available in the SETTING or MEASUREMENT block. The syntax of the TEMPVAR statement is:

```
TEMPVAR VARNAME:VARTYPE (, VARNAME:VARTYPE);
```

In the previous syntax definition, the statement type is TEMPVAR and VARNAME is the variable name. VARNAME is limited to 15 characters. VARNAME is followed by a colon (:), followed by VARTYPE, which is the variable type (i.e., INT, FLOAT, or STR), and must be specified for each VARNAME and separated from VARNAME by a comma. Following is an example:

```
TEMPVAR VOLTS:INT, MODE:STR;
```

In this example, the TEMPVAR statement declares variables named VOLTS, of the type <F1>INT, <F1> and MODE, of the type STR.

GPIB Commands — are meaningful only when the script is configured for an IEEE 488 compatible bus. These commands can be included in scripts that use a communications bus other than GPIB, but no action will be taken when they are executed. A GPIB command in a script with a VXIINTERNAL, MXI, CDSBUS, or RS232 BUSNOTE is treated as a NOOP (no operation) statement.

The syntax for a GPIB command is:

```
INTERFACEMSG [PARAM] ;
```

The statement TEXT for a GPIB command consists of one or more of the INTERFACEMSG messages in Table 2-4. A parameter may follow the INTERFACEMSG, if required. The INTERFACEMSG messages referred to here have been specified in IEEE 488.

Table 2-4: GPIB Commands

Command (InterfaceMsg)	Parameter (param)	Description
ATN	n	A single byte of the value n sent to the bus with attention asserted.
ATN	"STRING"	A string constant is sent to the bus with attention asserted.
DCL		Device <i>CLear</i> command sent out on the bus.
GTL		Go To Local is sent to the device.
GET		Group Execute Trigger.
IFC		<i>InterFace</i> Clear line is pulsed.
LLO		Local Lockout command is sent over the bus.
REN	TRUE	Remote <i>ENable</i> is asserted if TRUE.
REN	FALSE	Remote <i>ENable</i> is dropped if FALSE.
SDC		Selective Device Clear command is sent to the device.
TIM	num	The TIMEOUT value is redefined to be num (in milliseconds). This overrides the value specified in the Note section or the one specified by an earlier TIM command for the duration of the block containing the TIM command.
UNL		<i>UNListen</i> command is sent out on the bus.
UNT		<i>UNTalk</i> command is sent out on the bus.

Following are some GPIB command examples:

```
REN TRUE ;
ATN 10 ; (The number here represents an ASCII character.)
SDC ;
IFC ;
TIM 10 ;
```

You can perform a serial poll with the ISL special function **SERIALPOLL**. **SERIALPOLL** allows the user to create a GPIB instrument front panel control to poll and display instrument status. The format is:

```
X = SERIALPOLLO ;
```

Where x can be any integer variable.

Following is an example using **SERIALPOLL** to create a **TEXTBOX** to display status whenever an **Update!** menu item is selected in an IPG test procedure, or the automatic Refresh Rate mode is selected:

```
REM DISPLAY INSTRUMENT STATUS ON FRONT PANEL UPDATE .

CONTROLGROUP
CONTROL STATUS:INT TEXTBOX @2,3
  TITLE "STATUS" ;
END
MEASUREMENT
  STATUS = SERIALPOLL() ;
END
END
```

Assignment Statement — allows evaluation of any expression. An expression is created from one or more of the elements that have a value combined to represent a new combined value. Another way of explaining an expression is to describe it as one side of an algebraic formula. Several things to note about expressions are:

- Expressions can be very simple (one element) or extremely complicated.
- Usually, all values are separated from each other by an operator.

Using meaningful names makes the expression much easier to understand.

Values can be of different types, and the proper combination of these types is necessary for correct results.

The Assignment Statement has the following syntax:

```
VAR = expression ;
```

Table 2-5 lists the available operators for expressions.

Table 2-5: Expression Operators

Operator	Description	Type
+	plus	FLOAT and INT, unary
-	minus	FLOAT and INT, unary
*	multiply	FLOAT and INT
/	divide	FLOAT and INT
>=	greater than or equal	INT, FLOAT, or STR
<=	less than or equal	INT, FLOAT, or STR
>	greater than	INT, FLOAT, or STR
<	less than	INT, FLOAT, or STR
==	equal	INT, FLOAT, or STR
<>	not equal	INT, FLOAT, or STR
=	assignment	INT, FLOAT, or STR
and	logical AND	INT
or	logical OR	INT
not	logical NOT	INT
band	bitwise AND	INT (assumed to be unsigned)
BOR	bitwise OR	INT (assumed to be unsigned)
BXOR	bitwise EXCLUSIVE OR	INT (assumed to be unsigned)
BNOT	bitwise NOT	INT (assumed to be unsigned)
&	STRING concatenation	STR

Following are some examples of Assignment statements:

`A = A+B;` where A and B are numeric values

`B = 34/VOLTS;` where B and VOLTS are numeric values

`COMMAND = "CMD" & COMMANDTYPE;` where COMMAND and COMMANDTYPE are string variables

`POLLRESPONSE = SERIALPOLL();` where POLLRESPONSE is an integer variable

LOGICALVALUE = (NOT(A AND B)) OR C; where LOGICALVALUE, A, B, and C, are integer variables

BITVALUE = (BNOT(A AND B)) OR C; where BITVALUE, A, B, and C are integer variables

Parentheses can be used to define the order of evaluation of an expression. The default order of precedence for evaluation is as follows (highest precedence to lowest precedence):

not, BNOT	logical negation and bitwise negation
*, /	multiplication and division
+, -	addition and subtraction
&	string concatenation
<=, >=, >, <	less than or equal, greater than or equal, less than, greater than
==, <>	equal and not equal
band	bitwise AND
BOR, BXOR	bitwise OR and bitwise EXCLUSIVE OR
and	logical AND
or	logical OR
=	assignment

IF Statement — allows decision processing. The syntax for an IF statement is as follows:

```
IF expression THEN dialogs OR statements;  
  ELSE dialogs OR statements;  
ENDIF
```

OR

```
IF expression THEN dialogs OR statements;  
ENDIF
```

The *expression* in an IF *expression* THEN clause must evaluate to a zero (FALSE) or a non-zero (TRUE) number. By definition, a comparison of STRINGS, less than, greater than, less than or equal, greater than or equal, equal, not equal, bitwise, and logical operators result in a zero or non-zero number. When the *expression* in an IF statement results in a TRUE condition, the *dialogs OR statements* in the initial IF statement are executed. When the *expression* results in a FALSE condition, the *dialogs OR statements* in the ELSE portion of the construct are executed, or, in the case of no ELSE, the *dialogs OR statements* of the IF construct are bypassed without execution. Therefore, an IF statement would be used to choose between alternative actions, based on the result of the *expression*.

You can nest `IF` statements.

WHILE Statement — allows repeated dialog evaluation. The syntax for a `WHILE` statement is as follows:

```
WHILE expression DO dialogs OR statements;  
END
```

The *expression* in a `WHILE expression DO` clause must evaluate to a zero (FALSE) or a non-zero (TRUE) number. By definition, a comparison of strings, less than, greater than, less than or equal, greater than or equal, equal, not equal, bitwise, and logical operators result in a zero or non-zero number. The *dialogs* OR *statements* in the `WHILE` statement will continue to execute until the *expression* results in a FALSE condition.

You can nest `WHILE` statements.

Functions

CHR

The number to ASCII character conversion function is used in a controller dialog to send binary character strings to an instrument. The syntax for the function is `CHR(n)`, where *n* is a number from 0 – 255. You can use this function to send binary or format characters to an instrument. For example, you may have an instrument that needs a binary pattern of 00000010 to set it to a particular state. You could use the following `SETTING` block to accomplish this:

```
CONTROLGROUP  
CONTROL CH1RESET:INT PUSHBUTTON @24,12  
  STRING "CH1 RESET";  
END  
SETTING  
CONT -> CHR(2),CHR(2);  
END  
END
```

TIMEDELAY

Another function that is useful in a `SETTING` or `MEASUREMENT` block is the **TIMEDELAY** function. It can be used anytime you need to wait for a process to complete, such as closing a relay. The syntax for the `TIMEDELAY` function is `TIMEDELAY(n)`, where *n* is in milliseconds. The following is an example of the `TIMEDELAY` function.

```
SETTING  
CONT -> "CLOSE:F1.A1";  
  TIMEDELAY(100);  
END
```

Display

A third useful function is the display of a message to the operator. The syntax for this function is **DISPLAY**(*stringdata*), where *stringdata* is a string constant, string variable, or string expression. This function lets you display a message to the operator on the screen in a dialog box during script execution. It can be used to say that a value is out of range, or that an instrument function requested did not execute. For example, if the voltage output range specified is out of bounds, you could send a message to the operator using the following example script:

```
CONTROLGROUP
CONTROL VOLTS:FLOAT EDITBOX @20,10
END
SETTING
  IF (VOLTS > 24 OR VOLTS < 0.1) THEN
    DISPLAY("THE VOLTAGE MUST BE SET BETWEEN 24 AND 0.1
VOLTS");
  ELSE
    CONT -> "VOLTS: ",VOLTS;
  ENDIF
END
MEASUREMENT
CONT -> "VOLTS?";
INST -> "VOLTS: ",VOLTS;
END
END
```

FASTDCWRITE

If you have instruments that support the VXI Fast Data Channel protocol, you can use the FASTDCWRITE function in a SETTING block to send the contents of a file to an instrument. The syntax for the FASTDCWRITE function is as follows:

```
FASTDCWRITE(FILENAME, FDCBASEADDR, FDCSIZE, CMDSTR)
```

Where: *FILENAME* is the name of the file to read from (a string); *FDCBASEADDR* is the base address of the instrument FDC memory (an integer); *FDCSIZE* is the size of the FDC memory (an integer); and *CMDSTR* is the Word Serial command to send to the instrument to start an FDC transfer (a string).

The following is a simple **Script** example using the fast data channel functions:

```
SCRIPT "FDCTEST"

VXIINTERNAL
END

VIEW
CONTROLGROUP
```

```

CONTROL FDCREAD:STR EDITBOX @5,2
CONTROLTITLE "FDCREAD:";
NUMCOLS 14;
END

```

```
SETTING
```

```

REM QUERY THE INSTRUMENT FOR THE FDC PROTOCOL
REM PARAMETERS.

```

```

TEMPVAR BASEADDR:INT,SIZE:INT;
CONT -> "HEADER OFF";
CONT -> "FDCBASE?";
INST -> "#H",BASEADDR:%H;
CONT -> "FDCSIZE?";
INST -> SIZE;
CONT -> "HEADER ON";

```

```

REM READ FROM THE INSTRUMENT AND PLACE THE DATA INTO
REM A FILE.

```

```

FASTDCREAD(FDCREAD,BASEADDR,SIZE,"FDCLOAD?");
END
END

```

```
CONTROLGROUP
```

```

CONTROL FDCWRITE:STR EDITBOX @5,6
STRING "DEF.ED0" ;
CONTROLTITLE "FDCWRITE:";
NUMCOLS 14;
END

```

```
SETTING
```

```

REM QUERY THE INSTRUMENT FOR THE FDC PROTOCOL
REM PARAMETERS.

```

```

TEMPVAR BASEADDR:INT,SIZE:INT;
CONT -> 'NEW';
CONT -> "FDCBASE?";
INST -> "#H",BASEADDR:%H;
CONT -> "FDCSIZE?";
INST -> SIZE;
CONT -> "HEADER ON";

```

```

REM WRITE THE DATA FROM THE FILE TO THE INSTRUMENT.
FASTDCWRITE(FDCWRITE,BASEADDR,SIZE,"FDCLOAD");

```

```

END
END
END

```

Notice that FASTDCWRITE uses a SETTING block to transfer data. You need the name of the file to read data from, and the only way to get data from the front panel or test program to the instrument is by using a SETTING block.

FASTDCREAD

If you have instruments that support the VXI Fast Data Channel protocol, you can use the **FASTDCREAD** function in a SETTING block to read data from an instrument and place it into a file. The syntax for the FASTDCREAD function is as follows:

```
FASTDCREAD ( FILENAME , FDCBASEADDR , FDCSIZE , CMDSTR )
```

Where `FILENAME` is the name of the file to read from (a string); `FDCBASEADDR` is the base address of the instrument FDC memory (an integer); `FDCSIZE` is the size of the FDC memory (an integer); and `CMDSTR` is the Word Serial command to send to the instrument to start an FDC transfer (a string).

The following is a **Script** example using the fast data channel functions:

```
SCRIPT "FDCTEST"

VXIINTERNAL
END

VIEW
CONTROLGROUP
CONTROL FDCREAD:STR EDITBOX @5,2
CONTROLTITLE "FDCREAD:";
NUMCOLS 14;
END

SETTING

REM QUERY THE INSTRUMENT FOR THE FDC PROTOCOL
REM PARAMETERS.

TEMPVAR BASEADDR:INT, SIZE:INT;
CONT -> "HEADER OFF";
CONT -> "FDCBASE?";
INST -> "#H", BASEADDR:%H;
CONT -> "FDCSIZE?";
INST -> SIZE;
CONT -> "HEADER ON";

REM READ FROM THE INSTRUMENT AND PLACE THE DATA INTO
REM A FILE.

FASTDCREAD(FDCREAD, BASEADDR, SIZE, "FDCLOAD?");
END
END

CONTROLGROUP
CONTROL FDCWRITE:STR EDITBOX @5,6
STRING "DEF.ED0" ;
CONTROLTITLE "FDCWRITE:";
```

```

        NUMCOLS 14;
    END

    SETTING

    REM  QUERY THE INSTRUMENT FOR THE FDC PROTOCOL
    REM  PARAMETERS.

        TEMPVAR BASEADDR:INT, SIZE:INT;
        CONT -> "NEW" ;
        CONT -> "FDCBASE?" ;
        INST -> "#H", BASEADDR:%H;
        CONT -> "FDCSIZE?" ;
        INST -> SIZE ;
        CONT -> "HEADER ON" ;

    REM  WRITE THE DATA FROM THE FILE TO THE INSTRUMENT.

        FASTDCWRITE (FDCWRITE, BASEADDR, SIZE, "FDCLOAD" ) ;
    END
    END
    END

```

Notice that FASTDCREAD is not in a QUERY block. You need the name of the file to write data to, and the only way to get data from the instrument to the front panel or test program by using a SETTING block.

READLENGTH

The **READLENGTH** function sets the number of data bytes to read with the next **INST** dialog. The default value for the number of data bytes read is -1, or all bytes sent by an instrument. An instrument that talks without terminating the data string causes problems with unwanted timeouts. An instrument that does not stop sending data also causes problems. The **READLENGTH** function applies only to the next **INST** dialog. The **INST** dialog resets the **READLENGTH** value to -1 (read all data) when it is complete. The **READLENGTH** function does not set the absolute number of data bytes to read. It only put an upper limit on the number of bytes read. Fewer data bytes may be read depending on the data being sent from the instrument. The following is an example of the **READLENGTH** function.

```

SETTING
    CONT -> "*IDN?";
    READLENGTH(12);
    INST -> VALUE1;
END

```

If an instrument has more data ready to send than will be read by the **INST** dialog as modified by the **READLENGTH** function, the status of the extra data strings will be determined by the instrument. In some cases the data is discarded and in some cases, the data will be sent to the next **INST** dialog.

Variable Types

There are four variable types available in scripts: integers, floats, STRING and waveform. The variable name is followed by the variable type (:INT for integer, :FLOAT for floats, :STR for String and :WAVE for waveform). Variable names must start with an alphabetic character and be no more than 15 characters in length.

Integers

Integers require 4 bytes for storage. The range is $-2^{31} - 2^{31}-1$

Floats

Floats require 8 bytes for storage and use the IEEE format. The approximate range (as specified for Microsoft C) is $2.2E-308 - 1.8E+308$.

Strings

A string is any number of bytes, each byte being a character. The number of bytes in a string is limited to the amount of memory available.

Literal strings (enclosed in “ ”) are limited to 255 characters. If a literal string exceeds this limit, an error is reported and parse fails. Strings longer than 255 characters can be built up in script using the ampersand (&) concatenation operator. Several special printing and non-printing characters can be included in a literal string by using the backslash character (\) followed by the desired character. These special characters and their string entries are as follows:

\	\\
“	\\”
LF (Line Feed)	\\R
HT (Horiz Tab)	\\T
CRLF (Return Line Feed)	\\n

Carriage returns are ignored in literal strings except when explicit \\n appears.

Waveforms

Waveform variables are a special type of data and can be used only in very specific situations. You can either assign one waveform variable to another waveform variable or assign it to the function WAVEFORMTOVAR. Any other use of the waveform variable causes an error.

Variable Formats

In controller and instrument dialogs, you can precisely specify the format for variables.

Controller Dialog Formats

In a controller dialog, the format can be specified for a string variable.

To specify the format, each variable is followed by a colon, followed by the format. The format specification begins with a % and ends with a format character. Between the % and the format character one may specify the modifiers, width, and precision.

The following are examples of specifying formats:

```
CONT -> "VOLTS ", VOLTS:%9.5E;  
CONT -> "AMPS ", AMPS:%7.3G;
```

The following list shows the format types that are available, what they are used for, and the default field widths:

%nD	integer, the numbers 0 – 9, + and –. n specifies the number of digits. Default width of 10 if n is not specified.
%n.mG	floating point, integers, and numbers with decimal points. n defines the total field size, including the decimal point, and m defines the maximum number of characters following the decimal point. Numbers of greater precision than specified will be rounded. Default width of 19 if n is not specified.
%n.mE	integers, floating point, and exponential numbers. Scientific notation characters are 0 – 9, + and – , E or e. n defines the total field size, including the decimal point, m defines the maximum number of characters following the decimal point. Numbers of greater precision than specified will be rounded. Default width of 24 if n is not specified.
%nB	binary, the numbers 0 and 1. The number is assumed to be unsigned and is interpreted as right justified. n specifies the number of digits. Default width of 32 if n is not specified.
%nO	octal, the numbers 0 – 7. n specifies the number of digits. Default width of 11 if n is not specified.
%nH	hexadecimal, the numbers 0 – 9 and the letters A – F. Input is assumed to be unsigned. n specifies the number of digits. Default width of 8 if n is not specified.
%nS	string. n specifies the number of characters. All characters are accepted if n is not specified. No default width.
%F	file name. No default width.

If the specified format is incompatible with the type of the variable, an error is reported.

Binary, Octal, and Hexadecimal formats are used to write ASCII string representations of these types of numbers; they do not write binary representations of these formats.

If *n* is not specified, the processing is Free Format. Output is based on the minimum number of bytes required for that format.

If the format is not specified, the type of the variable is taken as the default format and the format is Free Format.

You can use appropriate width or width precision modifiers with these format types. The following are examples of specifying field widths and using precision modifiers with format types:

```
CONT -> VAR1:%6D;
CONT -> VAR1:%12.6E;
```

If the format width is specified but is not large enough to accommodate the number, an error is reported.

If the width is specified and the number of digits for the number is less than the width, the remaining characters are discarded.

The following is an example of specifying field widths, using a precision modifier, and using a formatting modifier:

```
CONT -> "VOLTS ",VOLTS:%+5.2G;
```

In this example, + is the formatting modifier, 5 is the specified width, 2 is the precision modifier, and G specifies the format type.

The available formatting modifiers are:

- Z** Leading zero fill instead of the default space fill.
- +** Always output sign.
- <** Left justify instead of the default right justify.

Following are examples in the use of formatting modifiers (assume the variable VOLTS contains the value 1.1):

```
CONT -> "VOLTS ",VOLTS:%<5.2G; output STRING "VOLTS 1.10 "
                                     (contains a trailing space)
CONT -> "VOLTS ",VOLTS:%+5.2G; output STRING "VOLTS +1.10"
CONT -> "VOLTS ",VOLTS:%Z5.2G; output STRING "VOLTS 01.10"
CONT -> "VOLTS ",VOLTS:%+Z5.2G; output STRING "VOLTS
+01.10"
```

Notice that you can combine formatting modifiers.

The file name format is used with a string variable and indicates that the entire transaction is between a file and an instrument. Using %F in a controller dialog will transfer the contents of a file to an instrument. If you use the %F format, it must be the only format specified in the dialog, or an error will be reported.

Following is an example of using the %F format:

```
SETTING
  DATAFILE = "TEST1.DAT";
  CONT -> DATAFILE:%F;
END
```

Table 2-6 lists the available formats for controller dialogs:

Table 2-6: Controller Dialog Format

<i>CONT</i>	B	<i>D</i>	E	<i>F</i>	G	H	<i>O</i>	S
STR ing				X				X
INT eger	X	X	X		X	X	X	
<i>FLOAT</i>		X	X		X			
Z	X	X	X		X	X	X	
+		X	X		X			
<	X	X	X		X	X	X	X
n	X	X	X		X	X	X	X
m			X		X			
<i>Default Width</i>	32	10	24		19	8	11	

NOTE

IPG converts all controller dialogs, except those using the %F format, to strings before sending them to the instrument. Therefore, it is very important to properly specify modifiers if and when they are used. For example, if the value 10.1 was to be sent to the instrument, and the modifier 2.1G was specified, only 10 would be sent. This occurs because IPG truncates data based on the modifier format specified. When truncation occurs, a warning message is generated by IPG. In this case, the proper modifier should have been 4.1G. This problem can be prevented by using the %G modifier format rather than the N.MG modifier format.

Instrument Dialog Formats

In an instrument dialog, the format can be specified for a variable.

To specify the format, each variable is followed by a colon, followed by the format. The format specification begins with a % and ends with a *format character*. Between the % and the format character one may specify the *modifiers, width, and precision*.

The following are examples of specifying formats:

```
INST -> STATUS:%6B;  
INST -> COUNT:%4D;
```

The following list shows the format types that are available, what they are used for, and the default field widths:

%nD	integer, the numbers 0 – 9, + and –. <i>n</i> specifies the number of digits. Default width of 10 if <i>n</i> is not specified.
%n.mG	floating point, integers, and numbers with decimal points. <i>n</i> defines the total field size, including the decimal point, and <i>m</i> defines the maximum number of characters following the decimal point. Default width of 19 if <i>n</i> is not specified
%n.mE	integers, floating point, and exponential numbers. Scientific notation characters are 0 – 9, + and –, E or e. <i>n</i> defines the total field size, including the decimal point, <i>m</i> defines the maximum number of characters following the decimal point. Default width of 24 if <i>n</i> is not specified.
%nB	binary, the numbers 0 and 1. The number is assumed to be unsigned and is interpreted as right justified. <i>n</i> specifies the number of digits. Default width of 32 if <i>n</i> is not specified.
%nO	octal, the numbers 0 – 7. <i>n</i> specifies the number of digits. Default width of 11 if <i>n</i> is not specified.
%nH	hexadecimal, the numbers 0 – 9 and the letters A – F. Input is assumed to be unsigned. <i>n</i> specifies the number of digits. Default width of 8 if <i>n</i> is not specified.
%nS	string. <i>n</i> specifies the number of characters. All characters are accepted if <i>n</i> is not specified — no default width.
%F	file name (no default width)

If the specified format is incompatible with the type of the variable, an error is reported.

Binary, Octal, and Hexadecimal formats are used to read ASCII string representations of these types of numbers; they do not read binary representations of these formats.

If *n* is not specified, the processing is Free Format. Input is accepted until either a maximum number of bytes have been received, or a non-valid character is received for that field.

If the format is not specified, the type of the variable is taken as the default format and the format is Free Format.

You can use appropriate width or width and precision modifiers with these format types. The following are examples of specifying field widths and using precision modifiers with format types:

```
CONT -> VAR1:%6D;
CONT -> VAR1:%12.6E;
```

If the format width is specified but is not large enough to accommodate the number, an error is reported.

If the width is specified and the number of digits for the number is less than the width, the remaining characters are discarded.

The file name format is used with a string variable and indicates that the entire transaction is between a file and an instrument. Using %F in an Instrument dialog will place all of the data read from the instrument into a data file. If you use the %F format, it must be the only format specified in the dialog, or an error will be reported.

Following is an example of using the %F format:

```
QUERY
  DATAFILE = "RESPONSE.DAT";
  INST -> DATAFILE:%F;
END
```

Table 2-7 lists the available formats for Instrument dialogs:

Table 2-7: Instrument Dialog Format

INST	B	D	E	F	G	H	O	S
STRing				X				X
INTeger	X	X	X		X	X	X	
FLOAT		X	X		X			
n	X	X	X		X	X	X	X
m			X		X			
Default Width	32	10	24		19	8	11	

Common Problems when Using Instrument Dialog Formatting

Not Enough Data Received — Parsing of an instrument dialog takes place from left to right. If the string received from the instrument does not contain enough bytes to completely parse the Instrument dialog, an ERROR is reported. Following is an example:

```
MEASUREMENT
  TMPVAR VAR1:STR;
  INST -> VAR1:%5S;
END
```

Assuming that the string returned from the instrument is `ON`, an error is reported because the string returned from the instrument contains only two characters and the Instrument dialog requires that five bytes be available.

Too Much Data Received — If the string returned from the instrument contains more bytes than required to parse an instrument dialog, then the excess bytes are ignored (the rest of the buffer is flushed). Following is an example:

```
MEASUREMENT
  TMPVAR VAR1:INT;
  INST -> VAR1;
END
```

Assuming that the string returned from the instrument is `12 VOLTS`, `VAR1` is assigned the value `12`, and the rest of the string, `VOLTS`, is ignored.

Non-Numeric Character in a Numeric Format Handling — If the width (for example, `n`) is specified in the format, then a non-valid character, if received before `n` characters have been received, delimits the input. A non-valid character is a character that is not compatible with the format; e.g., non-digit for `%D` format, or non-binary for `%B` format. If the first character is a non-valid character, numeric conversion is stopped at the invalid character and no characters are processed.

ISL Keywords

Table 2-8 provides an alphabetical list of the reserved keywords in the Instrument Script Language. You can not use these keywords for any other purpose, such as naming a variable; they are reserved for ISL use.

Table 2-8: ISL Keywords Listed Alphabetically

&	DEFAULTPOS	NORMALCLOSE	TEXT
*	DISPLAY	NORMALOPEN	TEXTBOX
+	DO	NOT	THEN
	EDIT	NUMCOLS	TIM
/	EDITBOX	NUMPOS	TIMEDELAY
:%	ELSE	NUMROWS	TIMEOUT
<	END	OFF	TITLE
<=	ENDIF	ON	TMO
=	EOI	ONEOFGROUP	TMPVAR
==	EOM	OPEN	TO
>	FASTDCREAD	OR	TOLEFTOF
=>	FASTDCWRITE	PARITY	TORIGHTOF
AND	FLOWCONTROL	PUSHBUTTON	TOSCRIP
ASCII	FALSE	QUERY	TOSCRIPTOFF
ATN	FILE	RADIOBUTTON	TOSCRIPTON
BAND	FLOAT	READLENGTH	TRUE
BAUD	GET	REM	UNL
BINARY	GPIB	REMARK	UNT
BNOT	GTL	REN	UPDATE
BOR	HELP	RS232	UPDATELIST
BXOR	HSCROLL	SCANSTRING	VIEW
CDSBUS	IF	SCRIPT	VSCROLL
CHECKBOX	IFC	SCROLLHORZ	VXIEVENT
CHR	INST	SCROLLVERT	VX5520
CLOSE	INSTRUMENT	SDC	VXIINT
CONNECT	INT	SERIALPOLL	VXIINTERNAL
CONT	LEARN	SET	WAVE
CONTROLLER	LISTBOX	SETTING	WAVEFORM-
CONTROL	LLO	SKIP	DISPLAY
CONTROL-	MEASURE	STATE	WAVEFORM-
GROUP	MEASURE-	STOPBITS	TOADIF
CONTROLTITLE	MENT	STR	WAVEFORMTO-
DATABITS	MIDSTRING	STRING	VAR
DCL	MXI	TEMPVAR	WHILE
DEFAULT	NO		YES

Table 2-9 lists the ISL keywords by their use.

Table 2-9: ISL Keywords Listed by Group

BUSNOTE	BAUD	TROL	TIMEOUT
	CDSBUS	GPIB	TMO
	DATABITS	MXI	VX5520
	EOI	PARITY	VXIEVENT
	EOM	RS232	VXIINT
	FLOWCON-	STOPBITS	VXIINTERNAL
CONTROL Block	CHECKBOX	NORMALCLOSE	STRING
	CLOSE	NORMALOPEN	TEXTBOX
	CONTROL	NUMCOLS	TITLE
	CONTROLTITLE	NUMPOS	TOLEFTOF
	DEFAULT	NUMROWS	TORIGHTOF
	DEFAULTPOS	OFF	TOSCRIP
	EDIT	ON	TOSCRIPTOFF
	EDITBOX	ONEOFGROUP	TOSCRIPTON
	FALSE	OPEN	TRUE
	FILE	PUSHBUTTON	UPDATE
	FLOAT	RADIOBUTTON	UPDATELIST
	HSCROLL	SCANSTRING	VSCROLL
	INT	SCROLLHORZ	WAVE
	LISTBOX	SCROLLVERT	WAVEFORM-
MIDSTRING	STATE	DISPLAY	
NO	STR	YES	
CONTROL- GROUP Block	CONTROL- GROUP		
Format	:%		
Functions	CHR	SERIALPOLL	WAVEFORM-
	FASTDCREAD	SKIP	TOADIF
	FASTDCWRITE	TIMEDELAY	WAVEFORMTO-
	READLENGTH		VAR
GPIB Commands	ATN	IFC	TIM
	DCL	LLO	UNL
	GET	REN	UNT
	GTL	SDC	
LEARN Block	ASCII	BINARY	LEARN
Miscellaneous	HELP	REM	REMARK

Table 2-9: ISL Keywords Listed by Group (Cont.)

Operators	&	<=	BAND
	*	=	BNOT
	+	==	BOR
	-	>	BXOR
	/	=>	NOT
	<	AND	OR
Script Block	SCRIPT		
SETTINGS and MEASUREMENT Blocks	CONT	IF	SET
	CONTROLLER	INST	SETTING
	DISPLAY	INSTRUMENT	TEMPVAR
	DO	MEASURE	THEN
	ELSE	MEASURE-	TMPVAR
	END	MENT	TO
	ENDIF	QUERY	WHILE
VIEW Block	CONNECT	TEXT	VIEW

Section 3

Writing Scripts

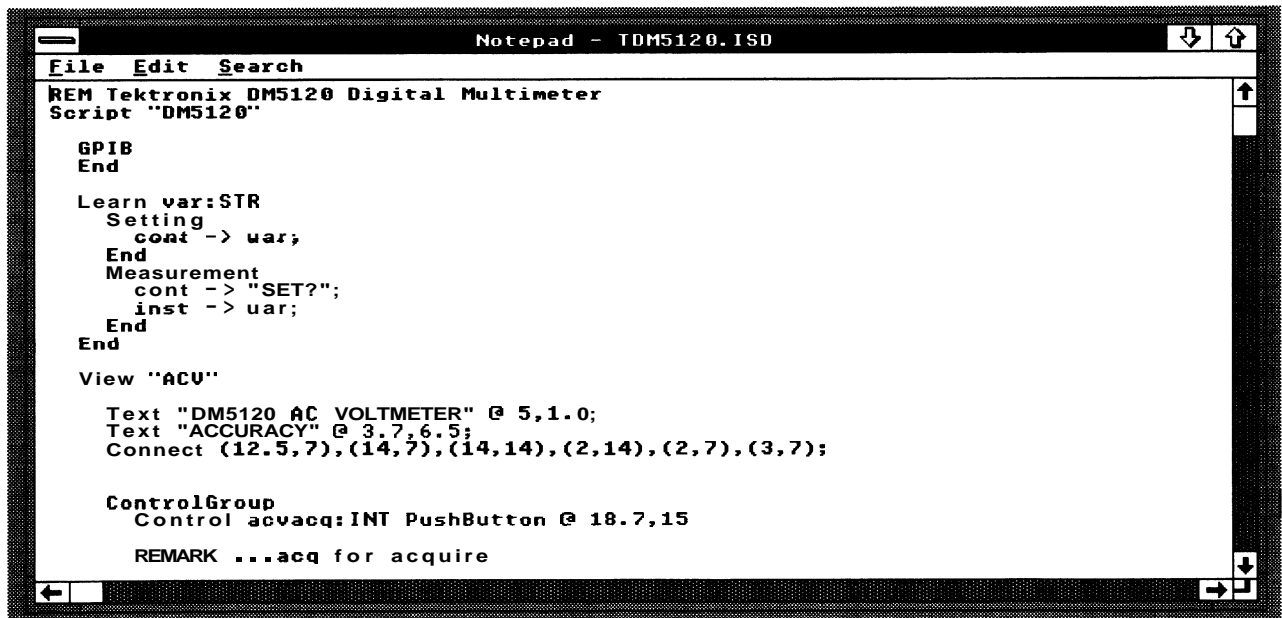
Writing Scripts

Introduction

The Editor

With TekTMS Instrument Script Language (ISL), developing scripts requires only a word processing editor that produces ASCII character code. For example, the Windows Notepad word processor has ASCII output and can be used easily with this section to write an example script.

Figure 3-1 shows Windows Notepad with the TDM5120.ISD file opened. You can save an *.ISD file with Notepad, then call it up in IPG and test it. See Testing Script Files, in this section, for further information.



```
Notepad - TDM5120.ISD
File Edit Search
REM Tektronix DM5120 Digital Multimeter
Script "DM5120"

GPIB
End

Learn var:STR
Setting
  cont -> uar;
End
Measurement
  cont -> "SET?";
  inst -> uar;
End
End

View "ACU"

Text "DM5120 AC VOLTMETER" @ 5,1.0;
Text "ACCURACY" @ 3,7,6.5;
Connect (12.5,7),(14,7),(14,14),(2,14),(2,7),(3,7);

ControlGroup
  Control acvacq:INT PushButton @ 18.7,15

REMARK ...acq for acquire
```

Figure 3-1: Windows Notepad with TDM5120.ISD Text

References

Refer to the other sections of this manual for formatting details while developing scripts. Section 2, ISL Descriptions, gives coding details on all blocks and related statements of the script structure.

Refer to a script model that is with your IPG software. Appendix C, *TDM5120.ISD Script Printout*, contains a printout of that entire TDM5120.ISD script model.

Refer to the respective bus interfacing manual relative to each different instrument for which you write or modify scripts.

The Script Model

The script file TDM5120.ISD is provided with IPG on one of the software diskettes. It can be used as a reference for the keywords and their arrangements needed to produce a script for front panel and computer-to-instrument interfacing.

NOTE

When you name your scripts, remember that they cannot have the same names as the **Device Names** defined in your GPIB.COM file. If you have configured DM5120, for example, in your *IBCONF*, you cannot use *DM5120.ISD* as a script file name. Notice that the DM5120 script model name is *TDM5120.ISD*.

All instrument scripts will use the same basic grammar for script development. Each instrument, however, will have its own special command syntax for use within the scripts for bus communication.

Coding for any particular bus parameters is explained with BUSNOTE Block description in Section 2. In the TDM5120 script, the coding is simply GPIB, with END.

This section will illustrate writing a script using the block grammar and instrument commands in Windows Notepad.

Detailed examples are included in this section to illustrate the concepts of each script block and associated statements.

Script Planning

Charting A Front Panel

Display — IPG Software uses character size to facilitate locating coordinate points on the display. Coordinate portions begin at the top left of the window user space at 0,0 horizontally and vertically.

Regardless of your display type (EGA, VGA, etc.) you define points on your display based on character size.

You can plan on the location of front panel control types by laying out coordinates of your display screen on a sheet of paper and then scaling controls in the coordinates, using the following guidelines.

Control Sizes (numbers are character spaces):

Textbox and Editbox	Height = NumRows + 2 Width = NumCols + 1 (add 2 in each case for scroll bar)
Listbox	Height = NumRows + 1 (add 2 in for scroll bar) Width = NumCols + 1
Pushbutton, Checkbox or Radiobutton	Height = 2 Width = NumCols + 3

Instrument — The screen display controls relate to the instrument functionality. The illustration on the following page shows how information is passed from the front panel control through the script to the instrument, and back.

Determining Control Types

The types of controls you will use in your scripts will be based on the instrument functionality that you want to present. The types of controls may be effected by the test system and the tests themselves. This manual section will give you an idea of what can be done with a digital multimeter, but cannot begin to cover all possibilities.

With the TDM5120.ISD model script, not all of the instrument's front panel controls available are represented. The ACV function and all ranges will be explained along with reading resolution selections and a filter switch with filter values.

We have chosen the use of an IPG View to accommodate the DM5120 instrument's ACV function, as shown in Figure 3-3. A DCV view is coded, but not explained (you can easily add an OHMS and other functions with other views in the script).

A Listbox is used for RANGE control, and a group of Radiobuttons is used to control the resolution of the Measurement reading.

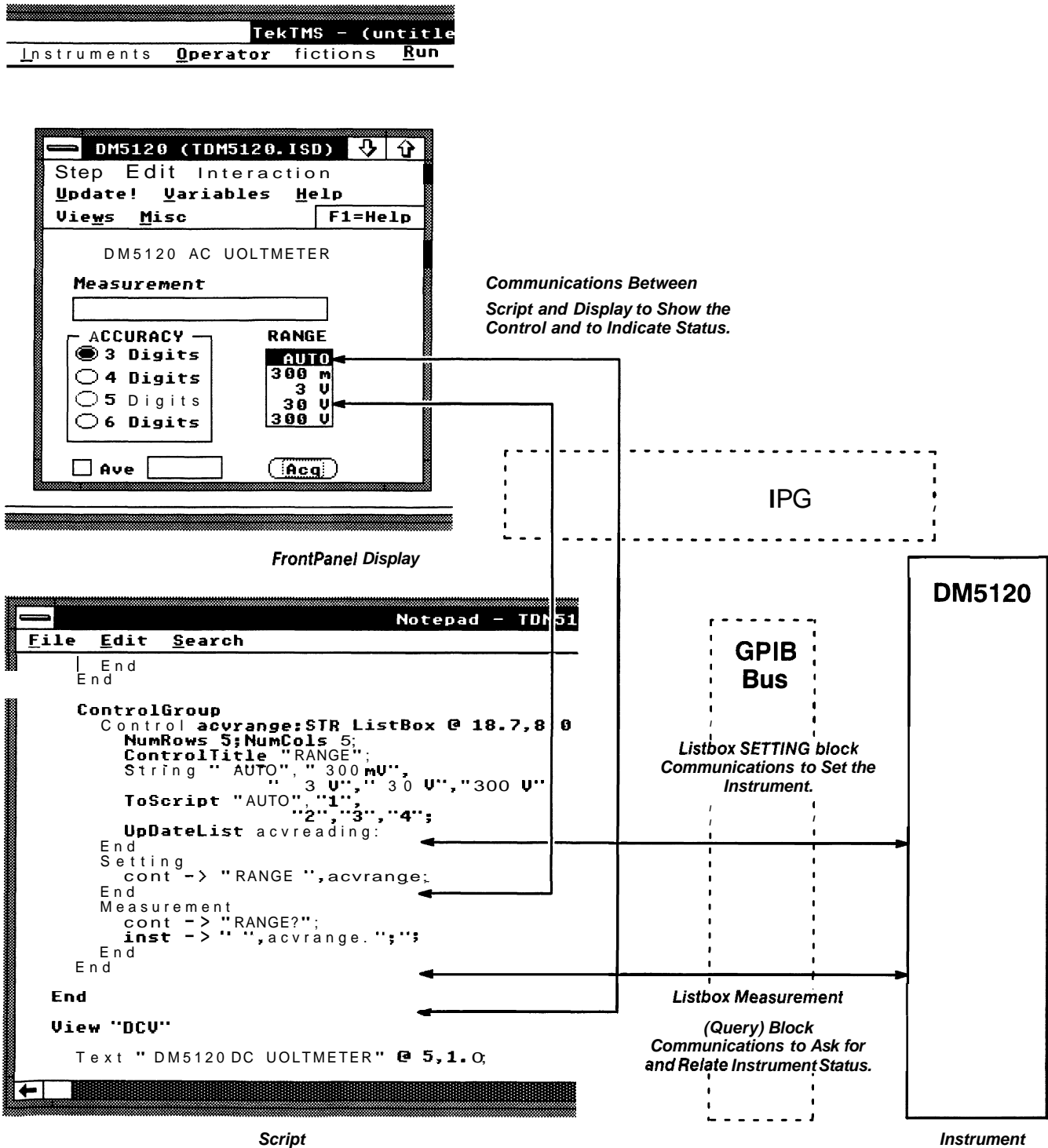


Figure 3-2: Instrument and Display Script Relationships

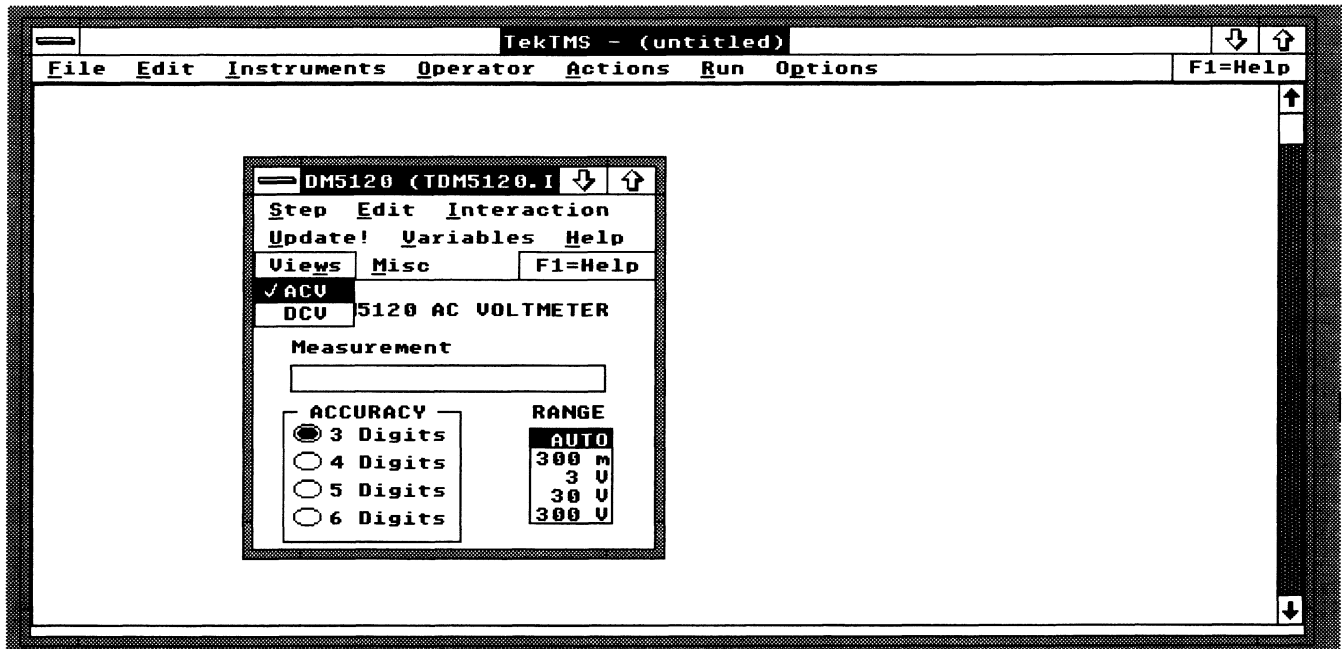


Figure 3-3: IPG View of DM5120 ACV Function

A Checkbox is used to turn the FILTER modifier switch (Ave) on and off; and an EDITBOX was chosen to allow the operator to input filter values. A Text box allows measurement readings from the instrument to the display.

We chose a Pushbutton (Acq) to acquire the Measurement reading upon operator request because it is a momentary action.

Then, we chose a set of Radiobuttons for resolution (3.5, 4.5, 5.5 and 6.5 digit Measurement display) since they could be grouped. A List box easily handles the RANGE with NUMROW and NUMCOL. A Checkbox is used to turn on a FILTER switch; and an EDITBOX is used in conjunction with the filter CHECKBOX to provide a way of entering filter values.

Interchangeability

Scripts should be designed such that one instrument can be easily replaced with another of the same type (e.g., a power supply by one manufacturer replaced by that of a different manufacturer). In order to make instruments of the same type interchangeable, an instrument control that is common between the two instruments must have the same Control ID (variable name) in each script, and should be of the same control type (e.g., EDITBOX, CHECKBOX, etc.) and the same data type (floating point, integer, string, etc.). For user clarity, it also is desirable that the control's label be the same on both instrument front panels.

Script Development Procedures

To get started, invoke your ASCII editor of choice (Notepad, Word, etc.). As a reminder, when you finish your script, save it with a *.ISD extension (example: TDM5120.ISD). Also, matching the script name to the instrument name will simplify locating that instrument script in your computer when needed, but remember, you cannot name an *.ISD file with a name that is the same as a **Device Name** in your GPIB IBCONF file.

Script and Description Presentation

Using this manual section you can write a script for the Tektronix DM5120 Programmable Digital Multimeter using the detailed description of the example script. The example script is shown in portions to highlight coding; and descriptions of what that coding does are presented beside the script.

Appendix C, *TDM5120.ISD* Script Printout, has a printout of the complete TDM5120.ISD script.

Notice also that the coding line(s) from that part of the script being described is in bold type in the partial script frame.

The first line of the script we have developed is a remark (REMark) to identify the associated instrument.

REMARK — REM or **REMARK** must be the first keyword on each remark line. Remarks can be placed on a line(s) anywhere in the script.

```
REM TEKTRONIX DM5120 DIGITAL MULTIMETER
SCRIPT "DM5120"
```

```
    GPIB
    END
```

```
    VIEW "ACV"
```

```
        TEXT "DM5120 AC VOLTMETER" @ 5,1.0;
        TEXT "ACCURACY" @ 3.7,6.5;
        CONNECT (12.5,7), (14,7), (14,14), (2,14), (2,7), (3,7);
```


Beginning a Script — The actual script is opened by typing `SCRIPT` and then giving it a name in quotation marks.

Because of constraints caused by the Help message system, the script name and control names must be unique in the first eight character spaces (refer to Section 4, Help Editor, for further information.).

```
REM TEKTRONIX DM5120 DIGITAL MULTIMETER
SCRIPT "DM5120"

GPIB
END

VIEW "ACV"

TEXT "DM5120 AC VOLTMETER" @ 5,1.0;
TEXT "ACCURACY" @ 3.7,6.5;
CONNECT (12.5,7),(14,7),(14,14),(2,14),(2,7),(3,7);
```

Choosing a Bus — The bus types currently supported are GPIB, **RS232** or **VXI** bus. For this script, GPIB is the bus type, with default parameters.

Refer to *BusNote* Block and GPIB Parameters in Section 2, ISL Descriptions for EOM, EOI and **TIMEOUT** defaults and other information.

```
REM TEKTRONIX DM5120 DIGITAL MULTIMETER
SCRIPT "DM5120"

GPIB
END

VIEW "ACV"

TEXT "DM5120 AC VOLTMETER" @ 5,1.0;
TEXT "ACCURACY" @ 3.7,6.5;
CONNECT (12.5,7),(14,7),(14,14),(2,14),(2,7),(3,7);
```

Defining the View — To begin the ACV voltage view of this instrument front panel, use the keyword **VIEW** and name the view "ACV".

```
REM TEKTRONIX DM5120 DIGITAL MULTIMETER
SCRIPT "DM5120"

GPIB
END

VIEW "ACV"

TEXT "DM5120 AC VOLTMETER" @ 5,1.0;
TEXT "ACCURACY" @ 3.7,6.5;
CONNECT (12.5,7),(14,7),(14,14),(2,14),(2,7),(3,7);
```

Text and Connect Statements — The first text statement specifies the title for the ACV view — `DM5120 AC VOLTMETER`, while the second text statement specifies the group title for the Radiobuttons — `ACCURACY`.

CONNECT places a box around the resolution Radiobuttons.

```
REM TEKTRONIX DM5120 DIGITAL MULTIMETER
SCRIPT "DM5120"

GPIB
END

VIEW "ACV"

TEXT "DM5120 AC VOLTMETER" @ 5,1.0;
TEXT "ACCURACY" @ 3.7,6.5;
CONNECT (12.5,7),(14,7),(14,14),(2,14),(2,7),(3,7);
```

Defining the First Control — The first control specified in this script will be a Pushbutton. This control will be used to set the instrument to ACV function and to initiate a measurement.

On the display screen, the Pushbutton will appear as a rectangle with **ACQ** shown inside it due to coding in its **CONTROL** block. A statement, `UPDATELIST`, will update instrument output to the **MEASUREMENT** display with the variable `ACVREADING`.

Defining the CONTROLGROUP Block — To define a **CONTROLGROUP** block, simply type the keyword **CONTROLGROUP**.

```
TEXT "ACCURACY" @ 3.7,6.5;
CONNECT (12.5,7),(14,7),(14,14),(2,14),(2,7),(3,7);

CONTROLGROUP
CONTROL ACVACQ:INT
    PUSHBUTTON @ 18.7,15
REMARK ...ACQ FOR ACQUIRE
    STRING "ACQ";
    UPDATELIST ACVREADING;
END
SETTING
    CONT -> "ACV";
```

Defining a CONTROL block — To specify the control characteristics for this Pushbutton, we start a CONTROL block with the keyword **CONTROL**.

Then the control type is specified (in this case PUSHBUTTON) .

Finally, the position of the Pushbutton on the display screen is positioned at 18.75 characters across and 15 characters down.

As a reminder, the control coordinates specify the upper left hand corner of the control.

```
TEXT "ACCURACY" @ 3.7,6.5;
CONNECT (12.5,7),(14,7),(14,14),(2,14),(2,7),(3,7);

CONTROLGROUP
  CONTROL ACVACQ:INT
    PUSHBUTTON @ 18.7,15
REMARK ...ACQ FOR ACQUIRE
  STRING "ACQ";
  UPDATELIST ACVREADING;
END
SETTING
  CONT -> "ACV";
```

Control Title — To show **ACQ** in the Pushbutton on the front panel display, we entered the line `STRING "ACQ"`.

```
TEXT "ACCURACY" @ 3.7,6.5;
CONNECT (12.5,7),(14,7),(14,14),(2,14),(2,7),(3,7);

CONTROLGROUP
  CONTROL ACVACQ:INT
    PUSHBUTTON @ 150,150
REMARK ...ACQ FOR ACQUIRE
  STRING "ACQ";
  UPDATELIST ACVREADING;
END
SETTING
  CONT -> "ACV";
```

Using UpDateList — With the UPDATELIST statement, when the ACQ Pushbutton is selected, the control ACVREADING will be updated.

```

TEXT "ACCURACY" @ 3.7,6.5;
CONNECT (12.5,7),(14,7),(14,14),(2,14),(2,7),(3,7);

CONTROLGROUP
CONTROL ACVACQ:INT
    PUSHBUTTON @ 150,150
REMARK ...ACQ FOR ACQUIRE
    STRING "ACQ";
    UPDATELIST ACVREADING;
END
SETTING
    CONT -> "ACV";

```

Ending a CONTROL block — To end the CONTROL block, use END.

```

CONTROL ACVACQ:INT
    PUSHBUTTON @ 150,150
REMARK ...ACQ FOR ACQUIRE
    STRING "ACQ";
    UPDATELIST ACVREADING;
END
SETTING
    CONT -> "ACV";
END
END

```

Defining the SETTING block — The SETTING block is defined by the keyword SETTING (or SET).

```

CONTROLGROUP
CONTROL ACVACQ:INT
    PUSHBUTTON @ 18.7,15
REMARK ...ACQ FOR ACQUIRE
    STRING "ACQ";
    UPDATELIST ACVREADING;
END
SETTING
    CONT -> "ACV";
END

```

Controller to Instrument Command — This line causes the controller to send "ACV" to the instrument.

ACV is a command defined by the DM5120 firmware to turn on the function ACV (AC voltage).

```

CONTROLGROUP
  CONTROL ACVACQ:INT
    PUSHBUTTON @ 18.7,15
  REMARK ...ACQ FOR ACQUIRE
    STRING "ACQ";
    UPDATELIST ACVREADING;
  END
  SETTING
    CONT -> "ACV";
  END

```

Ending the SETTING block — To end this SETTING block, use `END`.

Then, to end the CONTROL block, again use `END`.

```

CONTROLGROUP
  CONTROL ACVACQ:INT
    PUSHBUTTON @ 18.7,15
  REMARK ...ACQ FOR ACQUIRE
    STRING "ACQ";
    UPDATELIST ACVREADING;
  END
  SETTING
    CONT -> "ACV";
  END

```

Defining the Second Control — The next control used in the TDM5120.ISD script is the Textbox. It is used to display the instrument measurement.

As a reminder, the Textbox is a control with only a MEASUREMENT block.

Placing, Sizing and Labeling a Text Box — As with the last control, the CONTROLGROUP block is defined, then the CONTROL block is defined.

The control data type and the control type are specified, followed by the control display location.

The next line of control parameters specify the control size, 1 character space deep and 20 character spaces wide.

Finally, the control title "MEASUREMENT" is specified and the control is ended with END.

Because this is a Textbox, the CONTROLGROUP block has no SETTING block.

```

        CONT -> "ACV";
    END
END

CONTROLGROUP
    CONTROL ACVREADING:FLOAT TEXTBOX @ 2.5,4.5
        NUMROWS 1; NUMCOLS 20;
        CONTROLTITLE "MEASUREMENT";
    END
    MEASUREMENT
        CONT -> "SEND";
        INST -> READING, ";";
    END
END

```

Defining a MEASUREMENT block — The MEASUREMENT block is defined with the keyword **MEASUREMENT**. Other keywords that can be used to start a MEASUREMENT block are **MEASURE** and **QUERY**

```

        CONT -> "ACV";
    END
END

CONTROLGROUP
    CONTROL ACVREADING:FLOAT TEXTBOX @ 2.5,4.5
        NUMROWS 1; NUMCOLS 20;
        CONTROLTITLE "MEASUREMENT";
    END
    MEASUREMENT
        CONT -> "SEND";
        INST -> ACVREADING, ";";
    END
END

```

Talking to the Instrument — This dialog sends the instruction "SEND" to the instrument.

```

        CONT -> "ACV" ;
    END
END

CONTROLGROUP
    CONTROL ACVREADING:FLOAT TEXTBOX @ 2.5,4.5
        NUMROWS 1; NUMCOLS 20;
        CONTROLTITLE "MEASUREMENT" ;
    END
    MEASUREMENT
        CONT -> "SEND";
        INST -> ACVREADING, " ; " ;
    END
END

```

Instrument Measurement Update Path — This dialog receives and parses the instrument response to the previous command and places the measurement from the instrument into the control variable `acvreading`.

The floating variable will return the instrument reading. It will be in the format established by the variable and the instrument firmware (refer to the instrument manual for specific command data). In this case the measurement is in the variable `acvreading` and will end when a semicolon is read.

```

        CONT -> "ACV" ;
    END
END

CONTROLGROUP
    CONTROL ACVREADING:FLOAT TEXTBOX @ 2.5,4.5
        NUMROWS 1; NUMCOLS 20;
        CONTROLTITLE "MEASUREMENT" ;
    END
    MEASUREMENT
        CONT -> "SEND" ;
        INST -> ACVREADING, " ; " ;
    END
END

```

Ending the MEASUREMENT block — The MEASUREMENT block is ended with `END`, the same as the SETTING block.

End the CONTROLGROUP block with `END`.

```

        CONT -> "ACV" ;
    END
END

CONTROLGROUP
    CONTROL ACVREADING:FLOAT TEXTBOX @ 2.5,4.5
        NUMROWS 1; NUMCOLS 20;
        CONTROLTITLE "MEASUREMENT" ;
    END
    MEASUREMENT
        CONT -> "SEND" ;
        INST -> ACVREADING, ";";
    END
END

```

Setting the Resolution — The DM5120 has a capacity for 3 (3.5), 4 (4.5), 5 (5.5), or 6 (6.5) digits of resolution. Setting the desired resolution is done with a set of Radiobutton controls.

The coding for only one Radiobutton will be shown in the following example. See *Appendix A, TDM5120.ISD Script Printout*, for the differences between the script coding illustrated and that of the other three Radiobuttons.

Installing a Radiobutton — This CONTROLGROUP block and CONTROL block are defined in the same manner as the previous two controls.

Name the Radiobutton with the `STRING` Statement.

```

CONTROLGROUP
    CONTROL ACVTHREE:INT RADIOBUTTON @ 2.5,7.5
        STRING "3 DIGITS";
        ONEOFGROUP "ACCURACY" ;
        UPDATELIST ACVREADING;
    END
    SET
        IF (ACVTHREE==1) THEN
            CONT -> "DIGIT 3" ;
        ENDIF
    END
    QUERY
        TEMPVAR TMP:STR;
        ACVTHREE = 0;
        CONT -> "DIGIT?";
        INST -> "DIGIT ", TMP, ";";
        IF (TMP=="3") THEN
            ACVTHREE = 1;
        ENDIF
    END
END

```


Group the Radiobuttons— All four resolution Radiobuttons are made members of the same group (ACCURACY) with the ONEOFGROUP statement.

When one Radiobutton of the group is turned ON ,the others will be turned OFF.

```

CONTROLGROUP
  CONTROL ACVTHREE:INT RADIOBUTTON @ 2.5,7.5
  STRING "3 DIGITS";
  ONEOFGROUP "ACCURACY";
  UPDATELIST ACVREADING;
END
SET
  IF (ACVTHREE==1) THEN
    CONT -> "DIGIT 3";
  ENDIF
END
QUERY
  TEMPVAR TMP:STR;
  ACVTHREE = 0;
  CONT -> "DIGIT?";
  INST -> "DIGIT ", TMP, ",";
  IF (TMP=="3") THEN
    ACVTHREE = 1;
  ENDIF
END
END

```

Updating Measurement On Display — The UPDATELIST statement provides for the following action: When the "3 DIGITS" Radiobutton is selected, the control ACVREADING will be updated.

End the CONTROL block with END.

```

CONTROLGROUP
  CONTROL ACVTHREE:INT RADIOBUTTON @ 2.5,7.5
  STRING "3 DIGITS";
  ONEOFGROUP "ACCURACY";
  UPDATELIST ACVREADING;
END
SET
  IF (ACVTHREE==1) THEN
    CONT -> "DIGIT 3";
  ENDIF
END
QUERY
  TEMPVAR TMP:STR;
  ACVTHREE = 0;
  CONT -> "DIGIT?";
  INST -> "DIGIT ", TMP, ",";
  IF (TMP=="3") THEN
    ACVTHREE = 1;
  ENDIF
END
END

```

Setting the Resolution with IF-THEN — When this control is selected, the IF statement determines if it is being turned on. If it is, then the controller dialog sends "DIGIT 3," to the instrument.

The IF statement is terminated with ENDIF.

End the SETTING block with END.

```

CONTROLGROUP
  CONTROL ACVTHREE:INT RADIOBUTTON @ 2.5,7.5
    STRING "3 DIGITS";
    ONEOFGROUP "ACCURACY";
    UPDATELIST ACVREADING;
  END
  SET
    IF (ACVTHREE==1) THEN
      CONT -> "DIGIT 3";
    ENDIF
  END
  QUERY
    TEMPVAR TMP:STR;
    ACVTHREE = 0;
    CONT -> "DIGIT?";
    INST -> "DIGIT ", TMP, ",";
    IF (TMP=="3") THEN
      ACVTHREE = 1;
    ENDIF
  END
END

```

MEASUREMENT block Variables — The MEASUREMENT block, defined with the keyword **QUERY**, is used to send a query to the instrument for the state of the resolution setting.

The **TEMPVAR** statement declares a variable (local to the MEASUREMENT block) to store the response from the instrument.

The Assignment statement (`ACVTHREE = 0;`) INITIALIZES THE CONTROL VARIABLE TO 0.

```

CONTROLGROUP
  CONTROL ACVTHREE:INT RADIOBUTTON @ 2.5,7.5
  STRING "3 DIGITS";
  ONEOFGROUP "ACCURACY";
  UPDATELIST ACVREADING;
END
SET
  IF (ACVTHREE==1) THEN
    CONT -> "DIGIT 3";
  ENDIF
END
QUERY
  TEMPVAR TMP:STR;
  ACVTHREE = 0;
  CONT -> "DIGIT?";
  INST -> "DIGIT ", TMP, ", ";
  IF (TMP=="3") THEN
    ACVTHREE = 1;
  ENDIF
END
END

```

Controller and Instrument Dialogs — The controller dialog `CONT -> "DIGIT?";` queries the instrument for the number of digits of resolution.

The instrument dialog `INST -> "DIGIT ",TMP, ", ";` receives and parses the instrument response, placing the resolution in `TMP`. The instrument response is of the form "DIGIT 3;", so `TMP` will contain a 1 character string.

```

CONTROLGROUP
  CONTROL ACVTHREE:INT RADIOBUTTON @ 2.5,7.5
  STRING "3 DIGITS";
  ONEOFGROUP "ACCURACY";
  UPDATELIST ACVREADING;
END
SET
  IF (ACVTHREE==1) THEN
    CONT -> "DIGIT 3";
  ENDIF
END
QUERY
  TEMPVAR TMP:STR;

```

```

ACVTHREE = 0 ;
CONT -> "DIGIT?";
INST -> "DIGIT ", TMP, ",";
IF (TMP=="3") THEN
    ACVTHREE = 1;
ENDIF
END
END

```

Another IF Statement — This `IF` statement checks to see if the current resolution is 3 (3.5) digits and sets the control variable `ACVTHREE` to 1 (ON) if it is.

The `IF` statement is terminated with `ENDIF`.

End the `MEASUREMENT` block with `END`.

End the `CONTROLGROUP` block with `END`.

```

CONTROLGROUP
CONTROL ACVTHREE:INT RADIOBUTTON @ 2.5,7.5
STRING "3 DIGITS";
ONEOFGROUP "ACCURACY";
UPDATELIST ACVREADING;
END
SET
IF (ACVTHREE==1) THEN
    CONT -> "DIGIT 3 ";
ENDIF
END
QUERY
TEMPVAR TMP:STR;
ACVTHREE = 0 ;
CONT -> "DIGIT?";
INST -> "DIGIT ", TMP, ",";
IF (TMP=="3") THEN
    ACVTHREE = 1;
ENDIF
END
END

```

You can use this same `CONTROLGROUP` for the next four Resolution selections with a few modifications. The differences are in the control variable; in the control location; in the control title; in the controller dialog within the `SETTING` block; and in the expression within the `IF` statement.

For further information, refer to Appendix A, *TDM5120.ISD* Script Printout.

Many-to-one Control Groups

CHECKBOX and EDITBOX — Test instrumentation frequently requires setting and measurement results from more than one control based on the interdependence of those controls. In the case of the DM5120, the filtering function is turned on with a switch and keys are used to enter the filter value.

Within this script a CHECKBOX is used to turn on filtering and in conjunction (within the same CONTROLGROUP) an EDITBOX is used to accept an operator filter value input.

CHECKBOX Control — Define a CONTROLGROUP block for the CHECKBOX.

Define a CHECKBOX CONTROL block the same as with previous CONTROL blocks.

End the CHECKBOX control with `END`.

```
CONTROLGROUP
CONTROL ACVFILT:INT CHECKBOX @ 2.5,15.0
  STRING "AVE";
  UPDATELIST ACVREADING;
END
CONTROL ACVFILTAVAL:INT EDIT @ 8.7,15.0
  NUMROWS 1;NUMCOLS 5;
END
SETTING
  IF (ACVFILT==1) THEN
    CONT -> "FILTERVAL", ACVFILTVAL, ";FILTER ON";
  ELSE
    CONT - "FILTER OFF";
  ENDIF
END
MEASUREMENT
  TEMPVAR FILTON:STR;
  CONT -> "FILTER?;FILTERVAL?";
  INST -> " ", FILTON, ";", " ", ACVFILTVAL, ";";
  IF (FILTON=="ON") THEN
    ACVFILT=1;
  ELSE
    ACVFILT=0;
  ENDIF
END
END
```

EDITBOX Control — A second CONTROL block is defined for this CONTROLGROUP prior to Setting and MEASUREMENT blocks.

End the EDITBOX CONTROL block with END.

```

CONTROLGROUP
  CONTROL ACVFILT:INT CHECKBOX @ 2.5,15.0
    STRING "AVE";
    UPDATELIST ACVREADING;
  END
  CONTROL ACVFILTAVAL:INT EDIT @ 8.7,15.0
    NUMROWS 1;NUMCOLS 5;
  END
  SETTING
    IF (ACVFILT==1) THEN
      CONT -> "FILTERVAL", ACVFILTVAL, ";FILTER ON";
    ELSE
      CONT - "FILTER OFF";
    ENDIF
  END
  MEASUREMENT
    TEMPVAR FILTON:STR;
    CONT -> "FILTER?;FILTERVAL?";
    INST -> " ",FILTON, ";"," ",ACVFILTVAL, ";";
    IF (FILTON=="ON") THEN
      ACVFILT=1;
    ELSE
      ACVFILT=0;
    ENDIF
  END
END

```

A SETTING block With Two Variables — Both control variables defined with the CHECKBOX and the EDITBOX will be used in the CONTROLGROUP SETTING block.

In the SETTING block we use an IF statement to determine if the CHECKBOX is ON or OFF (the variable ACVFILT is equal to 1 if it is ON).

If the ACVFILT is equal to ON (1), we then use the second control variable as part of a command sent to the instrument.

End the IF statement with ENDIF.

End the SETTING block with END.

```

CONTROLGROUP
CONTROL ACVFILT:INT CHECKBOX @ 2.5,15.0
  STRING "AVE";
  UPDATELIST ACVREADING;
END
CONTROL ACVFILTAVAL:INT EDIT @ 8.7,15.0
  NUMROWS 1;NUMCOLS 5;
END
SETTING
  IF (ACVFILT==1) THEN
    CONT -> "FILTERVAL", ACVFILTVAL,";FILTER ON";
  ELSE
    CONT - "FILTER OFF";
  ENDIF
END
MEASUREMENT
TEMPVAR FILTON:STR;
CONT -> "FILTER?;FILTERVAL?";
INST -> " ",FILTON,";"," ",ACVFILTVAL,";";
IF (FILTON=="ON") THEN
  ACVFILT=1;
ELSE
  ACVFILT=0;
ENDIF
END
END

```

Measurement With Two Control Variables — First we declare a temporary variable, `FILTON`, to contain the `ON` or `OFF` status of the instrument.

Then we query the instrument for the `ON` or `OFF` status of the filter and the current filter value.

Finally, we read the instrument response and parse the result, placing the data in `FILTON` and `ACVFILTVAL` (temporary and control variables).

```

CONTROLGROUP
  CONTROL ACVFILT:INT CHECKBOX @ 2.5,15.0
    STRING "AVE";
    UPDATELIST ACVREADING;
  END
  CONTROL ACVFILTAVAL:INT EDIT @ 8.7,15.0
    NUMROWS 1;NUMCOLS 5;
  END
  SETTING
    IF (ACVFILT==1) THEN
      CONT -> "FILTERVAL", ACVFILTVAL, ";FILTER ON";
    ELSE
      CONT - "FILTER OFF";
    ENDIF
  END
  MEASUREMENT
    TEMPVAR FILTON:STR;
    CONT -> "FILTER?;FILTERVAL?";
    INST -> " ", FILTON, ";", " ", ACVFILTVAL, ";";
    IF (FILTON=="ON") THEN
      ACVFILT=1;
    ELSE
      ACVFILT=0;
    ENDIF
  END
END

```


An IF Statement for Conversion — This IF statement first examines the FILTON temporary variable to see if it contains "ON". If so, then the control variable ACVFILT is given the value 1. Otherwise, ACVFILT is given the value 0.

The integer 1 turns the Filter CHECKBOX display ON, and 0 turns it OFF (there is an X in the box display when the switch is ON and nothing in the box when it is OFF).

```

CONTROLGROUP
CONTROL ACVFILT:INT CHECKBOX @ 2.5,15.0
  STRING "AVE";
  UPDATELIST ACVREADING;
END
CONTROL ACVFILTAVAL:INT EDIT @ 8.7,15.0
  NUMROWS 1;NUMCOLS 5;
END
SETTING
  IF (ACVFILT==1) THEN
    CONT -> "FILTERVAL", ACVFILTVAL, ";FILTER ON";
  ELSE
    CONT - "FILTER OFF";
  ENDIF
END
MEASUREMENT
  TEMPVAR FILTON:STR;
  CONT -> "FILTER?;FILTERVAL?";
  INST -> " ", FILTON, ";", " ", ACVFILTVAL, ";";
  IF (FILTON=="ON") THEN
    ACVFILT=1;
  ELSE
    ACVFILT=0;
  ENDIF
END
END

```

List or Listbox — This Listbox control is used to allow range selections. In this DM5120 AC VOLTMETER view there are six range settings. A table in the DM5120 Instrument *Interfacing* Guide (and other associated manuals) provides all function/range and range-number relationships.

CONTROLGROUP for Function Ranges — The CONTROLGROUP and Control type blocks are defined.

As with previous examples, we define the CONTROLGROUP block and CONTROL block. We specify the control variable and its data type; the control LISTBOX and its position; and the size of the Listbox and its title.

```
CONTROLGROUP
CONTROL ACVRANGE:STR LISTBOX @ 18.7,8.0
  NUMROWS 5;NUMCOLS 5;
  CONTROLTITLE "RANGE";
  STRING " AUTO", "300 MV", " 3 V", " 30 V", "300 V";
  TOSCRIPT "AUTO", "1", "2", "3", "4";
  UPDATELIST READING;
END
SETTING
  CONT -> "RANGE ", ACVRANGE;
END
MEASUREMENT
  CONT -> "RANGE?";
  INT -> " ", ACVRANGE, " ";
END
END
END
END
```

Setting Range Strings — The STRING statement defines the contents of each row of the Listbox.

The TOSCRIPT statement defines the value that will be found in the control variable in the SETTING block.

```
CONTROLGROUP
CONTROL ACVRANGE:STR LISTBOX @ 18.7,8.0
  NUMROWS 5;NUMCOLS 5;
  CONTROLTITLE "RANGE";
  STRING " AUTO", "300 MV", " 3 V", " 30 V", "300 V";
  TOSCRIPT "AUTO", "1" "2" "3" "4";
  UPDATELIST READING;
END
SETTING
  CONT -> "RANGE ", ACVRANGE;
END
MEASUREMENT
  CONT -> "RANGE?";
  INT -> " ", ACVRANGE, " ";
END
END
END
END
```

There is a one-for-one relationship between the entries in the `STRING` statement and those in the `TOSCRIPT` statement. For example, the third element of the `STRING` statement is selecting the `3 V RANGE AS A LISTBOX DISPLAY`. THE THIRD VALUE OF `TOSCRIPT` is `2` for the control variable `ACVRANGE`.

Function RANGE SETTING block — This `SETTING` block functions the same as previously defined `SETTING` blocks with the exception that `ACVRANGE` will contain one of the elements from the `TOSCRIPT` statement.

End the `SETTING` block with `END`.

```

CONTROLGROUP
  CONTROL ACVRANGE:STR LISTBOX @ 18.7,8.0
    NUMROWS 5;NUMCOLS 5;
    CONTROLTITLE "RANGE";
    STRING " AUTO", "300 MV", " 3 V", " 30 V", "300 V";
    TOSCRIPT "AUTO", "1", "2", "3", "4";
    UPDATELIST READING;
  END
  SETTING
    CONT -> "RANGE ", ACVRANGE;
  END
  MEASUREMENT
    CONT -> "RANGE? ";
    INT -> " ", ACVRANGE, "; ";
  END
END
END
END

```

Function RANGE MEASUREMENT block — The `MEASUREMENT` block functions the same as previously defined `MEASUREMENT` blocks with the exception that the contents of `ACVRANGE` must correspond to one of the elements in the `TOSCRIPT` statement.

End the `MEASUREMENT` block with `END`.

End the `CONTROLGROUP` with `END`.

End the View block with `END`.

You can add views at this point if you want more functions in your script (`OHMS`, etc.).

End the script with `END`.

```
CONTROLGROUP
CONTROL ACVRANGE:STR LISTBOX @ 18.7,8.0
NUMROWS 5;NUMCOLS 5;
CONTROLTITLE "RANGE";
STRING " AUTO","300 MV"," 3 V"," 30 V","300 V";
TOSCRIPT "AUTO","1","2","3","4";
UPDATELIST READING;
END
SETTING
CONT -> "RANGE ",ACVRANGE;
END
MEASUREMENT
CONT -> "RANGE?";
INT -> " ",ACVRANGE,"-";
END
END
END
END
```

Save Your Script File — With Windows Notepad you simply call **Save As...** and give the file a name (in this case, MY5120.ISD).

NOTE

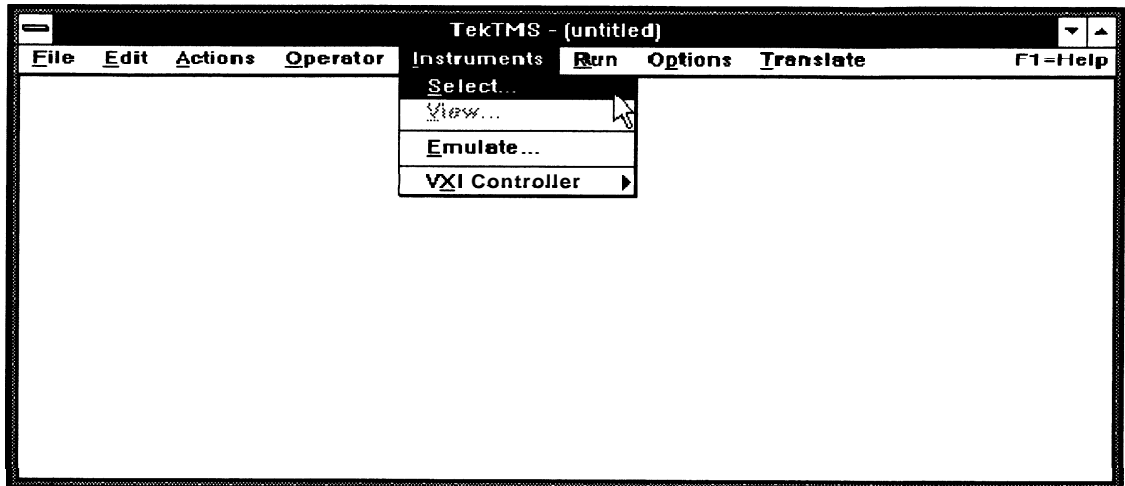
Remember, you cannot name an *.ISD file with a name that is the same as a **Device Name** in your GPIB IBCONF file.

Testing Scripts

With this TDM5120.ISD model script, you have a driver to control the Tektronix DM5120 digital multimeter on a GPIB bus. The next action, after writing the script, is to test the operation of your script.

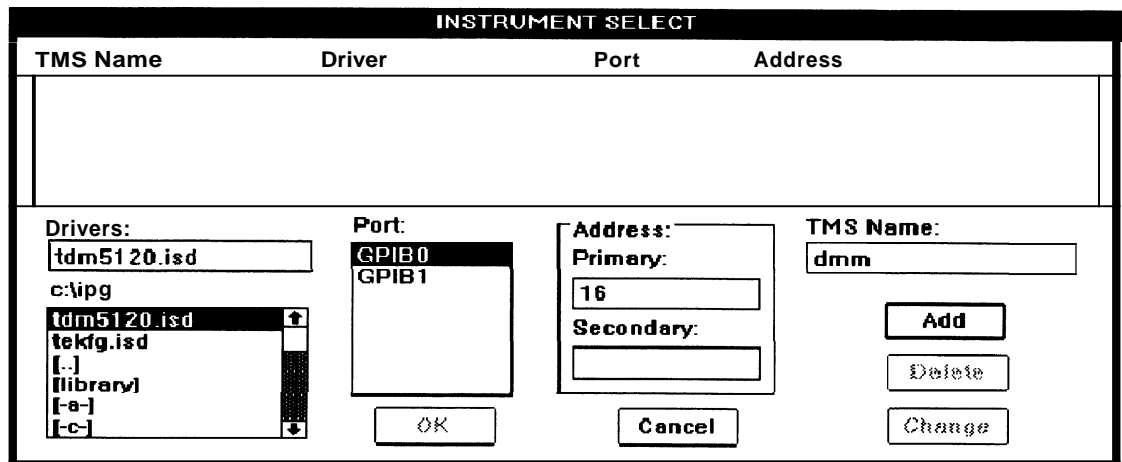
Initializing the Script

Initialize IPG from Windows. Select **Instruments** from the main menu bar, then select **Select...** from the Instruments menu, as shown in the following illustration.



Select the TDM5120.ISD script from the Instruments Select dialog box, as shown in the following illustration. You will see the script file name appear in the **Drivers:** edit control box; and *GPIB0* and *GPIB1* appear in the **Port:** list box.

After entering a **Primary Address:** for your DM5120 (the factory default is 16) and a **TMS Name:** (*DM5120*), select the **Add** Pushbutton, then select **OK** to load the script.



If IPG will not accept your new script, refer to A Rejected Script, following in this section.

Activating Controls

Checkbox — The Checkbox requires a single click of the mouse to activate.

Listbox — The List or Listbox control is implemented with a double click of the mouse, or a single click of the mouse, followed by pressing the ENTER or RETURN keyboard key.

Pushbutton and Radiobutton — The Pushbutton or Radiobutton requires a single click of the mouse to activate.

Checkbox with Editbox — A many-to-one group such as a Checkbox with an Editbox requires a single click of the mouse on the Checkbox and pressing the RETURN or ENTER keyboard key after the Editbox value has been entered.

When you test your AVE control from the TDM5120.ISD script, you may need to change both controls before activating them simultaneously.

Change the state of your AVE control by pointing and clicking with the mouse. Then move the cursor to the *filterval* Editbox (next to AVE) and click. Type in the filter value you want and press the ENTER or RETURN keyboard key to execute both controls simultaneously.

A Rejected Script

Error messages will be shown on your display if the script cannot be loaded by IPG. You will see one or more error messages, and one parse-failed message.

Error Messages — Explanatory error messages will be displayed on your screen when a script is rejected. Line numbers and the type of error will be indicated.

At this point you should click on the error **OK** Pushbutton(s) and on the parse failed **OK** Pushbutton.

Now **Cancel** the IPG loading function. Return to the word processor and repair the script. Then try loading again.

The Reloading Process — If you have a script loaded in IPG and you need to replace it with the same script after its has been modified, you must first delete the residing script from the IPG program. A change made in your word processor and then saved on a disk, will not update the script in your controller RAM.

To delete the old script, select **Instruments** from the IPG main menu bar, then select **Select...** Highlight the name of the script you want to delete, select the **Delete** Pushbutton, then select the **OK** Pushbutton. (If you do not select the **OK** Pushbutton, the script will not be deleted.)

Repeat the initializing process to load your modified script.

NOTE

If you have more than one version of the same script in a test procedure currently in IPG, you must delete all versions in order to reload a modified script. If any version of the same script is left in RAM, it will be used by IPG for reloading.

Any time you make a change to a script while in IPG, you must delete the related *.ISD in your working memory. The *.ISD is always called to RAM from disk for compilation with the IPG software.

Generating Test Procedures

Refer to your Interactive Procedure Generator Users Manual for detailed instructions on test procedure generation.

Modifying Scripts

If you need to generate new scripts, it may be easier to copy and modify an existing script. To do this, perform the following steps:

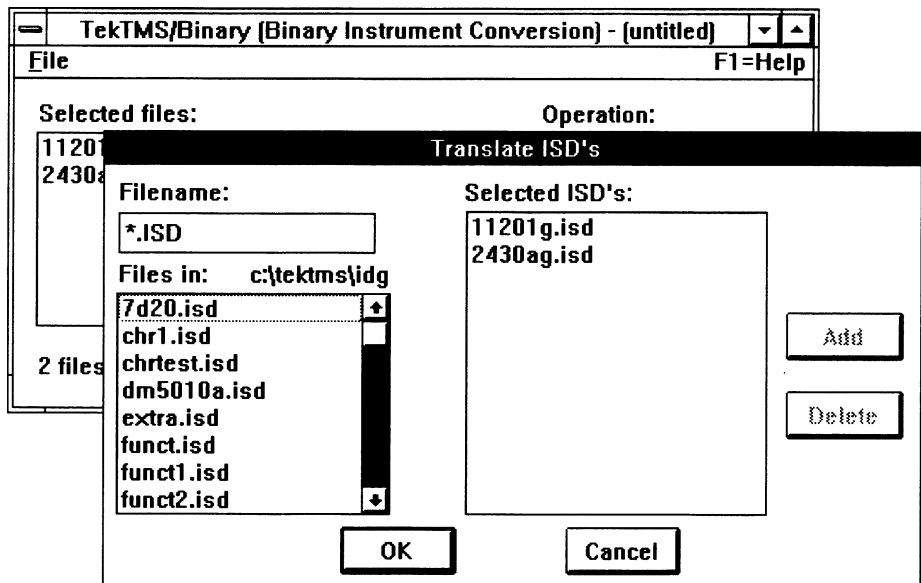
1. **Copy Existing Script.** Make a copy of the script and give it a new name. The new name should reflect the make and name of the instrument. It should also have an .ISD extension.
2. **Mark Changes.** Make a printout of your comparable ASCII script file. Use the printout to locate instrument commands. Mark the changes necessary. Use instrument interfacing manuals to determine equivalent and relevant commands.
3. **Make Changes.** With your word processor (any one with ASCII output) insert the new instrument commands where necessary in the existing script to make a new script that will work with the different instrument. When you have finished with making substitutions of commands in a script, save the modified script with its new name.

Binary ISD Files

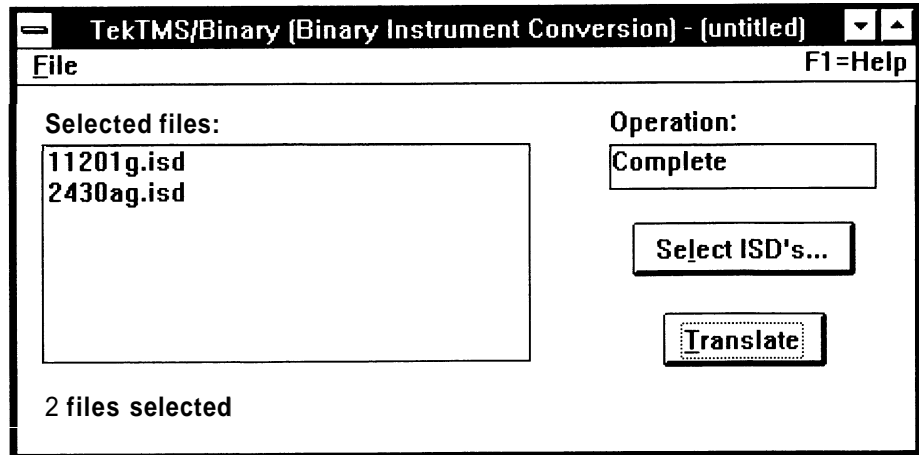
A Binary ISD file consist of a normal ASCII ISD file converted into a pre-parsed binary format. The advantage of a Binary ISD file over an ASCII ISD file is that a Binary ISD file loads faster than an ASCII ISD file which represents the same base ISD.

Using Binary

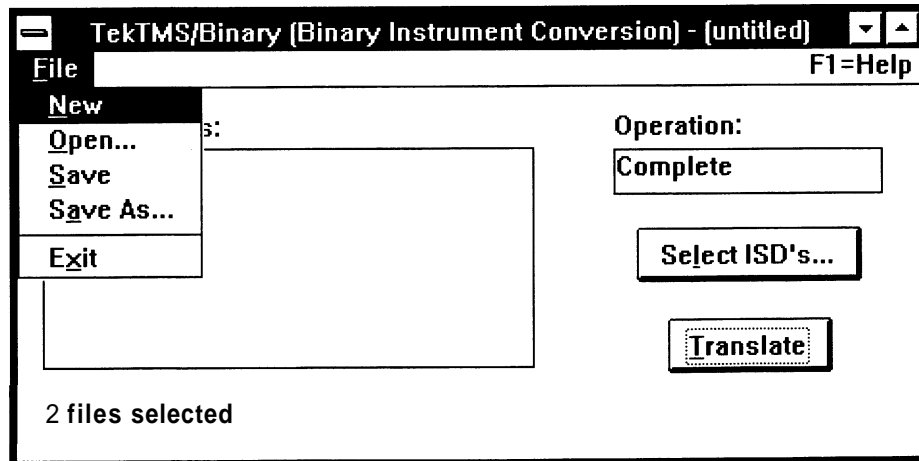
The MS Windows application Binary is simple to use. Start the application (typically by double clicking on the **Binary** icon in the TekTMS application group), **Select** the ISD files to convert from ASCII to binary, and then press the **Translate** button.



The Binary application reads each of the selected ISD files and creates a matching file with the .ISB file name extension. As each ASCII ISD file is translated the Operation: text box displays each view name as it is translated. When translation of all selected files is complete, the Operation: text box displays the status **Complete**.



The **File** menu item allows the list of selected ISD files to be saved for later use. Translation from ASCII to binary is normally only done once, but if the ASCII versions are modified, the binary form must be created again. Saving the list of translated ASCII ISD files makes creation of new binary form ISD files easy.



The **File** menu contains several choices.

- **New** — Delete the current list of ISD files from the selections: control.
- **Open** — Open an existing Binary .CFG file. The .CFG file contains a list of ISD file names from a previous use of Binary.
- **Save** — Save the current list of selected ISD files. If the .CFG file is not named, the Save As function is called.
- **Save As** — Save the current list of selected ISD files under a new name .CFG file.
- **Exit** — Close Binary. If the current list of selected ISD files has been changed since the last save, a message box will appear asking to verify whether or not the list of ISD files should be saved or abandoned.

Using Binary ISD Files

Binary ISD files may be used by either TekTMS/IPG or by TekTMS/IDG. Both applications show a list of available file with the extension .IS?. Any file with an extension having the first two characters 'IS' will be shown in the selection list. Select the .ISB file and proceed as though you had selected an .ISD file.

Section 4

Help Editor

Help Editor

The Help Editor is a utility program within IPG that allows you to create or modify help messages (.HLP files) for your scripts. These messages are available as you use scripts with IPG to configure "soft" front panels for programmable instrumentation as part of IPG test procedure generation. The messages display additional information about the soft front panel.

IPG Instrument Script Language (explained in previous sections) allows the user to create new scripts for programmable instruments in the IPG test procedure generation system. The Help Editor allows you to create help message files for new scripts; and to place in the message files those messages you find most helpful.

The Help Editor also allows you to customize messages in existing script help files. You can use the Help Editor to rewrite help messages, for example, in another national language. Also, the Help Editor can be used to print a help message file.

The Help Editor is part of the IPG software package; it is an application that runs under Microsoft Windows.

Refer to your Interactive Procedure Generator (*IPG*) Users Manual for instructions on how to use *.HLP files with related scripts. Script and help files have identical names preceding their extensions. However, in using **Yelp** with IPG you will not see the help file names.

Learning to Use the Help Editor

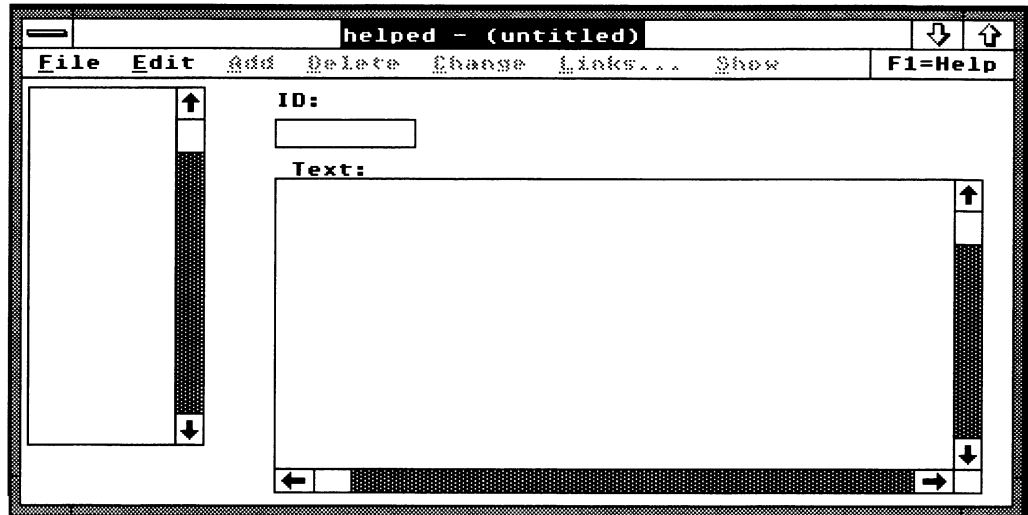
This part of the manual is organized in tutorial format and describes how to use the Help Editor to do the following:

- Modify a help message (.HLP) file for an instrument soft front panel script
- Create a new help file for use with a new, user-created script, add new messages, delete a message, and modify a message in a script help file
- Print a help message file

Starting the Help Editor Program

To begin using the Help Editor program, start Microsoft Windows, and then run HELPED.EXE.

The following illustration shows the Help Editor window as it appears when you start the program.



Help Message Files — Instrument front panel help messages are in a file associated with the specific script that defines the instrument soft front panel. As an example, the help messages for the DM5120 soft front panel are in a file, TDM5120.HLP, that is associated with the DM5120 script file, TDM5120.ISD. The TDM5120.HLP and TDM5120.ISD files are provided with IPG software.

You can modify, add, and delete messages in a soft front panel help file, provided that you also make the appropriate changes to the script. (For example: if you change a script control name and variable for the DM5120, you must change that name in both DM5120 files, TDM5120.ISD and TDM5120.HLP)

Also, if you create a new script, you can use the Help Editor to create a file of help messages for the new instrument soft front panel. The messages from help files are available while the new soft front panel is being set up during IPG test procedure generation.

Help File and Script File Internal Relationships — The **Instrument** menu item from **Help** in the IPG front panel display finds the file name given by the script code. The script **CONTROL** command finds the related ID and the name given by **CONTROL** in the script. Therefore, you must match the script name and each control name when modifying or developing script and help files.

For example: TDM5120.ISD is opened with SCRIPT "DM5120"

```
REM TEKTRONIX DM5120 DIGITAL MULTIMETER
```

```
SCRIPT "DM5120"
```

```
GPIB
```

```
END
```

The Help function uses the script name to find the related help message. When **Help**, is called from the IPG front panel display, a help message related to Script DM5120 is shown. A part of the actual script is shown above to help illustrate that relationship.

For each control, the Help function uses the control variable to find the related help message. When you select **Help** and Control from the IPG front panel display, a help message related to the currently focused control is shown. A related part of the actual script is shown below to help illustrate that relationship.

```
CONTROLGROUP
```

```
CONTROL ACVFILT:INT CHECKBOX @ 2.5,15.0
```

```
STRING "AVE";
```

```
UPDATELIST READING;
```

```
END
```

```
CONTROL ACVFILTVAL:INT EDIT @ 8.75,15.0
```

```
NUMROWS 1;NUMCOLS 5;
```

```
END
```

If the cursor is on the Checkbox control (STRING AVE), the ACVFILT ID in the help file is tied to that control. If the cursor is in the Edit control box, the ACVFILTVAL ID in the help file is tied to that control.

Help Message Parts — in a help message file, each message consists of the items described in the following descriptions. Refer to the following illustration for assistance in understanding the message item relationships.

Message ID 123, Caption, and Text

ID	123
Text	<p>Messages Messages are composed of a caption (the first line in the text box) and text (remaining lines). The caption and text lines will be centered in a message box that is automatically sized to hold the information</p>

Labels and Destinations for Message ID 123

Label	Destination
OK	-1
OK	-1
Help	136

Message ID 123 displayed

Messages	
<p>Messages are composed of a caption (the first line in the text box) and text (remaining lines). The caption and text lines will be centered in a message box that is automatically sized to hold the information</p>	
<input type="button" value="OK"/>	<input type="button" value="Help"/>

Message ID 136, Caption, and Text

ID	136
Limits	<p>Limits The caption is the first line of text. The message text can be as long as 24 lines of text.</p>

Limits	
<p>The caption is the first line of text. The message text can be as long as 24 lines of text</p>	

Remove message

Display Message ID 136

HELP EDITOR

DISPLAYED MESSAGES

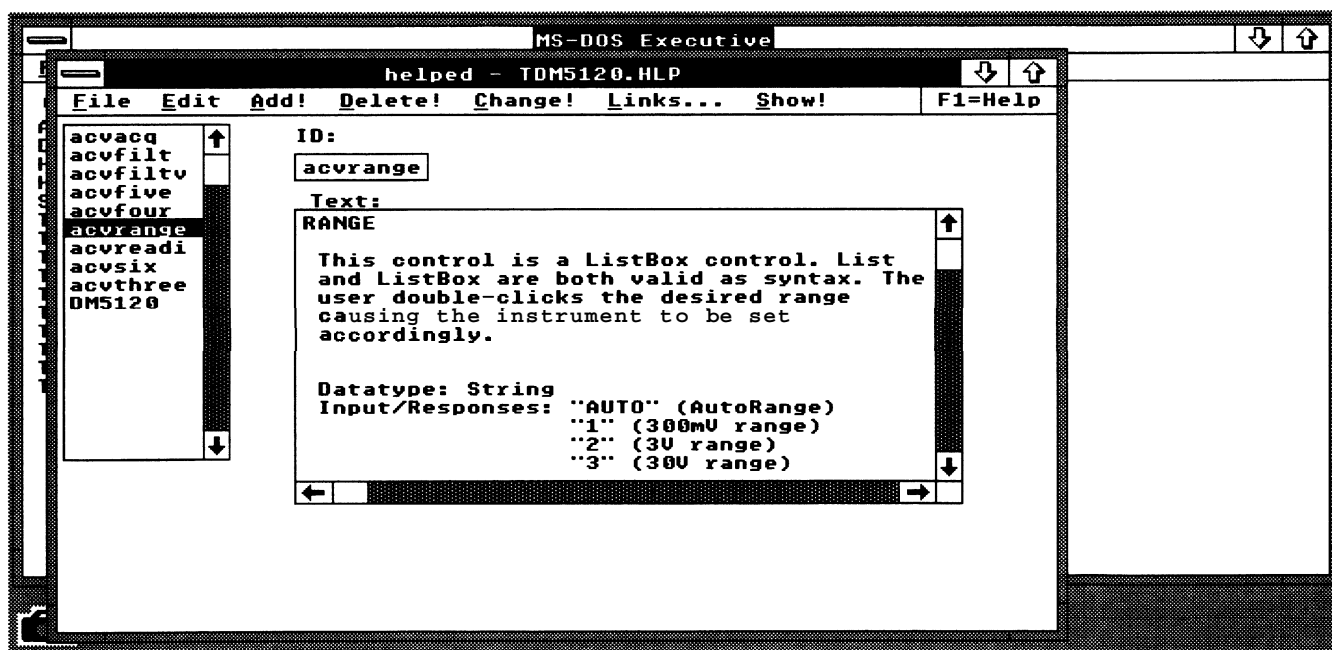
- **Message ID** — In the **ID:** box, a unique name or number is used to identify a particular message in the message file. The ID is used by the application to call a message and display it. The ID can be uppercase alpha characters and/or positive numerals, and it is limited to eight characters.
- **Caption** — The name above the message text in the **Text:** box; in the example, `MESSAGES`. The name is limited to one line in length; it is the first line of text in the Message text box.
- **Text** — The message text, in the **Text:** example, `MESSAGES ARE COMPOSED, .` The message text is limited to 24 lines. When the message is displayed, the Help Editor automatically sizes the message box to hold the message text and caption, and justifies each text line.
- **Labels** — The labels in the command buttons (i.e., `OK`). The command button labels can be any text you choose. A message can have as many as five buttons.

- Destination** — The action assigned to a command button. The destination determines what happens when you select the associated command button. The destination can be a negative number, which removes the message from the screen. For example, -1 is usually used as a destination for OK. Or, the destination can be the ID of another help message; in this case, that message is displayed when the associated command button is selected.

Editing Help Messages — The following text describes how to edit a message in a help file.

Open the TDM5120.HLP File — In the Help Editor window, select **Eile**, then **Open**. A **File name:** Listbox will appear. The files listed in the Listbox are those in the indicated directory. Select the drive and directory that contains TDM5120.HLP. Then select the TDM5120.HLP file with a double-click of your mouse; or with a single-click on the file name and a single-click on the **Open** Pushbutton.

The Help Editor lists the messages in the file, in the Listbox, by ID, as shown in the following illustration.



- Select and Edit a Message or Caption** — Select a message in the Listbox that you want to edit; for example, ACVRANGE. The message ID is shown in the ID box, and the message caption and text are shown in the **Text** box. To change the message, place the pointer on the text, click the mouse, and make the desired changes. The first line of text in the **Text** box is the **Caption**; this appears as the title of the message when the message is displayed.

You can use the **Cut**, **Copy**, and **Paste** features of the Edit menu on highlighted text in the **Text** box. Note that the text does not automatically word-wrap; you must insert carriage returns where you want a line to end. Also, keep in mind that each text line will automatically be left justified in the message box, when the message is displayed. You might want to add a space to the left at each message line for clarity.

Now to process an example, add the sentence List and Pushbutton are *both valid as syntax* to the **Text:** after `...PUSHBUTTON CONTROL`. The illustration shows the Help Editor window, with the suggested changes made to the text of message ID ACVRANGE.

NOTE

You may want to make a copy of your *TDM5120.HLP* file, using another name, for future reference. When you modify the file and save it in HELPED, it will stay modified. You can also use your original or backup software diskettes to restore *TDM5120.HLP* to the way it originally was.

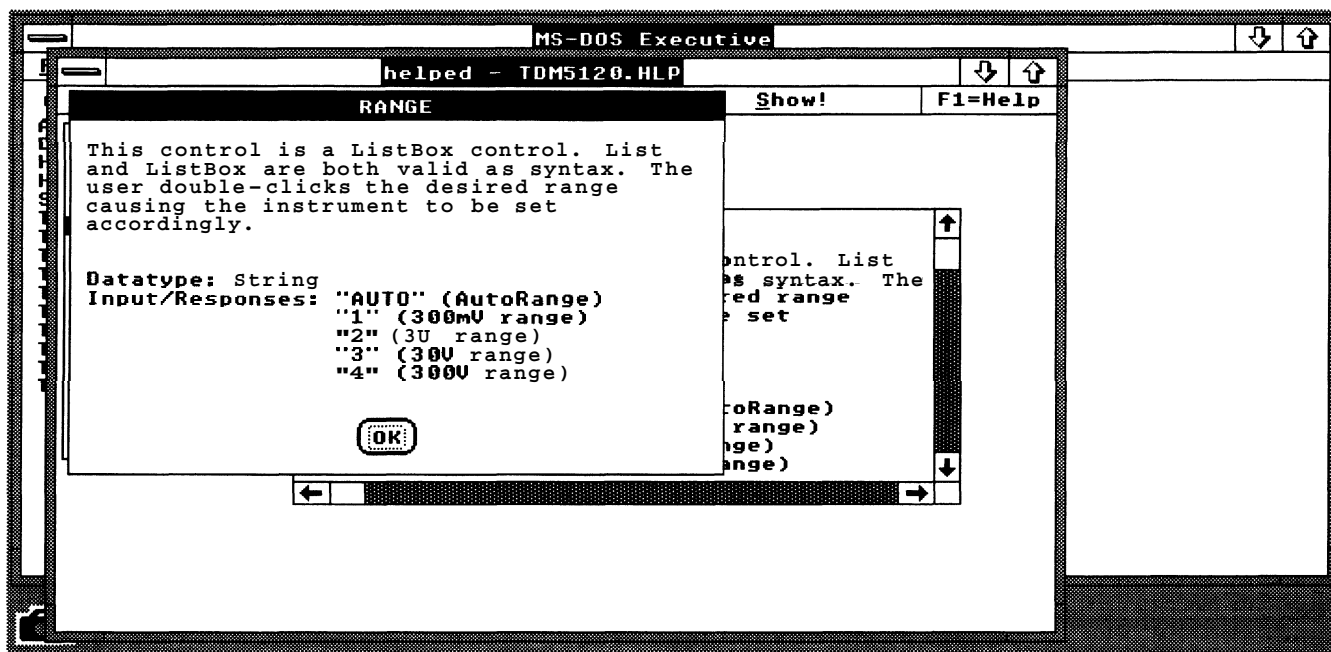
When you are done editing the message, select **Change!**; this updates the TDM5120.HLP file with the changed message.

NOTE

If you do not select **Change!** while your edited message is displayed, your changes will be lost.

Now, select **Show!** to see how your edited message looks. The following illustration shows the message modified in the Help Editor **Text:** window.

To remove the displayed message, select the **OK** Pushbutton in the displayed message box. The **OK** Pushbutton in the message box is called the Command Button Label.



Command Button Labels — Labels are the names of the command buttons that appear at the bottom of a message. Each label is linked to a destination, either to another message in the same *.HLP file, or to a negative number. These destinations determine what happens when a particular command button is selected while the message is displayed. For example, if the label for a command button is linked to the number -1 , the message is removed from the window when **OK** is selected. (Usually, the command button label **OK** is linked to -1 .)

NOTE

A link to -1 is necessary to ensure that the help message can be removed from the screen after it is called from IPG. If you do not have a -1 destination link with a command button (such as **OK**), you will have to soft boot the controller to remove the help message.

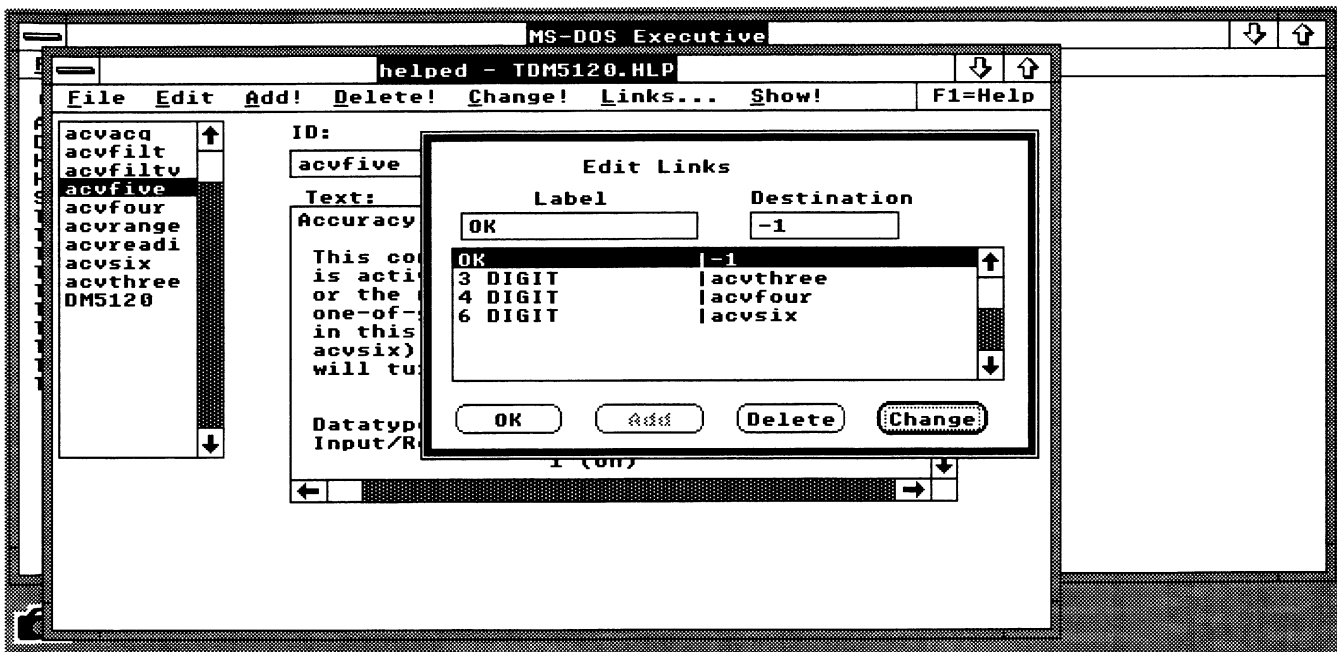
A command button label can also be linked to another message. When that command button is selected while the original message is displayed, the linked message is then displayed. In such a case, the destination of the label in the original message is the ID of the linked message.

You can link a number of messages to one message, in a star fashion using a label and destination for each linked message, or chain a series of messages, or you can use a combination of these methods.

NOTE

The display panel is sized to include the button text; however, the box around the text may be cut off. If this occurs, add blanks to the end of one line of message text to expand the display panel.

Displaying Message Links — You can look at the labels and destinations for a message. With the message ACVFIVE displayed (click on ID ACVFIVE from the Listbox), select **Links....** The Help Editor displays the **Edit Links** box. Labels are the names of the command buttons; these are listed on the left in the box. Destinations are listed on the right. The Label and Destination at the top of the list are also shown in the **Label** and **Destination** boxes, as shown in the following illustration.

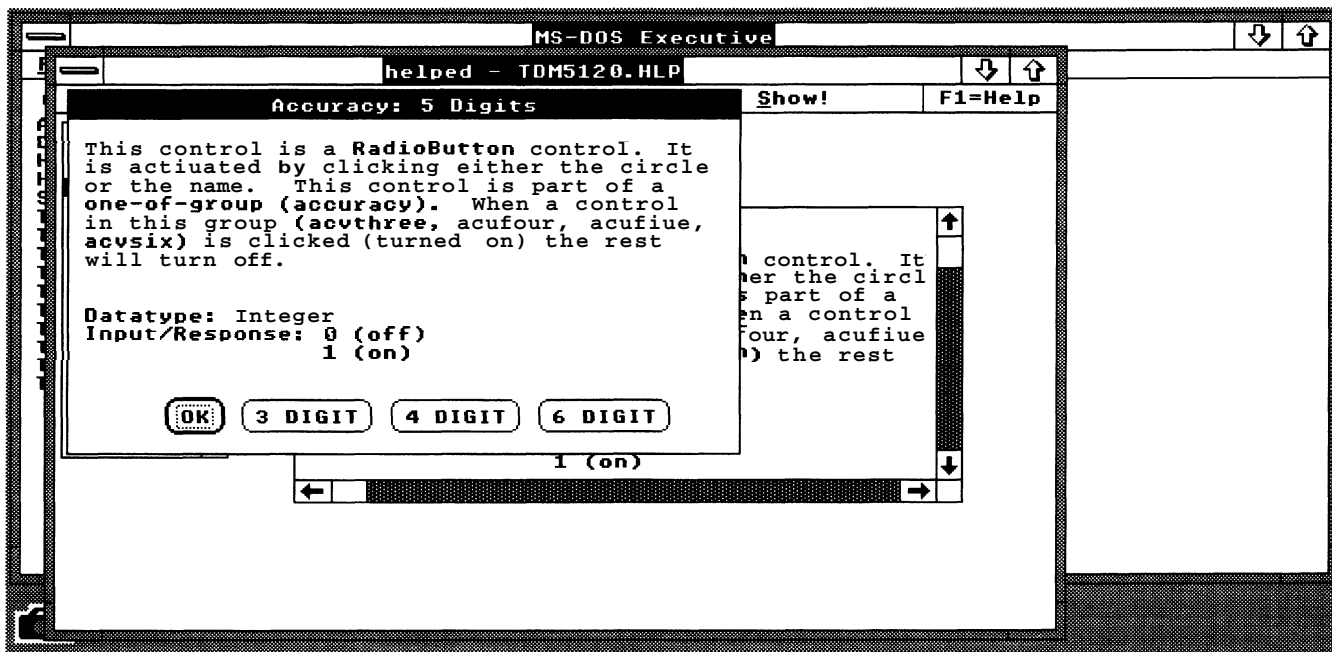


Select the **OK** PushButton to Exit the Edit Links Window

Displaying Command Buttons — The command button labels will be displayed when you select **Show!**. The help message ACVFIVE will have four command buttons set up, as shown in the following illustration.

(When you are in IPG and select **Help** for **Instrument** or **Control**, and you then select **OK**, the message is removed from the display.)

Now, select **OK** to remove the box from the HELPED window.



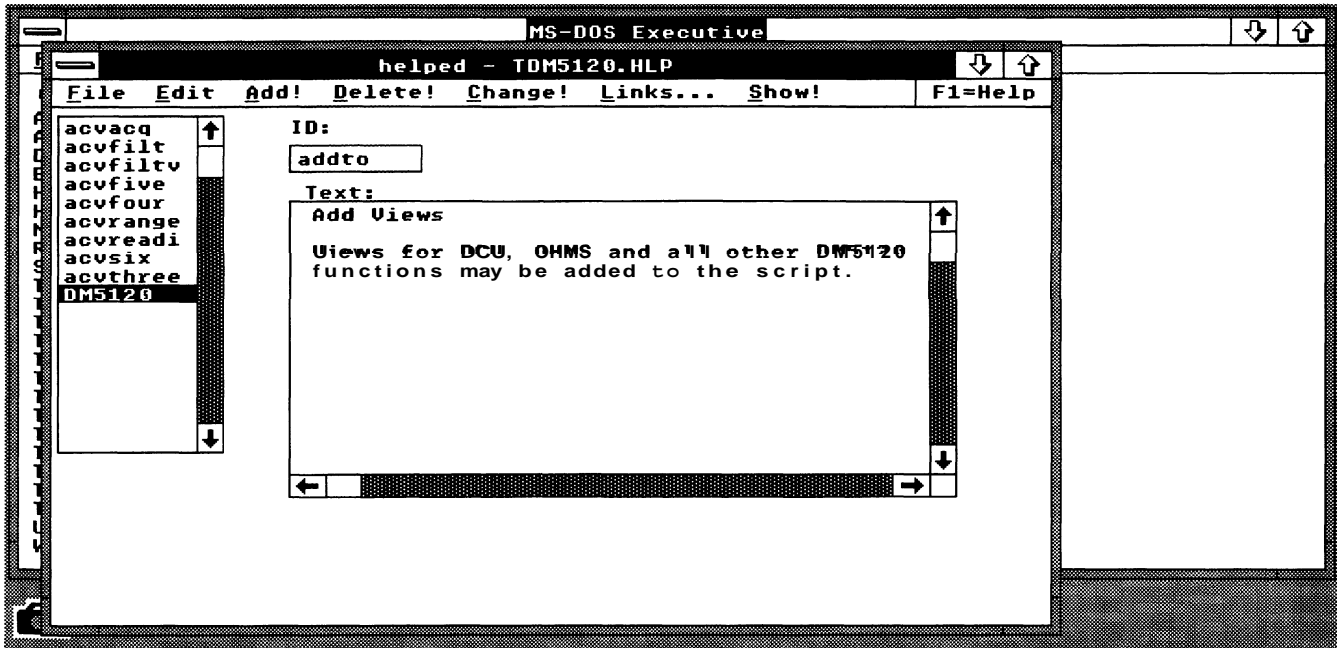
Adding and Linking a Message

Adding — You can add a new message, and link that message to ensure that it can be removed when it is used in IPG. You can also link it to an original message in the TDM5120.HLP file; or to another new message.

For example, we will add the message ID *addto* and create a message that reads: Views for *DCV*, *OHMS* and all other *DM5120* functions may be added in the script. Then we will link that new message to an **OK** Pushbutton, and to a message that exists in the file.

Call up ID *DM5120*. Change the *DM5120 ID:* to *addto*. Change the **Text:** Caption to Add Views. Change the text to read: Views for *DCV*, *OHMS* and all other *DM5120* functions may be added to the script. Refer to the following illustration for an example of making changes to an existing file to create a new message.

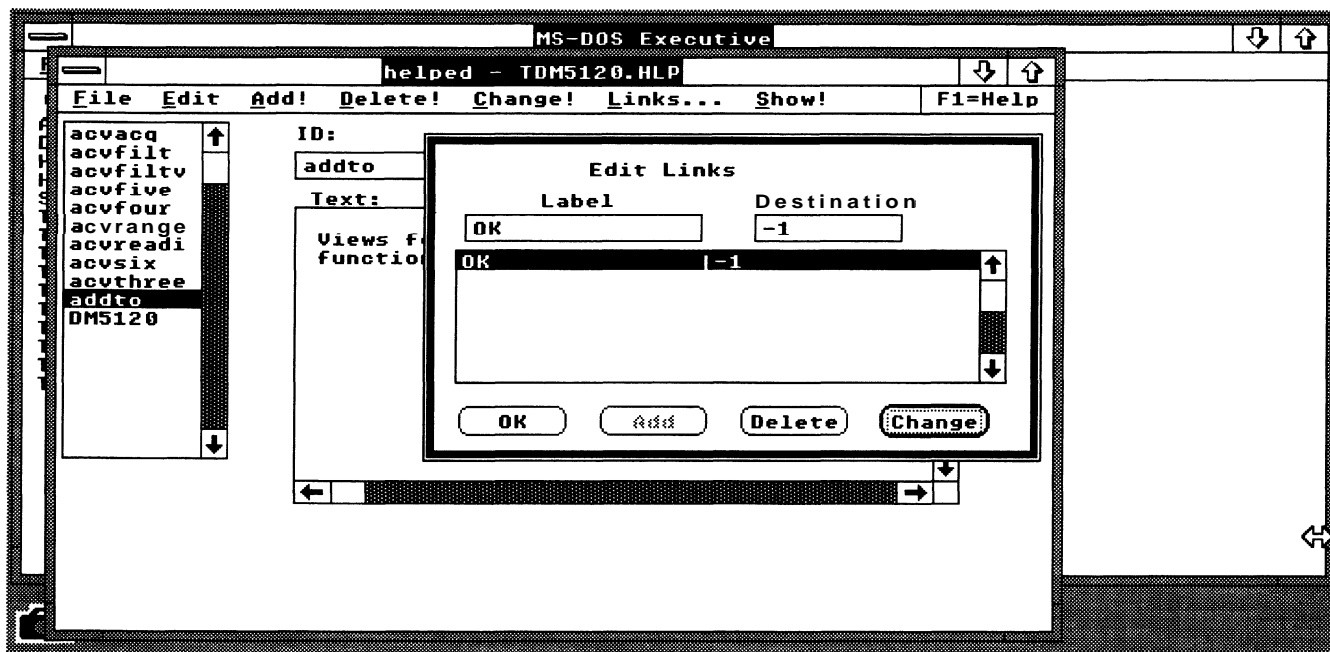
You must select **Add!** to implement the new material, and you must select **Save** from the File menu to save the TDM5120.HLP file for use.



Linking

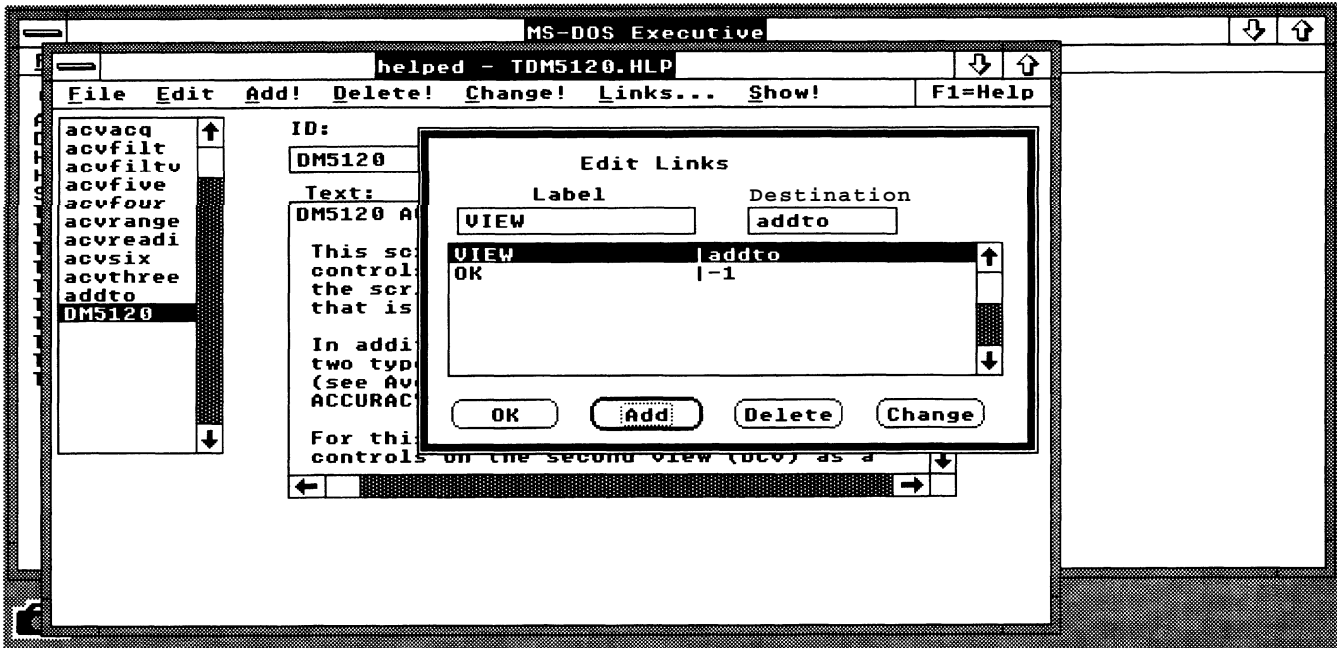
Linking the New Message — We will now link the new message by selecting **Links...** while the ID `ADDTO` is still on the screen. Type **OK** in the **Label** box; and type `- 1` in the **Destination** box, as shown in the following illustration. As a reminder, the **OK** pushbutton will allow you to remove the help message when it is called (without it you will not be able to get out of that help message).

Select the **Add** pushbutton and then the **OK** pushbutton to get back into the Help editor.



Linking To An Existing Message — We will now link the new message to an existing message (DM5120) with the command button `v■w`.

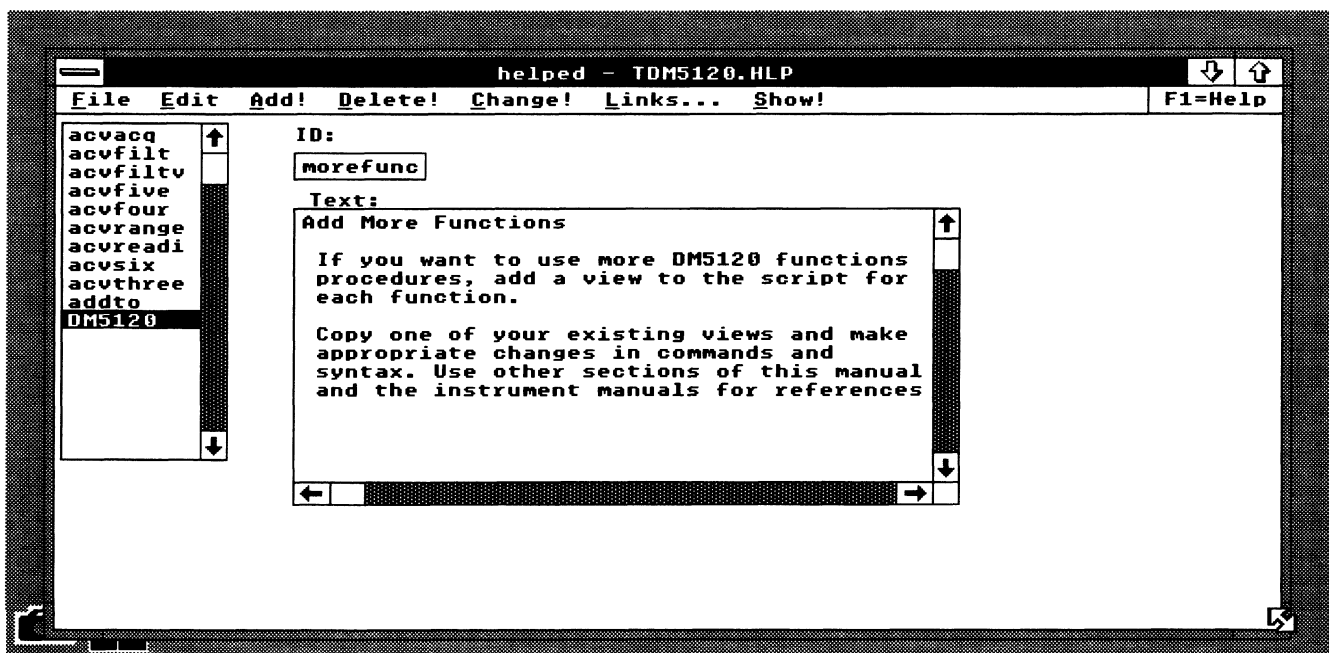
Call ID DM5120 again. Select **links...** from the Help Editor main menu. Type **VIEW** in the **Label** box and **addto** in the **Destination** box, as shown in the following illustration. Select the **Add** pushbutton and then the **OK** pushbutton. Select **Show!** to view your new command buttons. Select the **OK** pushbutton to exit.



Creating a New Message

To create a new message, select a message in the TDM5120.HLP list (possibly one that is similar to the message you want to add). For this exercise, select **ID** DM5120, and display its **ID:** and **Text:**.

Change the **ID** to what you want to assign to the new message; then change the message caption and text so that the **Text:** box contains the information you want the new message to display. For this example, change the **ID** to morefunc; change the caption (the first line of text in the **Text:** box) to **Add More Functions**; and change the text " to read relative to adding views in the script. In this case add: If you want to use more DM5120 functions..., as shown in the following illustration, then select **Add!** in the menu bar. The name of the new message is added to the list of messages in the Listbox. Use the scroll bar in the Listbox, if necessary, to check that the new message **ID** is listed.



Now select **Links...** to display the **Edit Links** box. Place the pointer in the **Label** box and type OK; move the pointer to the **Destination** box and type “-1”; then select the **Add** pushbutton. This will add the command button OK, so that you can remove the displayed message when you use it in IPG. Finally, select the **OK** pushbutton.

Linking the New Message to Another — As with the previous linking example, if you want to link the new message to another message in TDM5120.HLP, do the following: select the message you want to link your new message to, like ACVACQ; then select **Links....** Place the pointer in the **Label** box and type in the name of the command button you want to add to the message box that will call up your new message (i.e., VIEW2). Then, move the pointer to the **Destination** box and type the ID you gave the new message, morefunc. Then select the **Add** pushbutton and select the **OK** pushbutton. You can display the message ACVACQ, to see that the new control was added to it. To do this, select message ACVACQ, and then select **Show!** in the menu bar. To remove the displayed message, select **OK**.

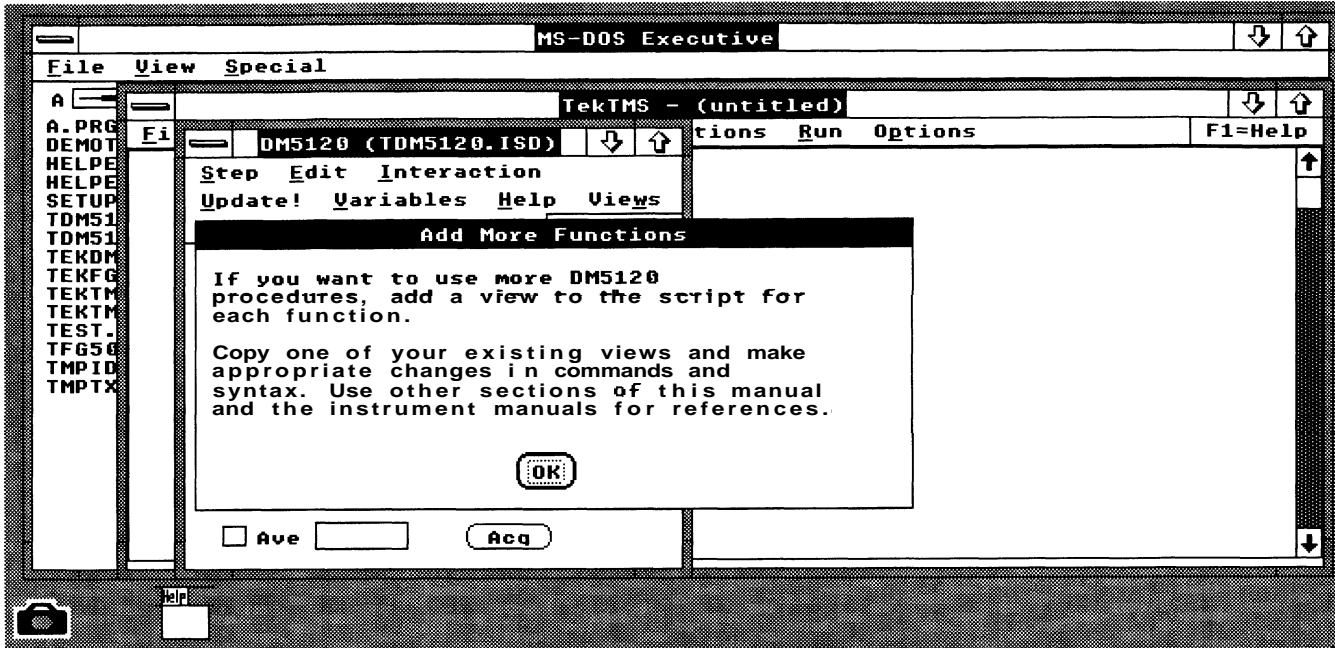
Testing the New Message — To try out the new message, save the TDM5120.HLP file by selecting **File** in the Help Editor window, and then selecting **Save**; reduce the Help Editor to an icon.

Start IPG, and display the TDM5120.ISD front panel. (Refer to Testing Scripts in Section 3 for instructions with using TEKTMS.EXE, if necessary.)

Select the **Acq** pushbutton, then select **Help**. Select **Control** from the Help menu. Notice that the message box has the VIEW2 command button that you added to the ACVACQ message earlier.

Now click on the VIEW2 command button. The MOREFUNC message will be displayed, as shown in the following illustration. Select **OK** to remove the message.

To continue, exit IPG and maximize the Help Editor window.



Deleting a Message

To delete a message, you first need to determine several things:

- That the ID is not tied to the Script name or to a control variable name. If it is, you may have to change the script along with the *.HLP file.
- The ID of the message you want to delete
- The ID of the message that has the link(s) to the message you want to delete
- Whether the message you want to delete has any links to other messages; if it has, these will also need to be deleted, so you will need to know their **IDs** as well.

In this tutorial, we will delete the message `MOREFUNC` that you added in the previous tutorial. We already know the following:

- That the message `MOREFUNC` is not an original message tied to a control variable name, so no changes to the script are necessary
- That the **ID** of the message is `MOREFUNC`
- That the ID of the message having the link to the message that will be deleted is `ACVACQ`

- That the message to be deleted has no links to any other messages.

If we were deleting some other message, we could determine some of these items by examining the links of the message to be deleted; however, it might be difficult to determine all of the links, and to determine which message has the link to the message we want to delete. In this event, you could print the *.HLP file and then find the link to the file you want to delete in the printout. The printout lists the **ID**, **Caption**, **Text:**, command button labels, and destinations for all messages in the file. (To print a *.HLP file, select the file, and then select **print** in the File menu.)

We will now actually delete the message `MOREFUNC`. Select the message from the `TDM5120.HLP ID` list, so that the **ID** and **Text:** are displayed. Select **Delete**, and the **ID** and **Text:** will disappear, and the **ID** will be removed from the Listbox.

Next, we will delete the link to the message just deleted. Select the message `ACVACQ` that has the link to message `MOREFUNC` that has been deleted. Select **Links...** Highlight the Label-Destination line `VIEW2 | MOREFUNC`, then select **Delete**.

Both the message `MOREFUNC` and the link from message `ACVACQ` to message `MOREFUNC` have now been deleted. Select **OK** to remove the Edit Links box.

Ending an Edit Session

When you are finished editing, you need to save the changes to the *.HLP file. To save the changes, select **File**; then select **Save**.

At this point, you can open another file for edit, if you wish, or you can exit the Help Editor.

Making a New ISD Help Message File

This information describes how to create a new help message file for an instrument soft front panel in IPG, add messages to it, and link the messages. For information on how to call these messages from the script, refer to the Interactive Procedure Generator (IPG) Users Manual.

Before creating a new message file, decide what messages you want in the file and the relationship between messages where you will have linked messages. Two forms are shown at the end of this section; one form provides a way to record the relationship between messages, and information about a message can be recorded on the other form. Copy these forms and use the copies to plan and then create the new message file.

Modify an Existing Help File — There are two ways to create a new help message file. One way is to open an existing *.HLP file, save it to another file name, and then modify the file that has the new name. This method works well when an existing help file is similar to the help file you want to create.

Start A New Help File — The other way to create a new help message file is to select **New** in the File menu. Fill in the ID and **Text:** box with information for the first message, and select Add to place the message ID in the Listbox. You can add more messages and add links between messages as previously described; however, at some point you need to select Save to save the file to a new file name. The new file name must be the same as the script (i.e., if the script is TFG5010.ISD, the file from the Help Editor must be TFG5010.HLP).

NOTE

As a reminder, you cannot give an *.ISD file and *.HLP file the same names as a name in your GPIB.COM /BCONF file.

The following illustration shows part of the preceding message link map filled in. The first **ID** on the left (column **1A**) is the **ID** of a message that is linked to the new message (in column **4B**).

	1	2	3	4	5	6	7	8	9	0	11	12
	ID	Label	Dst	ID	Label	Dst	ID	Label	Dst	ID	Label	Dst
A	301	OK	-1									
B		Help	401	401	Label	501	501	OK	-1			
C					Text	502	502	-1	-1			
D												
E	302	OK	-1									
F		Help	402									
G												
H												

Message Form

ID _____

Link from Message ID _____ to this message.

Links from this message to other messages.

Label: _____ Linked to: _____

Label: _____ Linked to: _____

Label: _____ Linked to: _____

Label: _____ Linked to: _____

Caption:

Text:

Printing a *.HLP File

The Help Editor can be used to print all message information contained in a .HLP file. The printout lists the ID, Caption, **Text**, command button labels, links, and destinations for each message in the file. To print a *.HLP file, run the Help Editor, select the *.HLP file that you want to print, and then select **Print** in the File menu.

A printout will look similar to the sample portion illustrated below.

It might be helpful, before you begin using the Help Editor to modify a *.HLP file, to print the file you are going to change. Changes can be marked on the printout as you plan what changes to make, then the printout can be used for reference during the edit session.

```
ID: ACQUIRE
    ACQ
```

```
THIS IS A PUSHBUTTON CONTROL, AN INPUT ONLY DEVICE. THE
USER CLICKS THE BUTTON CAUSING AN ACTION TO OCCUR. THE
CONTROL WILL MOMENTARILY DARKEN AND THEN REVERT TO ITS
NORMAL STATE.
```

```
THIS PARTICULAR CONTROL WILL CAUSE THE INSTRUMENT TO GO
TO ACV MODE AND THEN UPDATE MEASUREMENT DISPLAY.
```

```
DATATYPE: STRING
```

```
INPUT: 1 - PUSH
```

```
BUTTONS:
```

```
    OK          -1
```

```
ID: ACVRANGE
```

```
    RANGE
```

```
THIS CONTROL IS A PUSHBUTTON CONTROL. THE USER DOUBLE-
CLICKS
```

```
THE SELECTION THAT IS DESIRED.
```

```
DATATYPE: STRING
```

```
INPUT/RESPONSES: "AUTO" (AUTORANGE)
```

```
                  "1" (300MV RANGE)
```

```
                  "2" (3V RANGE)
```

```
                  "3" (30V RANGE)
```

```
                  "4" (300V RANGE)
```

```
BUTTONS:
```

```
    OK          -1
```

Appendices

Appendix A

TDM5120.ISD Script Listing

A listing of TDM5120.ISD is provided here so you can examine the entire script while developing new scripts or modifying scripts.

```
REM  TEKTRONIX DM5120 DIGITAL MULTIMETER SCRIPT "DM5120"

    GPIB
    END

LEARN VAR:STR
    SETTING
        CONT -> VAR;
    END
    MEASUREMENT
        CONT -> "SET?";
        INST -> VAR;
    END
END

VIEW "ACV"

    TEXT "DM5120 AC VOLTMETER" @ 5,1.0;
    TEXT "ACCURACY" @ 3.75,6.5;
    CONNECT(12.5,7),(14,7),(14,14),(2,14),(2,7),(3,7);

    CONTROLGROUP
        CONTROL ACVACQ:INT PUSHBUTTON @ 18.75,15

        REMARK .ACQ FORACQUIRE

        STRING "ACQ";
        UPDATELIST ACVREADING;
    END
    SETTING
        CONT -> "ACV";
    END
END

    CONTROLGROUP
        CONTROL ACVREADING:FLOAT TEXTBOX @ 2.5,4.5
        NUMROWS 1; NUMCOLS 20;
        CONTROLTITLE "MEASUREMENT";
    END
END
```

Appendix A: TDM5120.ISD Script Printout

```
MEASUREMENT
  CONT -> "SEND";
  INST-> ACVREADING, ";";
END
END

CONTROLGROUP
CONTROLACVTHREE:INT RADIOBUTTON @ 2.5,7.5
  STRING "3 DIGITS";
  ONEOFGROUP "ACCURACY";
  UPDATELIST ACVREADING;
END
SET
  IF (ACVTHREE==1) THEN
    CONT -> "DIGIT 3";
  ENDIF
END
QUERY
  TEMPVAR TMP:STR;
  ACVTHREE =0;
  CONT -> "DIGIT?";
  INST -> "DIGIT ", TMP, ";";
  IF (TMP=="3") THEN
    ACVTHREE = 1;
  ENDIF
END
END

CONTROLGROUP
CONTROL ACVFOUR:INT RADIOBUTTON @ 2.5,9
  STRING "4 DIGITS";
  ONEOFGROUP "ACCURACY";
  UPDATELISTACVREADING;
END
SETTING
  IF (ACVFOUR==1) THEN
    CONT -> "DIGIT 4";
  ENDIF
END
MEASUREMENT
  TEMPVAR TMP:STR;
  ACVFOUR = 0;
  CONT -> "DIGIT?";
  INST -> "DIGIT ", TMP, ";";
  IF (TMP=="4") THEN
    ACVFOUR =1;
  ENDIF
END
END
```

```

CONTROLGROUP
CONTROLACVFIVE:INT RADIOBUTTON @ 2.5,10.5
  STRING "5 DIGITS";
  ONEOFGROUP "ACCURACY";
  UPDATERELIST ACVREADING;
END
SETTING
  IF (ACVFIVE==1) THEN
    CONT -> "DIGIT 5";
  ENDIF
END
MEASUREMENT
  TEMPVAR TMP:STR;
  ACVFIVE = 0;
  CONT -> "DIGIT?";
  INST -> "DIGIT ", TMP, ";";
  IF (TMP=="5") THEN
    ACVFIVE = 1;
  ENDIF
END
END

```

```

CONTROLGROUP
CONTROL ACVSIX:INT RADIOBUTTON @2.5,12.
  STRING "6 DIGITS";
  ONEOFGROUP "ACCURACY";
  UPDATERELIST ACVREADING;
END
SETTING
  IF (ACVSIX==1) THEN
    CONT -> "DIGIT 6";
  ENDIF
END
MEASUREMENT
  TEMPVAR TMP:STR;
  ACVSIX = 0;
  CONT ->"DIGIT?";
  INST -> "DIGIT ", TMP, ";";
  IF (TMP=="6") THEN
    ACVSIX = 1;
  ENDIF
END
END

```

Appendix A: TDM5120.ISD Script Printout

```
CONTROLGROUP
CONTROL ACVFILT:INT CHECKBOX @ 2.5,15.0
  STRING "AVE";
  UPDATERLIST ACVREADING;
END
CONTROLACVFILTVAL:INT EDIT @ 8.75,15.0
  NUMROWS 1;NUMCOLS 5;
END

SETTING
  IF (ACVFILT==1) THEN
    CONT -> "FILTERVAL ",ACVFILTVAL,";FILTER ON";
  ELSE
    CONT -> "FILTER OFF";
  ENDIF
END
MEASUREMENT
  TEMPVAR FILTON:STR;
  CONT ->"FILTER?;FILTERVAL?";
  INST -> " ",FILTON,";"," ",ACVFILTVAL,";";
  IF (FILTON=="ON") THEN
    ACVFILT=1;
  ELSE
    ACVFILT=0;
  ENDIF
END
END

CONTROLGROUP
CONTROL ACVRANGE:STR LISTBOX @ 18.75,8.0
  NUMROWS 5;NUMCOLS 5;
  CONTROLTITLE "RANGE";
  STRING " AUTO","300 MV", "3 V"," 30 V","300 V";
  TOSCRIPT "AUTO","1", "2","3","4";
  UPDATERLIST ACVREADING;
END
SETTING
  CONT -> "RANGE ",ACVRANGE;
END
MEASUREMENT
  CONT -> "RANGE?";
  INST -> " ",ACVRANGE,";";
END
END
END
```



```

VIEW "DCV"

TEXT "DM5120 DC VOLTMETER" @ 5,1.0;
TEXT "ACCURACY" @ 3.75,6.5;
CONNECT (12.5,7),(14,7),(14,14),(2,14),(2,7),(3,7);

CONTROLGROUP
  CONTROL DCVACQUIRE:FLOAT PUSHBUTTON @ 18.75,15
    STRING "ACQ";
    UPDATELIST DCVREADING;
  END

SETTING
  CONT -> "DCV";
END

END

CONTROLGROUP
  CONTROL DCVREADING:FLOAT TEXTBOX @ 2.5,4.5
    NUMROWS 1; NUMCOLS 20;
    CONTROLTITLE "MEASUREMENT";
  END
  MEASUREMENT
    CONT -> "SEND";
    INST -> DCVREADING, ",";
  END
END

CONTROLGROUP
  CONTROL DCVTHREE:INT RADIOBUTTON @ 2.5,7.5
    STRING "3 DIGITS";
    ONEOFGROUP "ACCURACY";
    UPDATELIST DCVREADING;
  END
  SET
    IF (DCVTHREE) THEN
      CONT -> "DIGIT 3";
    ENDIF
  END
  QUERY
    TEMPVARTMP:STR;
    DCVTHREE = 0;
    CONT -> "DIGIT?";
    INST-> "DIGIT ", TMP, ",";
    IF (TMP=="3") THEN
      DCVTHREE =1;
    ENDIF
  END
END

```

Appendix A: TDM5120.ISD Script Printout

```
CONTROLGROUP
  CONTROLDCVFOUR:INT RADIOBUTTON @ 2.5,9
  STRING "4 DIGITS";
  ONEOFGROUP "ACCURACY";
  UPDATERLIST DCVREADING;
END
SETTING
  IF (DCVFOUR) THEN
    CONT -> "DIGIT 4";
  ENDIF
END

MEASUREMENT
  TEMPVAR TMP:STR;
  DCVFOUR = 0;
  CONT -> "DI .GIT?";
  INST -> "DIGIT ", TMP, "i";
  IF (TMP=="4") THEN
    DCVFOUR = 1;
  ENDIF
END
END

CONTROLGROUP
  CONTROL DCVFIVE:INTRADIOBUTTON @ 2.5,10.5
  STRING "5 DIGITS";
  ONEOFGROUP "ACCURACY";
  UPDATERLIST DCVREADING;
END
SETTING
  IF (DCVFIVE) THEN
    CONT -> "DIGIT 5";
  ENDIF
END
MEASUREMENT
  TEMPVAR TMP:STR;
  DCVFIVE = 0;
  CONT -> "DIGIT?";
  INST -> "DIGIT ", TMP, "i";
  IF (TMP=="5") THEN
    DCVFIVE = 1;
  ENDIF
END
END
```

```

CONTROLGROUP
CONTROL DCVSIX:INT RADIOBUTTON @ 2.5,12.
  STRING "6 DIGITS";
  ONEOFGROUP "ACCURACY";
  UPDATELIST DCVREADING;
END
SETTING
  IF (DCVSIX) THEN
    CONT -> "DIGIT 6 ";
  ENDIF
END
MEASUREMENT
  TEMPVAR TMP:STR;
  DCVSIX = 0;
  CONT -> "DIGIT?";
  INST -> "DIGIT ", TMP, ";";
  IF (TMP=="6") THEN
    DCVSIX =1;
  ENDIF
END
END

CONTROLGROUP
CONTROL DCVFILT:INT CHECKBOX @ 2.5,15.0
  STRING "AVE";
  UPDATELIST DCVREADING;
END
CONTROL DCVFILTVAL:INT EDIT @ 8.75,15.0
  NUMROWS 1;NUMCOLS 5;
END
SETTING
  IF (DCVFILT==1) THEN
    CONT -> "FILTERVAL ",DCVFILTVAL,";FILTERON";
  ELSE
    CONT -> "FILTER OFF";
  ENDIF
END
MEASUREMENT
  TEMPVAR FILTON:STR;
  CONT ->"FILTER?;FILTERVAL?";
  INST -> " ",FILTON,";"," ",DCVFILTVAL,";";
  IF (FILTON=="ON") THEN
    DCVFILT=1;
  ELSE
    DCVFILT=0;
  ENDIF
END
END

```

Appendix A: TDM5120.ISD Script Printout

```
CONTROLGROUP
CONTROL DCVRANGE:STR LISTBOX @ 18.75,8.0
  NUMROWS 5;NUMCOLS 5;
  CONTROLTITLE "RANGE";
  STRING " AUTO","300 MV", "3 V", " 30 V","300 V";
  TOSCRIPT "AUTO","1", "2","3","4";
  UPDATELIST DCVREADING;
END
SETTING
  CONT -> "RANGE ",DCVRANGE;
END
MEASUREMENT
  CONT -> "RANGE?";
  INST -> " ",DCVRANGE,"";
END
END
END
END
```

Appendix B: Software Performance Report

This Software Performance Report is for your use in reporting any problems you experience when using TekTMS/FPE. It provides us with a way to track problems with a particular system and it ensures that we provide you and other customers with a prompt solution to the problem. Please supply the following information:

Customer (Company Name): _____

User (Person making report): _____

Address: _____

City: _____ State: _____ Zip: _____

Country: _____ Telephone: _____

System Description (Product Name, S/W Version, Serial Number): _____

Product Name: _____ PC Model/Make: _____

Version: _____ DOS Version: _____

Serial No: _____ Windows Version: _____

Other Software (TSR's, ...)

Problem Description: _____

(If possible, include exact steps to recreate the problem.)

Itemize attached documentation (i.e., listings, diskettes, ...)

Mail the report to:

Tektronix, Inc.
Instrument Controllers and Software Marketing Dept.
MS 47-665
P.O. Box 500
Beaverton, OR 97075-9965

Or FAX to:

(503) 627-2933

Marked for ICS Marketing, 47-665

Appendix B: Software Performance Report

Index

Symbols

- &, 2-43
 - Precedence, 2-44
- %, 2-51
- %F Format, Definition, 2-51, 2-54
- %n.mE Format, Definition, 2-51, 2-54
- %n.mG Format, Definition, 2-51, 2-54
- %nB Format, Definition, 2-51, 2-54
- %nD Format, Definition, 2-51, 2-54
- %nH Format, Definition, 2-51, 2-54
- %nO Format, Definition, 2-51, 2-54
- %nS Format, Definition, 2-51, 2-54
- +, 2-43
 - Precedence, 2-44
- † Formatting Modifier, Definition, 2-52
- , 2-43
 - Precedence, 2-44
- *, 2-43
 - As a string entry, 2-50
 - Precedence, 2-44
- /, 2-43
 - Precedence, 2-44
- =, 2-43
 - Precedence, 2-44
- ==, 2-43
 - Precedence, 2-44
- <, 2-43
 - Precedence, 2-44
- < Formatting Modifier, Definition, 2-52
- <=, 2-43
 - Precedence, 2-44
- <>, 2-43
 - Precedence, 2-44
- >, 2-43
 - Precedence, 2-44
- >=, 2-43
 - Precedence, 2-44
- \, As a string entry, 2-50

A

- Activating Controls, 3-28
- Actual Versus Composite Controls, 1-4
- addition, Precedence, 2-44
- and Operator, 2-43
 - Precedence, 2-44
- Application Oriented Front Panel Views, 1-1
- ASCII, Using, 2-8
- ASCII Character Conversion Function, Definition, 2-45
- ASCII Learn Block, 2-9
- assignment Operator, 2-43
 - Precedence, 2-44
- Assignment Statement
 - Definition, 2-35, 2-42
 - Examples, 2-43
- Assignment Statements, 2-40
- ATN Command, Definition, 2-41

B

- band Operator, 2-43
 - Precedence, 2-44
- BINARY, Using, 2-8
- Binary, Definition, 2-51, 2-54
- Binary Format, Using, 2-52, 2-54
- Binary Icon, for translation, 3-31
- Binary ISD Files, 3-31–3-34
 - Using binary, 3-31
 - Using Binary ISD Files, 3-33
- BINARY Learn Block, 2-9
- bitwise AND Operator, 2-43
 - Precedence, 2-44
- bitwise EXCLUSIVE OR Operator, 2-43
 - Precedence, 2-44

- bitwise negation, Precedence, 2-44
- bitwise NOT Operator, 2-43
 - Precedence, 2-44
- bitwise OR Operator, 2-43
 - Precedence, 2-44
- bnot Operator, 2-43
 - Precedence, 2-44
- bor Operator, 2-43
 - Precedence, 2-44
- BusNote, Keyword listing, 2-58
- BusNote Block
 - CDS 53 Series Parameters, 2-7
 - Definition, 2-4
 - GPIB Parameters, 2-5
 - MXI Parameters, 2-7
 - RS232 Parameters, 2-6
 - VX5520 Parameters, 2-6
 - VXI Parameters, 2-7
- bxor Operator, 2-43
 - Precedence, 2-44

C

- CDS 53 Series Parameters
 - Default settings, 2-8
 - Description, 2-7–2-10
- CDSBUS, Definition, 2-4
- Charting a Front Panel, 3-3
- CheckBox, 3-19
 - Activating, 3-28
 - Data Types, 2-15
 - Definition, 2-21–2-23
 - Using, 3-5, 3-19
- CheckBox with EditBox, Activating, 3-28
- chr(n) Function
 - Definition, 2-45
 - Using, 2-37
- Common Problems when Formatting, 2-55

- Condition, Definition, 2-36
- CongrolGroup Block, Using, 3-8
- Connect Statement
 - Definition, 2-11-2-13
 - Example, 2-12
- Connect Statements, Using, 3-8
- Construct Definitions, Listing, 2-15
- cont ->, 2-37
- Control Block
 - Definition, 2-13
 - Description, 2-15
 - Keyword listing, 2-58
 - Using, 3-9
- Control ID, Using, 2-13, 2-14
- Control Sizes (in character spaces), Listing, 3-3
- Control Title, Using, 3-9
- Control Type Summary Chart, 2-33
- ControlGroup Block
 - Definition, 2-12-2-34
 - Description, 2-15
 - Keyword listing, 2-58
 - Using, 3-24
- Controller Dialog
 - Description, 2-37
 - Using, 3-11, 3-17
- Controller Dialog Format, 2-53
- ControllerDialogFormats, Description, 2-51
- ControlTitle, Definition, 2-15
- Converting ASCII to Binary, 3-31
- Converting waveforms to variables, 2-26-2-30
- Coordinate System, 2-11
- CRLF (ReturnLine Feed), As a string entry, 2-50

D

- Data Values, For Controls, 2-15
- DCL Command, Definition, 2-41
- DefaultPos, Definition, 2-15
- Determining Control Types, 3-3
- Device Name, 3-26

- Dialogs
 - Listing, 2-35
 - Using, 2-37-2-40
- Display Coordinates, Explained, 3-3
- Display String Function
 - Definition, 2-46
 - Example, 2-46
- Display Types, 2-10
- divide Operator, 2-43
- division, Precedence, 2-44

E

- EditBox
 - Data Types, 2-15
 - Definition, 2-18-2-19
 - Using, 3-19, 3-20
- EOI (End Of Input) Parameter, GPIB definition, 2-5
- EOM (End Of Message) Parameter, GPIB definition, 2-5
- equal Operator, 2-43
 - Precedence, 2-44
- Error Messages, 3-28
- Errors
 - NonNumeric character, 2-56
 - Not Enough Data Received, 2-55
 - Too Much Data Received, 2-56
- Exponential Numbers, Definition, 2-51, 2-54
- Expressions
 - Using in a Controller Dialog, 2-37
 - Using in an Instrument Dialog, 2-38

F

- FastDCRead Function
 - Definition, 2-48-2-49
 - Example, 2-48
 - Using, 2-47
- FastDCWrite Function
 - Definition, 2-46
 - Example, 2-46
 - Using, 2-47

- file name, Using in %F format, 2-51, 2-54
- floating point, Definition, 2-51, 2-54
- Floats, Definition, 2-50
- Format Character, 2-51, 2-58
- Format Types
 - Free Format, 2-52
 - Leading zero fill, 2-52
 - Left justify, 2-52
 - Listing, 2-51
 - Output sign, 2-52
- Formats, Common Problems, 2-55
- Free Format, 2-2, 2-52
- Front Panel Appearance, 1-5
- Front Panel Helps, 1-5
- Front Panel View Optimization, 1-3
- Front Panel Views, 1-1
- Full Interaction Mode, Example, 2-13
- Fully Functional Instrument Front Panel Views, 1-1
- Function Call, Definition, 2-35
- Functions
 - CHR, 2-45
 - Descriptions, 2-45-2-49
 - Display, 2-46
 - FastDCRead, 2-48-2-49
 - FastDCWrite, 2-46-2-47
 - Keyword listing, 2-58
 - Readlength, 2-49-2-50
 - TimeDelay, 2-45

G

- Generating Test Procedures, 3-29
- GET Command, Definition, 2-41
- Global Variable, 2-13
- GPIB, Definition, 2-4
- GPIB Bus, Using, 3-7
- GPIB Command, Keyword listing, 2-58
- GPIB Commands, 2-40
 - Definition, 2-35
 - Where Used, 2-40

GPIB Parameters

- Default settings, 2-6
- Description, 2-5-2-6
- End of Input (EOI), 2-5-2-6
- End of Message (EOM), 2-5-2-6
- Timeout, 2-5-2-6

greater than Operator, 2-43

- Precedence, 2-44

greater than or equal Operator, 2-43

- Precedence, 2-44

Grouping RadioButtons, 3-15

GTL Command, Definition, 2-41

H

Help Editor

- Adding a Message, 4-9
- Captions, 4-4
- Command Button Labels, 4-7
- Creating a New Message, 4-12
- Deleting Messages, 4-14
- Description, 4-1
- Destination, 4-5
- Displaying Command Buttons, 4-8
- Displaying Message Links, 4-8
- Edit Links Box, 4-8
- Editing Help Messages, 4-5
- Ending, 4-15
- Help Editor & Script Internal Relationships, 4-2
- Help Message Files, 4-2
- Help Message Parts, 4-3
- Labels, 4-4
- Learning to Use, 4-1
- Linking Messages, 4-10, 4-13
- Message Form, 4-19
- Message ID, 4-4
- Message Link Map Form, 4-17
- Message Text, 4-4
- Printing a .HLP File, 4-20
- Starting, 4-2
- Testing Messages, 4-13

Hexadecimal, Definition, 2-51, 2-54

Hexadecimal Format, Using, 2-52, 2-54

HT (Horizontal Tab), As a string entry, 2-50

IEEE 488, 2-40

IF Conditional Structure, Definition, 2-36

IF Statement, 2-44

- Definition, 2-36
- Using, 3-16, 3-18, 3-23

IF Statements, 2-40

If the endif Statement, Definition, 2-35

If then else endif Statement, Definition, 2-35

IFC Command, Definition, 2-41

input-expression, Definition, 2-38

inst ->, 2-37

INST Dialog, Readlength function, 2-49-2-50

Instrument Dialog

- Description, 2-38
- Using, 3-17

Instrument Dialog Format, 2-55

Instrument Dialog Formats, Description, 2-54

Instrument Front Panel, 1-1

Instrument Measurement Update Path, 3-13

Instrument Script Language (ISL), 3-1

Instrument Select Menu Item, 2-36

Instruments Menu Item, 3-27

Integer, Definition, 2-54

integer, Definition, 2-51

Integer Values, For Controls, 2-15

Integers, Definition, 2-50

Interactive Front Panel Views, 1-1

Interactive Views, 1-2

ISD Files

- ASCII, 3-31
- binary, 3-31
- Using binary, 3-31
- Using binary ISD files, 3-33

ISL Keywords, Listing, 2-57-2-60

K

Keywords, Listing, 2-57-2-60

L

Learn Block

- Definition, 2-8-2-9
- Keyword listing, 2-58
- with Measurement block, 2-8
- with Setting block, 2-8

Learn Setting...Menu Item, 2-9

LEARNED SETTING Step, 2-8

less than Operator, 2-43

- Precedence, 2-44

less than or equal Operator, 2-43, 2-43

- Precedence, 2-44

LF (Line Feed), As a string entry, 2-50

ListBox

- Activating, 3-28
- Data Types, 2-15
- Definition, 2-19-2-20
- Using, 3-3, 3-24

Literal Strings, 2-50

LLO Command, Definition, 2-41

logical AND Operator, 2-45

- Precedence, 2-44

Logical Instrument window, 1-1

logical negation, Precedence, 2-44

logical NOT Operator, 2-43

logical OR Operator, 2-43

- Precedence, 2-44

M

Many-To-One Control Groups, 3-19

Measurement Block

- Definition, 2-35
- in a Learn Block, 2-8

Keyword listing, 2-59
Using, 3-12

Measurement Block Variables, Using, 3-17

Measurement Block with two Variables, 3-22

minus Operator, 2-43

Miscellaneous Functions, Keyword listing, 2-58

Modifiers, 2-51

Modifying Scripts, 3-29

Multiple Views, 2-10

multiplication, Precedence, 2-44

multiply Operator, 2-43

MXI, Definition, 2-4

MXI Bus Parameters
Default settings, 2-7
Description, 2-7

N

NonNumeric Character, As the cause of an error, 2-56

NOOP (no operation), 2-40

Not Enough Data Received Error, 2-55

not equal Operator, 2-43
Precedence, 2-44

not Operator, 2-43
precedence, 2-44

Numcols, Definition, 2-15

Numeric Format Handling, Possible problem conditions, 2-56

NumRows, Definition, 2-16

O

Octal, Definition, 2-51, 2-54

Octal Format, Using, 2-52, 2-54

OneOfGroup, Definition, 2-16

OneOfGroup Statement, Using, 3-15

Operators, Listing, 2-42, 2-59

or Operator, 2-43
Precedence, 2-44

output-expression, Definition, 2-37
Overview, 1-1

P

Parentheses. Using, 2-34

plus Operator, 2-43

Precedence for Evaluation, Parentheses, 2-44

Precision, 2-51

Precision Modifiers, Using, 2-55

Primary Address. Using, 3-27

Program Generation Front Panel Views, 1-1

Program Generation Views, 1-2

PushButton
Activating, 3-28
Data Types, 2-15
Definition, 2-20-2-21
Using, 3-5, 3-8

R

RadioButton
Activating, 3-28
Data Types, 2-15
Definition, 2-23-2-25
Using, 3-5, 3-14

Readlength Function, Definition, 2-49-2-50

Reloading Process, 3-28

Remark Keyword, Using, 3-6

REN Command, Definition, 2-41

Reserved Words, Listing, 2-57-2-60

RS232, 2-40
Definition, 2-4

RS232 Bus, Using, 3-7

RS232 Parameters
Default Settings, 2-6
Description, 2-6

S

Save As... Menu Item, 3-26

Script and Description Presentation, 3-6

Script Basics, 2-1-2-4

Script Block, Definition, 2-4-2-5

Script Block Keywords, Listing, 2-59

Script Development Procedures, 3-6

Script Identifier, 2-4

Script Interchangeability, 3-5

Script Keyword, Using, 3-7

Script Model, 3-2

Script Planning, 3-3

Scripts, *i*

ScrollHorz, Definition, 2-16

ScrollVert, Definition, 2-16

SDC Command, Definition, 2-41

Select... Menu Item, 3-27

Setting Block
Definition, 2-35
in a Learn Block, 2-8
Keyword listing, 2-59
Using, 3-10

Setting Block with Two Variables, 3-21

Setting, Learn, & Measurement Ctrl Functions, 1-3

Skip Characters Function, Description, 2-39

skip(n) Function
Definition, 2-39
Example, 2-38
Using in An Instrument Dialog, 2-39

State Construct, Definition, 2-16

Statements
Description, 2-40-2-45
Listing, 2-35-2-36

Step Menu Item, 2-9

string, Using, 2-51, 2-54

String concatenation, Precedence, 2-44

string concatenation Operator, 2-43

String Constants
 Using in a Controller Dialog, 2-37
 Using in an Instrument Dialog, 2-38

String Construct, Definition, 2-16

String Statement, Using, 3-24

Strings
 Definition, 2-50
 Literal, 2-50

subtraction, Precedence, 2-44

T

Tab, As a string entry, 2-50

TALK/LISTEN Function, 2-4

Talking to an Instrument, 3-13

TDM5120.ISD Printout, A-1

Temporary Variables
 Declaration, 2-40
 Definition, 2-40

Tempvar Statement, Definition, 2-40

Testing Scripts, 3-27

Text Statement
 Definition, 2-11–2-13
 Example, 2-11

Text Statements, Using, 3-8

TextBox
 Data Types, 2-15
 Definition, 2-17–2-18
 Using, 3-12

TIM Command, Definition, 2-41

TimeDelay Function, Definition, 2-45

Timedelay Function, Example, 2-45

Timeout Parameter, GPIB definition, 2-5

Title Construct, Definition, 2-16

TMS Name, Using, 3-27

Too Much Data Received Error, 2-56

ToScript, Definition, 2-16

ToScript Statement, Using in a ListBox, 3-24, 3-25

ToScriptOff, Definition, 2-16

ToScriptOn, Definition, 2-16

Translate Button, ASCII to Binary, 3-31

U

UNL Command, Definition, 2-41

UNT Command, Definition, 2-41

Update Menu Item, 1-5

Update!, 1-5

UpdateList
 Definition, 2-17
 Using, 2-14, 3-10

UpdateList Statement, Using, 3-15

Updating Features, 1-5

User Requirements, 1-1

V

Variable Declarations, Definition, 2-35

Variable Formats
 Controller Dialog variables, 2-51–2-60
 Description, 2-51–2-56
 Instrument Dialog variables, 2-54–2-60

Variable Types
 Description, 2-50
 Float, 2-50
 Integer, 2-50
 String, 2-50
 Waveform, 2-50
 WaveformToVar definition, 2-26–2-30

Variable:%format
 Using in a Controller Dialog, 2-37
 Using in a Instrument Dialog, 2-38

View Block
 Definition, 2-9–2-12
 Keyword listing, 2-59

View Identifier, 2-9

View Keyword, Using, 3-7

View Layout, 2-10–2-11

VX5520, 2-40
 Definition, 2-4

VXI Bus Parameters
 Default Settings, 2-6–2-7
 Description, 2-6–2-7

VXI Bus, Using, 3-7

VXI Fast Data Channel protocol, 2-46, 2-48–2-49

VXI Internal Bus Parameters
 Default settings, 2-7
 Description, 2-7

VXIINTERNAL, 2-40
 Definition, 2-4

W

Waveform Display
 Data Types, 2-15
 Definition, 2-25–2-33
 WaveformToADIF, 2-25
 WaveformToVar, 2-25

WaveformDisplay, Example, 2-31

WaveformToADIF, waveform display variable, 2-25

WaveformToAdif, Definition, 2-30

WaveformToVar
 Definition, 2-26–2-30
 variable use, 2-50
 waveform display variable, 2-25

While do end Statement, Definition, 2-35

WHILE Statement, 2-45

WHILE Statements, 2-40

Width, 2-51

Width Modifiers. Using, 2-55

Windows Notepad, 3-1

Z

Z Formatting Modifier, Definition, 2-52

