HP Interactive Test Generator for MS-DOS^R

HP ITG User's Handbook

HEWLETT PACKARD

HP Part No. E2020-90000
Printed in USA

Hewlett-Packard Interactive Test Generator and Microsoft Windows

HP Interactive Test Generator (HP ITG) operates in a graphics environment called Microsoft Windows, created by Microsoft Corporation. An extension of the MS-DOS operating system, Microsoft Windows gives a standard look and feel to HP ITG and all other Windows applications.

The HP Interactive Test Generator package contains all the software necessary to run HP ITG. You can also run HP ITG under Microsoft Windows version 2.X.

With Microsoft Windows, you can take advantage of these additional features of the Windows environment:

- Running multiple applications: You can run several applications under Windows at one time and easily switch between them, creating an integrated work environment.
- Data exchange between applications: You can transfer data between HP ITG and other standard DOS applications as well as other Windows applications.
- Windows control of the DOS environment: From the Windows environment you can easily access all Windows and non-Windows applications, files, directories, and disks, and control all DOS-related tasks such as directory or file management and formatting disks.

To run HP Interactive Test Generator under Microsoft Windows, you need to license and install Microsoft Windows version 2.X.

Printing History

First Edition - December 1989

Contents

Figures

Tables

1

About HP ITG

Overview

Welcome to the HP Interactive Test Generator for MS-DOS.

Although the HP Interactive Test Generator (HP ITG) is an easy-to-use system for developing test programs, you may have to turn to this handbook from time to time. It can help you get started quickly and is designed to answer any questions you may have while using HP ITG.

In addition, HP ITG provides an online Help system that serves as a quick reference guide. It also includes a tutorial that demonstrates proper use of HP ITG.

A Software Development Tool

This version of the HP Interactive Test Generator is a PC-based application. It simplifies test-software development by allowing you to generate instrument-control code without knowing the programming languages of the instruments in the test system.

HP ITG provides two environments in which to work:

- **Development environment:**

  Using instrument panels displayed in HP ITG, you can change individual controls as well as an instrument's entire configuration, and generate instrument-control code for your test program. Using the programming environments for languages supported by HP ITG, you can enhance and debug your program.

- **Run-time environment:**

  When the test programs you develop are done, you can create standalone executables that you run on instrument controllers attached to automated test systems.

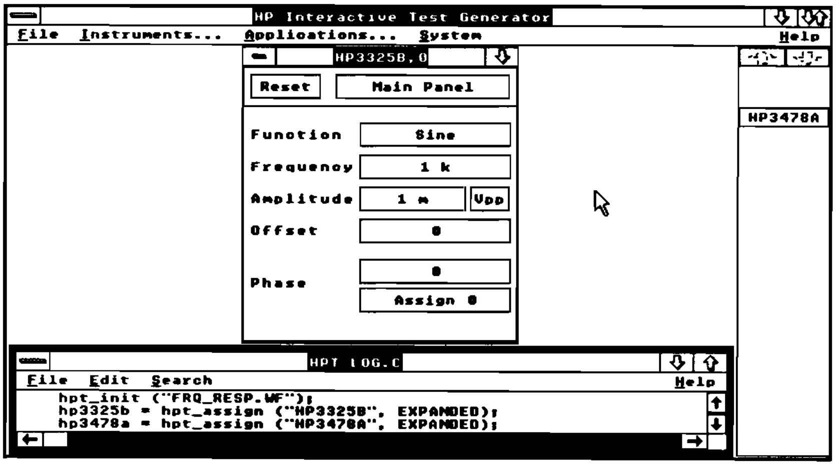
Panels and Drivers

HP ITG uses instrument-specific information stored in a **driver.** An instrument driver is a text file that describes the layout of the **soft panel** for the instrument and its HP-IB commands. The soft panel appears in the HP ITG **work area** and lets you interact with an instrument.

You must use a driver for each instrument you want to control with HP ITG. Drivers for many HP instruments are supplied with HP ITG. If you have a special driver need, the companion manual, *How to Write an HP ITG Driver,* explains how to develop drivers for other instruments.

Development Environment

HP ITG provides a flexible program development environment. When you install HP ITG, you can choose from the supported programming languages. Then HP ITG generates the instrument-control code in the language you selected. The generated code is easily loaded into the Microsoft^{R} QuickC^{R} or Microsoft QuickBASIC environments to complete your program development.



```
    hpt_init ("FRQ_RESP.WF");
    hp3325b = hpt_assign ("HP3325B", EXPANDED);
    hp3478a = hpt_assign ("HP3478A", EXPANDED);
```

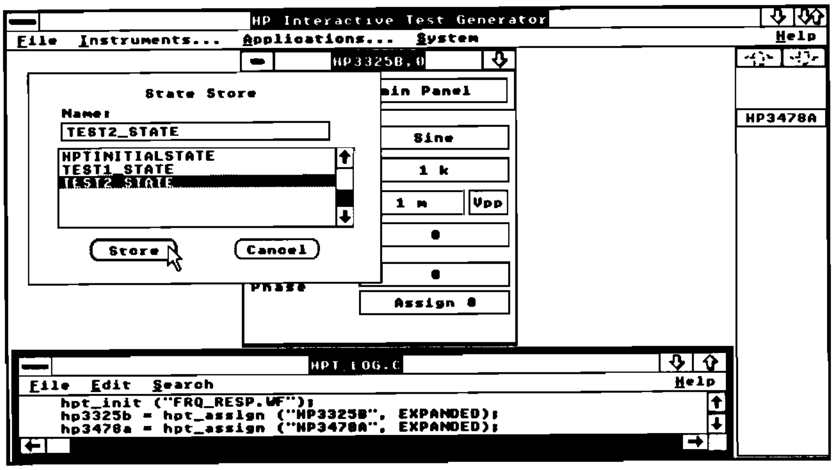
Figure 1-1. The HP ITG Development Environment

Creating Measurement Procedures

Once HP ITG is set, you can begin to create your **measurement procedure.** A procedure includes the code that recalls instrument states, adjusts instrument controls, and makes measurements. You can generate instrument-control code interactively within HP ITG, even if instruments are not available to connect to the HP-IB bus.

Saving Instrument States

To avoid setting controls individually each time you use an instrument, you can store and recall complete instrument states. The combination of instrument panels and states is called a **soft test system.** After creating the soft test system, you can save it as a **workfile** for use in the Quick and run-time environments or return to a particular soft test system when re-entering HP ITG's development environment.



```
hpt_init ("FRQ_RESP.WF");
hp3325b = hpt_assign ("HP3325B", EXPANDED);
hp3478a = hpt_assign ("HP3478A", EXPANDED);
```

Figure 1-2. Store Instrument Setups as States for Use in Related Measurements

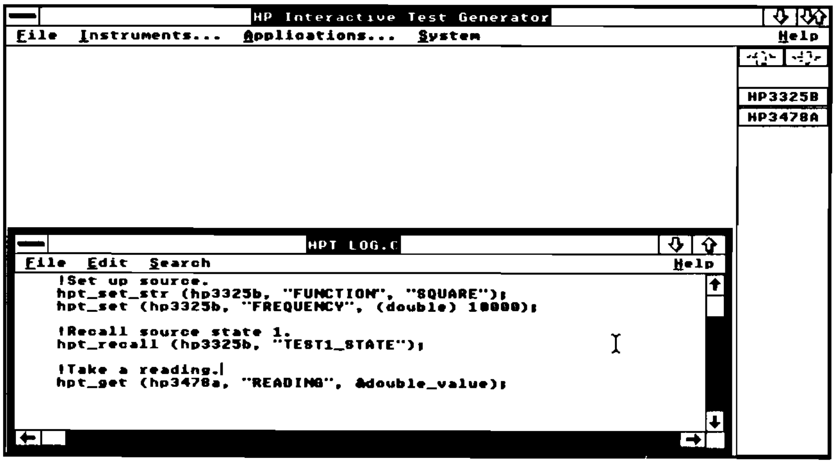
Generating Code

As you adjust controls and make measurements, HP ITG can automatically translate your actions into test code using the programming language you selected during installation. You can even perform simple editing and add comments without leaving HP ITG.

The code that HP ITG generates consists of calls to subprograms in the **HP ITG Library.** The subprograms access the instrument drivers. Basically, each subprogram does one of the following:

- Sets a control.
- Gets a reading (or makes a measurement).
- Recalls a stored instrument state.

To give you the best possible throughput when recalling a stored state, HP ITG sends the minimum set of commands needed to change the instrument to the desired state. HP ITG monitors the current instrument state and determines the specific commands required to change to the new state. This eliminates the time required to send extraneous commands as well as the time required for the instrument to interpret the commands and respond.



```
    !Set up source.
    hpt_set_str (hp3325b, "FUNCTION", "SQUARE");
    hpt_set (hp3325b, "FREQUENCY", (double) 10000);

    !Recall source state 1.
    hpt_recall (hp3325b, "TEST1_STATE");

    !Take a reading.
    hpt_get (hp3478a, "READING", &double_value);
```

Figure 1-3. HP ITG Generates Instrument-Control Code

Completing Your Program

To include features from other applications, you can quit HP ITG and include calls to routines in other MS-DOS libraries using your programming language editor or Quick environment to build a complete test application. These libraries can provide the routines to accomplish various tasks such as data acquisition and analysis, and database and graphics development.

Editing a Program

You can integrate the code generated by HP ITG into new or existing programs using the HP ITG **Editor Window** or the editor of your choice. You then can expand your program to include other subprograms from the HP ITG Library or other MS-DOS libraries. These might include a user interface, data acquisition/analysis, and graphics as required for your test program specifications.

Run-Time Environment

The run-time environment involves creating the standalone program, an executable, then running it on a test station controller. You should do this after testing and debugging your program. The test station's run-time controller should be configured to support the program you run on it.

About This Handbook

To help you use this handbook, please read about the following conventions used to describe HP ITG.

Typefaces

The following typefaces are used to help you understand how terms and phrases are used:

Bold	**Bold** indicates the introduction of an HP ITG feature. These terms are listed in the glossary.
Italics	Terms are printed in *italics* for emphasis. The HP ITG Library descriptions in chapters 14 and 15 also use italics to distinguish the names of variables.
`Computer`	Computer-style type indicates commands you should type or that appear in the HP ITG development environment. Program listings also appear in this type.
$\boxed{\text{Key}}$	Individual keyboard keys are distinguished by the keycap-style border around the key's name.

Names

The following names are used to help you relate written instructions to symbols appearing on the monitor's display:

screen	"Screen" refers to the monitor's display.
display	"Display" refers to the area on an instrument driver's soft panel that displays the results of instrument measurements.
pointer	"Pointer" refers to the mouse cursor appearing in the HP ITG work area.
cursor	"Cursor" refers to the indicator where text is entered during any editing activity.

Libraries

HP ITG provides libraries of subprograms for the supported programming languages, QuickBASIC and C. Differences exist between the libraries that require this handbook to use some conventions to refer to these libraries in common.

Technically, the HP ITG Library for QuickBASIC uses subprograms. Their names do not contain underscores, as in **`hptrecall`**. The HP ITG Library for C uses functions with names that do contain underscores, as in **`hpt_recall`**. When this handbook describes the HP ITG Library in general, the references are to *subprograms,* and the names use *underscores.* Information specific to the individual libraries uses their respective naming conventions.

System Requirements

Overview

System requirements vary depending on whether you are using HP ITG in the development environment to control instruments and generate code, or whether you are running your test program in the run-time environment.

Development Requirements

You can develop programs using HP ITG on various personal computer configurations. The following list describes the basic requirements to begin using HP ITG in the development environment:

Computer

HP Vectra personal computer or IBM PC-AT or compatible running with i286 or i386 microprocessor and MS-DOS 3.0 (or higher) operating system.

Memory

i286 System

640 Kbytes plus 1 to 2 Mbytes of expanded memory. The expanded memory must conform to Lotus^{R}-Intel-Microsoft (LIM) 4.0 Expanded Memory Specification (EMS).

i386 System

2 to 3 Mbytes of total system memory. See appendix C in this handbook for more information about memory managers.

Disk Drives

20 Mbytes or larger hard disk drive and one 3.5- or 5.25-inch floppy disk drive.

Monitor and Video Adapter

Enhanced Graphics Adapter (EGA) or Video Graphics Adapter (VGA) with compatible color or monochrome monitor.

Interface Standard

You must install at least one interface board if you want to control any instruments. HP ITG supports the following IEEE-488 interfaces:

- Hewlett-Packard HP-IB (HP 82335A or HP 82990A).
- National Instruments Corporation GPIB-PCII/IIA.
- Radix MicroSystems, Inc. VXIbus.

HP ITG can support more than one interface board at a time. Each board must be set to a different select code. See appendix B in this handbook for more information about using these interface standards.

Pointing Devices and Printers

HP ITG requires you to use a pointing device such as a mouse. You may use any serial or parallel interface device supported by MS-DOS and Microsoft Windows.

You can use any printer supported by MS-DOS and Microsoft Windows. You will be able to configure its interface in HP ITG. Though a printer is not required, it lets you print the HP ITG screen, as well as the contents of the Editor window and online Help. See chapter 6 for more directions about configuring your printer.

Programming Language

In addition to HP ITG, you should install software for the programming language you want to use for test code generation. HP ITG supports the following programming languages:

- Microsoft C 5.X
- Microsoft QuickC 2.0
- Microsoft QuickBASIC 4.5

Run-Time Requirements

The run-time computer system should be compatible with the test program you develop. Generally, the run-time system requirements are less than those for a development system. The actual run-time system you assemble depends mostly on your test program needs. If you have included features and libraries from other applications to provide a user interface, data collection, and analysis, you might need to assemble a more powerful run-time system. Typically, a test application that just controls instruments should consist of the following components:

Computer

HP Vectra personal computer or IBM PC-AT or compatible running with i286 or i386 microprocessor and MS-DOS 3.0 (or higher) operating system.

Memory

i286 System

640 Kbytes plus 1 to 2 Mbytes of expanded memory. The expanded memory must conform to Lotus^{R}-Intel-Microsoft (LIM) 4.0 Expanded Memory Specification (EMS).

i386 System

2 to 3 Mbytes of total system memory. See appendix C in this handbook for more information about memory managers.

Disk Drives

20 Mbytes or larger hard disk drive for data storage and one 3.5- or 5.25-inch floppy disk drive as needed to transport programs.

Monitor and Video Adapter

You can use any monitor and video adapter that supports the run-time requirements for your test application.

Interface Standard

Use the interface configuration required by test application.

Pointing Devices

Use devices if required by test program.

Printers

Use any printer supported by MS-DOS if required by test application.

Quick Environments

To run your test application in a Quick environment, you must install the appropriate Quick product in your run-time system that supports the language you are using.

3 Installing HP ITG

Overview

This chapter explains the steps you should follow to install and use HP ITG. An installation program called Setup is provided to help you install the HP ITG files and prepare the development environment.

The chapter's sections describe each of the following steps:

- Gathering materials.
- Preparing your computer.
- Installing support applications.
- Installing HP ITG.
- Starting HP ITG.

When you prepare your computer, add accessories, and install supporting software, use the instructions provided with each package. The Setup program supplied with the HP ITG distribution disks installs only those files required for HP ITG.

Please complete each step in the given order. If you have already completed certain steps, just jump ahead to the next one. The HP ITG installation itself takes about 20 minutes and uses about 4 Mbytes of hard disk memory.

Gathering Materials

Before you begin the installation process, gather the materials you will need:

- Computer's video adapter, monitor, printer, and mouse.
- MS-DOS operating system software.
- Memory board with LIM 4.0 EMS.
- Interface board.
- Support applications software.
- HP ITG software.

The additional memory board is required for computers equipped with the i286 microprocessor. Computers equipped with the i386 microprocessor usually have the additional memory already installed.

Preparing Your Computer

Set up your computer by installing its accessories and the MS-DOS operating system software. Use the respective manufacturers' instructions for these installations.

Install the boards required for common computer peripherals such as a printer, the video adapter, pointing device, and network communications. Be sure to install the instrument interface and memory boards which are required by HP ITG. Connect the monitor, keyboard, mouse, and printer.

Note

If you use a printer with an HP-IB interface, the printer must be configured so it is connected to the LPT1 port.

Installing the Interface

You should install the interface in one of the computer's I/O slots now. If you received the HP-IB interface with your HP ITG software, the guide packaged with the interface, *Installing the HP-IB Interface,* explains configuration and installation procedures. Though the factory settings are satisfactory for most systems, the quide's section, "Setting Switches," explains how to reconfigure the interface for your computer. That section explains how to set the select code and prevent address space conflict with the memory manager and other interfaces you might be using in your computer.

If you are using another manufacturer's interface, be sure to read the instructions for that interface. Other manufacturers use different methods to set select codes and prevent address space conflicts.

Note For more information about setting select codes for the interfaces supported by HP ITG, see appendix B in this handbook, "I/O Interfaces."

Installing Memory

If you are installing additional memory in your computer, you need to do so *before* you install the HP ITG software. For i386-based computers, configure additional memory as extended. For i286-based computers, configure additional memory as expanded. Additional configuration may be required for the additional memory to prevent address space conflicts with the instrument interface. Read the memory manufacturer's instructions for details. For information about memory managers in i386-based computers, see appendix C in this handbook, "Expanded and Extended Memory."

MS-DOS

Install the MS-DOS operating system software. Edit the files, `CONFIG.SYS` and `AUTOEXEC.BAT` as needed for the accessories you have installed in your computer. Be sure the MS-DOS operating system recognizes all of the installed accessories.

Installing Support Applications

You should install the support applications you will use with HP ITG. The programming language software you will use to develop your test programs is the most important application to install now.

When you install Microsoft C, QuickC, or QuickBASIC, please use the guidelines described in the following subsections.

You may want to add other applications to include additional functionality to your test programs. These might include data collection and analysis, and graphics for a user interface.

Installing Microsoft C and QuickC

Caution

HP ITG only supports version 5.X of Microsoft C and version 2.0 of Microsoft QuickC.

HP ITG Setup works best if you follow these guidelines. The QuickC installation program makes it easy for you to follow them, and HP ITG Setup expects this configuration:

- Select the emulator library for the math options.
- Select only the medium model when defining the Memory Model.

- You may add the GRAPHICS.LIB library if you plan to include graphics in your C programs. HP ITG does not require this library for its operation.
- Accept the default options for the QuickC directories.
- Edit AUTOEXEC.BAT as described in the QuickC installation instructions to set environment variables for LIB and INCLUDE and modify the PATH command.

Installing QuickBASIC

Caution HP ITG only supports version 4.5 of Microsoft QuickBASIC.

HP ITG Setup works best if you follow these guidelines. The QuickBASIC installation program makes it easy for you to follow them, and HP ITG Setup expects this configuration:

- Use the default choices when installing QuickBASIC.
- Be sure all QuickBASIC files are loaded into directory QB45.
- Add the QB45 directory to the PATH command in AUTOEXEC.BAT.

Installing HP ITG

This section explains how to use the HP ITG Setup program. Setup lets you specify your computer's configuration, identify the programming language, name directories, copy files, and edit the WIN.INI and AUTOEXEC.BAT files. Setup presents the instructions you'll need to complete the HP ITG configuration and install the required files on your computer's hard disk drive. Refer to this section during Setup for more information.

When you finish Setup, be sure to store your HP ITG distribution disks in a safe place.

Starting Setup

Before you start Setup, the screen should display the MS-DOS prompt for the active drive, such as `C>`.

Note

The installation instructions refer to floppy disk drive A. You may use any compatible floppy disk drive, but you must be sure to specify the drive as shown in the following procedure. Then use the same drive for the remainder of the installation.

1. Insert the HP ITG Disk 1 in floppy disk drive A.
2. Type `a:` and press Enter.
3. Type **`setup`** and press Enter.
4. Follow the Setup program's instructions.

Microsoft Windows

Setup can install a portion of Microsoft Windows which supplies the graphics features for HP ITG. This creates a single application environment (SAE) for HP ITG. You will not have all of the features normally available with the complete Windows application. You can choose whether or not to install the SAE at this time. If you install the SAE, Setup asks several questions about your computer system to configure the Windows SAE correctly.

If you choose to install the complete Windows application, you need to purchase and install it separately, then rerun HP ITG's Setup. If you install the complete Windows application, be sure to use the default directory name offered by the installation program. Then add the directory name to the PATH command in the `AUTOEXEC.BAT` file. The Windows installation program describes how to do this. HP ITG's Setup expects to find this same directory name.

Note

You can install the full Windows application after installing the SAE by using the following procedure:

1. Delete the files `HPITG.COM`, `WIN200.BIN`, and `WIN200.OVL` from the directory `HPITG`.
2. Install Windows as instructed.
3. Edit the new `WIN.INI` file to include the HP ITG settings added when you installed the SAE.

Selecting the Programming Language

Setup displays Microsoft QuickBASIC as the default selection. You may select either of the other supported languages, Microsoft C or Microsoft QuickC. HP ITG generates code in the language you select.

Identifying Directories

Setup identifies five default directories used during the installation. The specifications include the hard disk drive where files will be installed. If you used the default options when installing the previous applications, Setup's default directories will support your current directory structure.

Caution

Prevent losing data on your hard disk drive. Setup will create needed directories if they do not exist. If a directory already exists, Setup **overwrites** existing files that happen to have the same names as the new files. Do not use the name of an existing directory unless you are sure there are no files that will be overwritten.

Setup copies files to directories based on the following rules:

- All HP ITG files are put in `\HPITG`.
- All SAE Windows files are put in `\HPITG`.
- If using Windows/286, all Windows files are put in `\WINDOWS`.
- If using Windows/386, all Windows files are put in `\WIN386`.
- All utilities are put in `\BIN`.
- If using QuickBASIC, all libraries and include files are put in `\QB45`.
- If using QuickC:
  - All libraries are put in `\QC2\LIB`.
  - All include files are put in `\QC2\INCLUDE`.
- If using C:
  - All libraries are put in `\LIB`.
  - All include files are put in `\INCLUDE`.

Editing WIN.INI

Setup edits the Windows initialization file, `WIN.INI`, which was installed with Windows. These required changes adjust the colors used in the HP ITG environment and specify the selected programming language. The additional commands are appended to `WIN.INI` in a separate section for HP ITG settings. These settings do not adversely affect the use of other applications.

Setup gives you the option to overwrite the existing `WIN.INI` file or to create an alternate file with the file name, `WININI.ITG`. If you choose to create the alternate file, you will need to edit the `WIN.INI` file so the commands take effect when you start HP ITG. To edit

WIN.INI yourself, add the changes from the WININI.ITG file.

Setting HP ITG Colors. If necessary, you can edit the WIN.INI file to change the colors HP ITG uses. The following explanations apply if you are using a standard VGA display in color mode. HP ITG uses a color-set established for the personal computer (PC) environment. The following command in the WIN.INI file enables that color-set:

```
MATCH_COLORS=0
```

To change the color-set to the same one used by the HP ITG version that runs on the HP 9000 Series 300 workstation, set the MATCH_COLORS command to 1:

```
MATCH_COLORS=1
```

Changing the Programming Language. You can switch languages HP ITG uses to generate code by editing the WIN.INI file.

Caution

You must have only one programming language selected in the WIN.INI file.

The language you pick must be supported by HP ITG. Language selection is done by removing the semicolon comment marker from in front of the `language=` command line identifying the language you prefer. You must then insert a semicolon comment marker in front of all other `language=` command lines. The following sample lines from WIN.INI shows QuickBASIC as the selected language:

```
; language=msc
language=qb
; language=qc
```

If you change languages, be sure to set the environment variable in AUTOEXEC.BAT for the directories where the HP ITG include files exist for each language. The following example shows what the command should look like, assuming the default directories were used during installation:

```
set INCLUDE=C:\QC2\INCLUDE;C:\INCLUDE;C:\QB45
```

Editing AUTOEXEC.BAT

Setup edits the AUTOEXEC.BAT file. The required changes add the environment variable HPITG. Setup also adds any new directory created by the HP ITG Setup program into the MS-DOS search path. The changes should not interfere with the use of the operating system or other applications.

Setup gives you the option to overwrite the existing AUTOEXEC.BAT file or to create an alternate file with the file name, AUTOEXEC.ITG. If you chose to create the alternate file, you will need to edit the AUTOEXEC.BAT by adding the changes included in the AUTOEXEC.ITG file.

After Setup is finished, reboot your computer so the new commands take effect. Press Ctrl, Alt, and Del simultaneously.

If, after rebooting, you get the message `Out of environment space`, it is because too many environment variables have been added to AUTOEXEC.BAT. See your MS-DOS manual to increase the environment space.

Starting HP ITG

When the installation is done, you should verify the status of the instrument interface and expanded memory before starting HP ITG. Two programs supplied by HP ITG supply the information for you. Run them from the MS-DOS command line.

1. To verify the interface status, type **`hpiostat`**, then press (Enter).
2. To verify the memory status, type **`hpemstat`**, then press (Enter).

Caution

To ensure proper operation of HP ITG, leave *all* files copied from the HP ITG distribution disks in their current directories.

Before starting HP ITG, you may want to create a working directory for the files you create while using the development environment. Start HP ITG from the MS-DOS command line.

1. Type **`hpitg`**.
2. Press (Enter).

To learn about the HP ITG development environment, please read chapter 4 in this handbook, "Getting Started."

4

Getting Started

Overview

After you install and start HP ITG, use this chapter to help you begin using HP ITG and understand its relationship to the working environment used to develop complete test programs.

HP ITG System Overview

The following figure shows the HP ITG development environment with an instrument driver soft panel added to the work area and the Editor Window. Take a few moments to learn the locations of the various menu and window controls. Appendix A in this handbook, "Menus Index," gives detailed information about the operation of each command and the menus they provide.

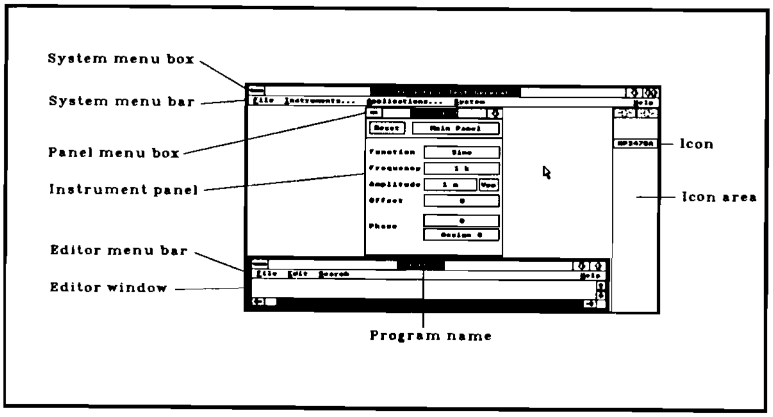


Figure 4-1. The HP ITG Development Environment

Learning the Basics

You can control the HP ITG development environment menus and windows using the computer's keyboard and mouse.

Using the Mouse

As you move the mouse across a flat surface, notice its **pointer** moving across the screen. You can use the mouse to operate HP ITG by moving the pointer so it touches a screen element, then pressing the left mouse button. The following terms are used in the instructions to describe mouse operation:

point	Move the mouse pointer until its tip touches the element you want to select.
click	Press and release the left mouse button.
drag	Press the left mouse button, and while holding it down, move the mouse to a new location.

Using the Keyboard

You can use the keyboard to choose commands and make menu selections. First, activate the window you want to use, such as the main HP ITG window or the Editor window, by clicking on the preferred window. Then use the keyboard to make selections:

- To select a command in a menu bar, press (Alt) and the underlined letter in the command name.
- To make a menu selection, press the underlined letter in the command.
- To activate the different fields in **dialog boxes**, press (Tab).
- To cancel any menu or dialog box, press (Esc).

- To select a name from a list, press (Tab) to access the **list box,** press the arrow keys to highlight the name you prefer, then press (Enter). If you know the name, you can type it into the text box, then press (Enter).

Online Help System

HP ITG provides an online Help system, which includes a tutorial that demonstrates the basic operation of HP ITG. You can run the tutorial using the following instructions:

1. Click on `Help` on the **System menu bar.**
2. Click on `Tutorial` ...

You might want to take the time to use the tutorial. It illustrates the major features of HP ITG. The exercise in chapter 5 of this handbook presents instructions to help you learn how to develop your own test application.

The HP ITG Help system also includes the following information:

How-to ...

Provides step-by-step instructions on the most common tasks you'll perform with HP ITG.

Instrument Help ...

Provides information on all instrument drivers included with HP ITG.

Application Help ...

Provides information on the applications distributed with HP ITG.

Subprograms ...

Provides information on all of the subprograms provided with the HP ITG Library.

I/O Status ...

A message box displays the select code for each interface installed in your computer.

Latest Information ...

Lists the instrument drivers currently available with HP ITG, and provides information about HP ITG revisions not included in the printed documentation.

The Working Environment

To develop test applications, you will be using a development environment and a run-time environment. The development environment involves using HP ITG to generate instrument-control code and your selected Quick programming language environment to debug the test application. The run-time environment involves creating an executable which you can install on other PCs and run. Generally, all applications you use to develop and run your programs will start from the MS-DOS command line and will be used in a certain sequence.

The Development Environment

The development environment uses HP ITG to create a **soft test system** and generate instrument-control code. The soft test system contains the instrument drivers and their states that simulate the actual test station hardware your test application will control. The soft test system is saved in a data file on your computer's mass storage. This data file is known as a **workfile.** The generated program uses the workfile to control the test station instruments when running the test application. To continue test application development, you can enhance and debug your program using the particular programming language environment you chose during installation.

Ways to Start HP ITG

After completing the installation described in chapter 3, you start HP ITG by entering `hpitg` at the MS-DOS prompt. This loads the default program file, `HPT_LOG`, into the HP ITG Editor window. It will include a file name extension, such as `.C` or `.BAS`, depending on the programming language you selected.

If you have created a workfile during a previous development session, you can automatically load it when starting HP ITG. For example, if you saved a workfile under the file name `FRQ_RESP.WF`, you can load it when starting HP ITG:

- Type `hpitg frq_resp` and press (Enter).

Note Workfile file names must include the `.WF` extension for HP ITG to recognize them as workfiles. However, you do not need to include the extension when starting HP ITG or when saving workfiles. HP ITG automatically appends the extension.

Adding Instrument Drivers

Whether you are creating a new workfile or modifying an existing one, you can add drivers to the soft test system for the instruments you want to control. The drivers provide soft panels that let you control instruments and set up instrument states.

Saving the Workfile

When the soft test system contains the drivers and instrument states your test program will require, save the soft test system in a workfile. Saving the workfile before generating instrument-control code ensures HP ITG will generate code that supports the specific workfile.

Generating Code

Use the HP ITG Editor and the instrument soft panels to generate the instrument-control code. HP ITG generates the code in the programming language you specified during the installation procedure. The editor's features let you observe instrument-control code generation as you interact with the instrument soft panels.

Saving the Program

After generating code, save your program. You will use this program file to complete your test application development.

Starting the Quick Program Environment

You can start QuickC or QuickBASIC from the MS-DOS prompt using special batch files supplied by HP ITG. These batch files start the Quick environment, and load your test program. The batch files include the options to support the calls made to HP ITG subprograms. For QuickBASIC, the option loads the **HP ITG Quick Library.** For QuickC, a make file is created and loaded with the program. Just execute the batch file for the language you are using and include the program name as shown in these examples:

```
   qbstart hpt_log.bas
   qcstart hpt_log.c
```

Completing Your Program

Now you can add the program-control code and other enhancements to your program. These enhancements can include calls to additional HP ITG Library subprograms, or to routines in other libraries to provide data collection and analysis, graphics, and a user interface for your test application. This process involves the development and debugging required to produce a complete test application.

Note

If you are using Microsoft C, you should use a text editor to complete the coding. Then run the compiler/linker as described in that product's user's guide to debug and run your program. HP ITG provides the batch file, `CMAKE.BAT`, to run the compiler/linker. Enter **`cmake hpt_log`** to create an executable.

The Run-Time Environment

The run-time environment involves creating the standalone program to control a test station. You can create QuickC or QuickBASIC standalone programs from the MS-DOS prompt using special batch files supplied by HP ITG. These batch files compile your program and link against the correct HP ITG libraries, then create the executable. Just run the batch file for the language you are using and include the program name as shown in the following examples:

```
qbmake hpt_log
qcmake hpt_log
```

You can then distribute the executable, `HPT_LOG.EXE`, along with `HPITG.ERR` to run-time computers at test stations and run it by entering **`hpt_log`** at the MS-DOS prompt.

5

A Complete Example

Overview

The following exercise illustrates the key features of HP ITG. Before you do the exercise, you must install HP ITG and its drivers (see chapter 3 for details). See the glossary for information about unfamiliar terms.

A Frequency-Response Measurement

This example illustrates the steps needed to write a program that measures the frequency-response of a circuit. The program you write will set the HP 3325B Function Generator to produce a 1 Vrms sinewave. Then it will change the frequency in 1 kHz steps from 10 kHz through 20 kHz. The program will also set the HP 3478A Multimeter to take 4.5 digit, AC voltage readings at each frequency step. Even if you don't actually have an HP 3325B or an HP 3478A attached to your interface, you can use HP ITG to generate the example test program.

To complete this example, you will perform the following steps:

1. Run HP ITG.
2. Create a soft test system.
3. Set up the instruments for the measurement.
4. Save the soft test system.
5. Generate the instrument-control code.
6. Edit the code.
7. Save your program.
8. Start the Quick environment.
9. Create an executable program.

Step 1: Run HP ITG

After you have installed HP ITG and supporting applications (chapter 3), you can run HP ITG and begin developing your test program. This step sets `HPITG` as the working directory and starts HP ITG. By identifying a working directory now, the files you save while using HP ITG will be placed in that directory.

Assuming the monitor is displaying the MS-DOS prompt, `C:\`, change to the HPITG directory and run HP ITG:

- Type `cd \hpitg` and press Enter.
- Type `hpitg` and press Enter.

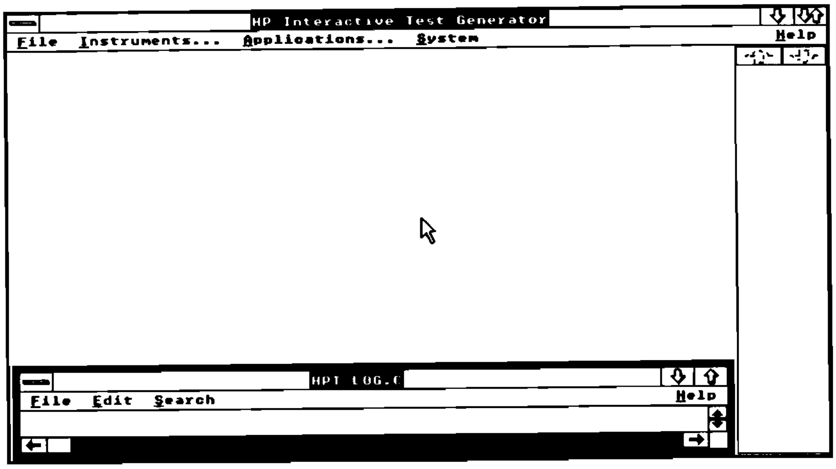


Figure 5-1. HP ITG at Startup

Step 2: Create a Soft Test System

In this step, you add the instrument drivers you need and specify any necessary configuration information such as HP-IB addresses and subaddresses, timeouts, and logical names. This combination of instruments is a **soft test system.** You will save this soft test system later as an HP ITG workfile. Then, when you run your program, HP ITG uses the workfile to know which commands to send to which instrument.

To add the HP 3325B to the work area:

1. Click on `Instruments ...` on the System menu bar.
2. Select `HP3325B.ID` and click on `Open`.

   `HP3325B.ID` is the name of the driver file for the HP 3325B Function Generator. If `HP3325B.ID` is not listed, scroll to it by clicking on the arrows at the top and bottom of the scroll bar next to the list.

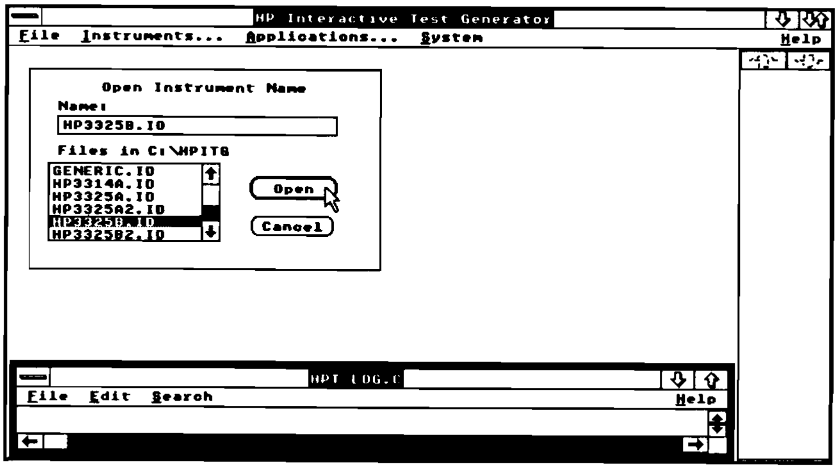
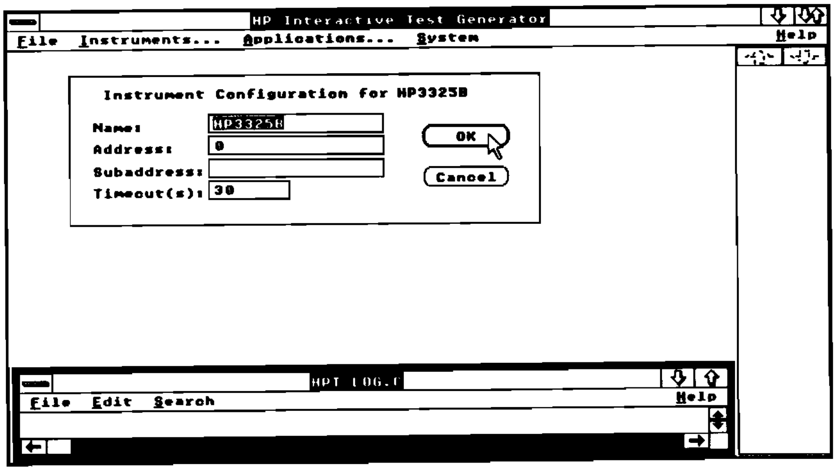


Figure 5-2. List Box for Instruments

HP ITG should now display the dialog box shown in the following figure. A cursor appears in the `Name:` field.


HP Interactive Test Generator
File Instruments... Applications... System Help
OK
Cancel
HPT LOG.T
File Edit Search Help

Instrument Configuration for HP3325B

Name: HP3325B
Address: 0
Subaddress:
Timeout(s): 30

Figure 5-3. The Instrument Configuration Dialog Box

3. Do not edit the `Name:` field.

   The `Name:` field contains the default logical name that HP ITG has selected for the instrument. This field lets you enter a more descriptive name for an instrument, such as SYNTHESIZER instead of the default HP3325B. Also, if you should ever use more than one of the same model instrument in your soft test system, you could name the first one SRC_1, and the second, SRC_2.

4. Press (Tab) to select the `Address:` field.

   a. If you do not have an HP 3325B attached to your interface board, leave the `Address:` field set to `0`. This prevents HP ITG from communicating with the interface board.

   b. If you do have an HP 3325B attached to your interface board, type in the instrument's three-digit device select code/address. The select code identifies the interface board. The address identifies a particular instrument attached to that board. For example, if the instrument's HP-IB address is set to 17, and it is attached to an interface board set to select code 7, then type `717` into the `Address:` field.

   c. Click on `OK`.

5. Repeat steps 1-4 for the HP 3478A. Its driver file name is `HP3478A.ID`.

6. Move the `HP3478A` panel to the side by clicking on its title bar and dragging the panel.

The display on your screen should resemble the following figure.

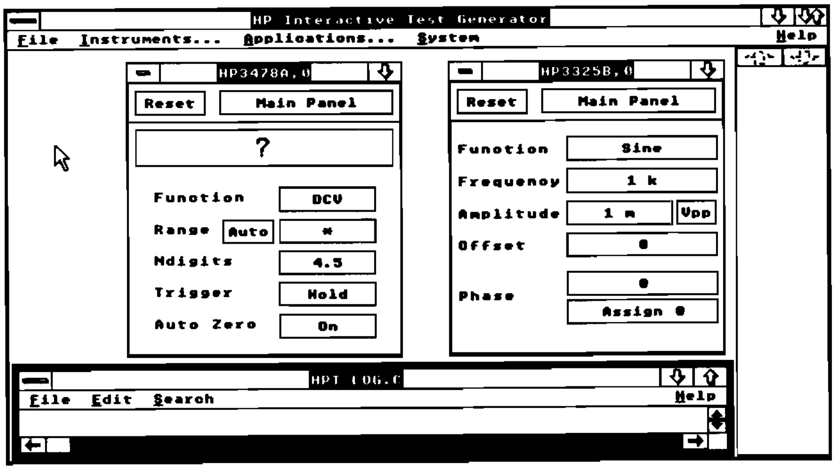


Figure 5-4. HP 3478A and HP 3325B Are Part of the Soft Test System

Step 3: Set Up the Instruments

The Initial State

The driver files for the HP 3325B and HP 3478A contain information that tells HP ITG how to configure their panels when you add them to your soft test system. This information is known as the **instrument state.** The instrument state refers to the values that appear on the panel's control buttons. The particular values that appear when you add the panels are the **initial values** and exist for every driver as the **initial state.**

Now, notice the **Reset** button in the top left portion of each panel. If you get lost while working with the instrument panels, just click on this button to return the panel to its initial state.

Multi-Layered Panels

Many instrument drivers use subpanel layers to define more instrument features. The control to the right of the `Reset` button lets you change subpanels. Each subpanel gives you access to more controls, displays, and buttons. You will not need to change subpanels for this exercise.

Set Up the HP 3325B Panel

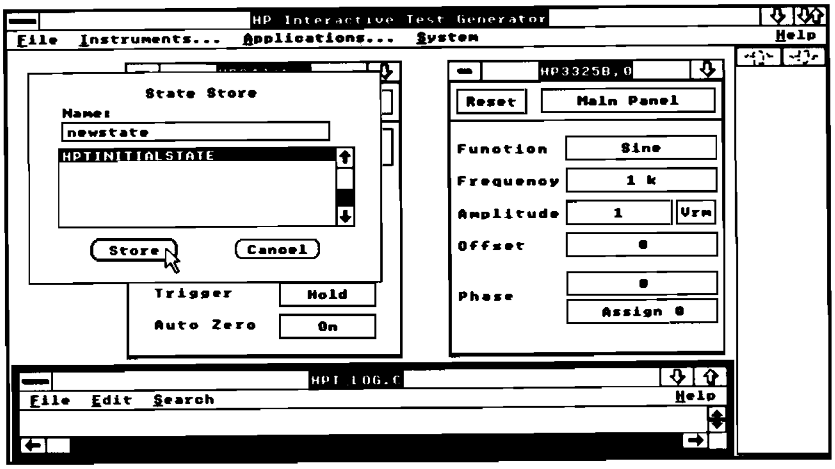
Set the panel for a 1 volt rms sinewave. You will only need to change the amplitude setting now, since the remaining initial values are appropriate for this example. You will change the frequency later when you generate code in Step 5.

1. Click on `Reset` to ensure the panel is set to the initial state.
2. Set the amplitude to 1 volt rms.
    a. Click on the button next to `Amplitude`.
    b. Click on `1`, then click on `OK`.
    c. Click on the amplitude units box to the right of `Amplitude`.
    d. Click on `Vrm`, then click on `OK`.

Store an Instrument State

Now that you have the HP 3325B set up as needed, HP ITG lets you store the setup as a state so you can use it throughout your program.

1. Click on the instrument panel menu box (top left corner) to access the panel menu.
2. Click on `Store State ...`
3. Type `newstate` and press Enter.



Name: newstate

HPINITIALSTATE

Trigger	Hold
Auto Zero	On

Function	Sine	
Frequency	1 k	
Amplitude	1	Vrm
Offset	0	
Phase	0	
	Assign 0	

Figure 5-5. The State Store Dialog Box

Set Up the HP 3478A Panel

Set the panel to read AC volts. You will need to change only the `Function` control because the multimeter's remaining initial values are appropriate for this example.

1. Click on **Reset** to ensure the panel is set to the initial state.
2. Set `Function` to read AC volts.
   a. Click on the button next to `Function`.
   b. Click on **ACV**, then click on `OK`.
3. Store this instrument setup as a state.
   a. Click on the instrument panel menu box (top left corner).
   b. Click on `Store State`.
   c. Type `newstate` and press (Enter).

Note

`newstate` can be used as the state name for both the HP 3325B and the HP 3478A because state names are linked directly to the particular instrument driver. Since state names are case-sensitive, be sure to use upper and lower case characters consistently.

Step 4: Save the Soft Test System

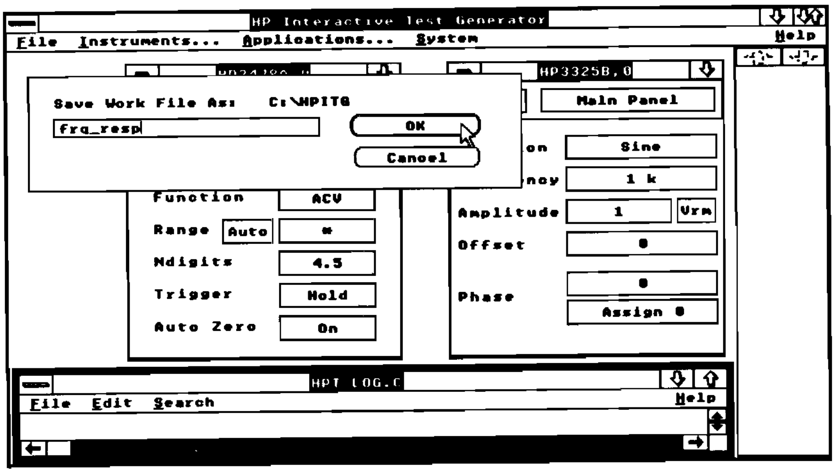
It is very important to save the soft test system before you begin generating code. The soft test system is saved in a workfile. This workfile must exist to provide the information HP ITG uses to generate the correct code for the soft test system and its instrument states. Be sure you have added all of the instruments you will use and created the instrument states. As an added benefit, the next time you work with HP ITG, you can reuse the workfile to generate other test programs.

Caution

Before you begin to generate code, save the soft test system that you created in the previous step. This ensures that HP ITG will generate the correct code.

1. Click on `File` on the System menu bar.
2. Click on `Save Workfile As` ...
3. Type in a workfile name, such as `frq_resp`, then press (Enter).

This saves the soft test system in the workfile, `FRQ_RESP.WF`. HP ITG automatically appends the `.WF` extension to identify these files as workfiles.


HP Interactive Test Generator
File Instruments... Applications... System
Help
HP3325B,0
Main Panel
OK
Cancel
HPT_LOG.C
File Edit Search
Help

Save Work File As: C:\HPITG
frq_resp

Function ACV
Range Auto *
Ndigits 4.5
Trigger Hold
Auto Zero On

on Sine
ncy 1 k
Amplitude 1 Vrm
Offset 0
Phase 0
Assign 0

Figure 5-6. Saving the Soft Test System to a Workfile

Step 5: Generate Code

The code that HP ITG generates is displayed in the HP ITG Editor window. You can edit each line of code, and add and delete whole lines. As you work in the HP ITG Editor, your keyboard's cursor and edit keys, such as (Home), (End), and the arrow keys are fully functional.

Note

The current file name for your program is displayed on the Editor title bar. The file name extension is consistent with the programming language you selected when you installed HP ITG. If you are using C, the default file name is `HPT_LOG.C`. If you are using QuickBASIC, the default file name is `HPT_LOG.BAS`.

Preparing the Editor Window

Initially, the Editor's size can be kept small to allow full view of the panels. As you change the panel settings, HP ITG generates code in the Editor regardless of its size. You can resize the Editor to a convenient size at any time.

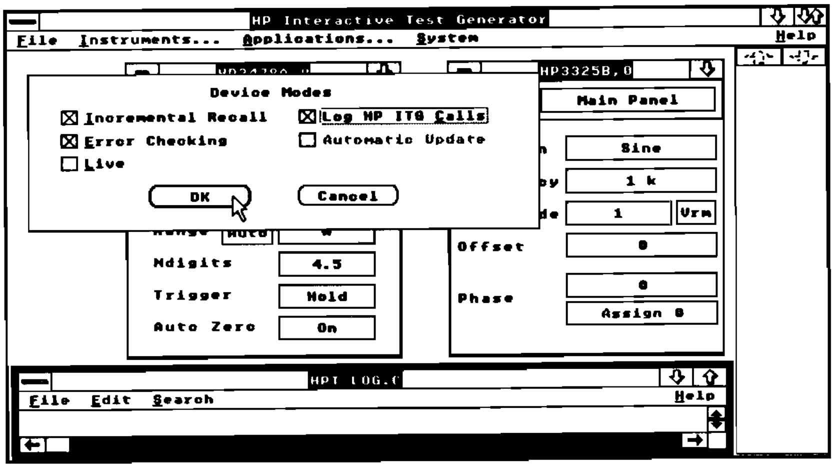
1. Click on the `HPT_LOG` icon if the Editor window is not displayed. HP ITG icons appear in the column at the far right of the work area.
2. Clear the Editor if it contains program code.
   a. Click on `Edit` on the **Editor menu bar.**
   b. Click on `Select All`.
   c. Click on `Edit` again, then click on `Clear`.

Set Modes

Each instrument panel provides a mode option called Log HP ITG Calls. You must enable this mode for HP ITG to generate instrument-control code. With Log HP ITG Calls enabled, HP ITG generates calls to subprograms in the HP ITG Library. The HP ITG subprograms interpret the panel interactions to control instruments. The code is generated in your selected programming language. Turn on Log HP ITG Calls mode for each instrument. If you entered HP-IB addresses for instruments connected to the bus, you can control them as you generate code by turning on Live mode at this time:

1. Click on the `HP3325B` panel menu box.
2. Click on `Modes` ...
3. Click on the box next to `Log HP ITG Calls` so an `X` appears in the box. An `X` in the box means the mode is on.
4. Click on the box next to `Live` so an `X` appears in the box if you connected the instrument to the interface bus.

5. Click on OK.
6. Enable Log HP ITG Calls (and Live if the instrument is connected to the interface) in the HP3478A panel.


HP Interactive Test Generator
File Instruments... Applications... System Help
HP3325B,0
Main Panel
HPT LOG.C
File Edit Search Help
OK
Cancel

Device Modes

☒ Incremental Recall ☒ Log HP ITG Calls
☒ Error Checking ☐ Automatic Update
☐ Live

Ndigits 4.5
Trigger Hold
Auto Zero On

Sine
1 k
1 Vrm
Offset 0
Phase 0
Assign 0

Figure 5-7. The Device Modes Dialog Box

Generate Initialization Code

The HP ITG Editor lets you automatically generate initialization code. All programs generated with Log HP ITG Calls require a certain amount of initialization code. This code includes program statements that declare certain variables and arrays, assigns HP-IB addresses to instrument variable names, and identifies the workfile containing the instruments and their states. You will enlarge the Editor for this step to view the initialization code as it's generated:

1. Point to the Editor's top border until a bidirectional arrow appears, then click on the border and drag it halfway up the screen.
2. Click on Edit on the Editor's menu bar.
3. Click on Generate Initialization Code.

4. Click on `Yes` in response to the message, since you already saved a workfile.

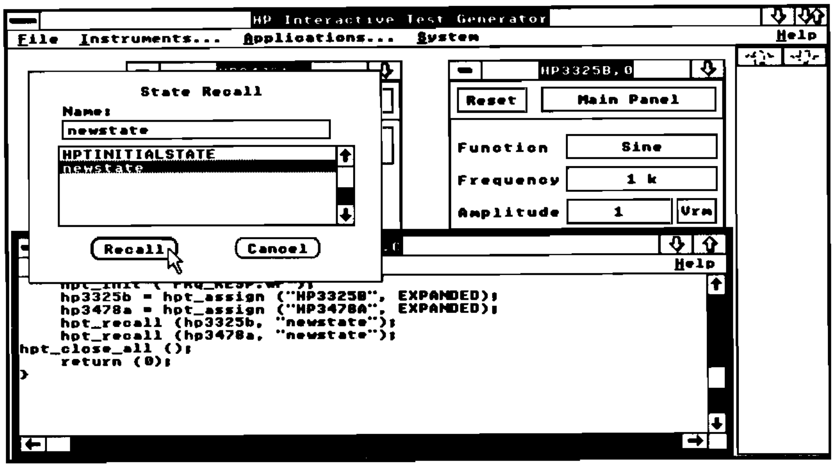
Note

After generating the initialization code, HP ITG places the Editor's cursor at the line where you should continue code generation. You can scroll through the code using the scroll bar at the right side of the Editor. To avoid moving the cursor when activating the Editor, click on the Editor's title bar, not in the edit area.

Recall States

Now you need to generate the code that sets up the instruments to make the measurement. You can do this by recalling the instrument states you stored in Step 3.

1. Click on the panel menu box in the top left corner of the HP 3325B panel.
2. Click on `Recall State` ...
3. Click on `newstate`, then click on `Recall`. This generates a call statement in the Editor window.
4. Repeat this procedure for the HP 3478A and recall its instrument state, `newstate`.



```
    [illegible]
    hp3325b = hpt_assign ("HP3325B", EXPANDED);
    hp3478a = hpt_assign ("HP3478A", EXPANDED);
    hpt_recall (hp3325b, "newstate");
    hpt_recall (hp3478a, "newstate");
hpt_close_all ();
    return (0);
}
```

Figure 5-8. The State Recall List Box

Adjust Frequency and Take a Reading

You are now ready to generate the code needed to take the first of 11 voltage readings, from 10 kHz through 20 kHz at 1 kHz intervals. As you change the instrument panel settings, HP ITG generates code that duplicates your actions when you run the program. You might need to adjust the Editor window size to access the instrument panels, but you will still see the statements as HP ITG generates them.

1. Change the HP 3325B frequency setting.
   a. Click on the button next to `Frequency`.
   b. Click on `1`, `0`, then **`kilo`**.
   c. Click on `OK` to generate the command.
2. Take a reading by clicking on the HP3478A panel's display.
3. Enlarge the Editor window to full screen to view the entire generated program. Click on the up arrow in the upper right corner of the Editor window. Use the scroll bar at the right to move through the code.

Step 6: Edit the Code

The following instructions explain how to add a loop in your program to make the other ten measurements, and print the results. The loop will repeat the calls to the subprograms that change the HP 3325A frequency and take reading with the HP 3478A. Please use the section in this step that describes the programming language you are using.

QuickBASIC

1. Click on the space just before the call to hptset and insert the following line:

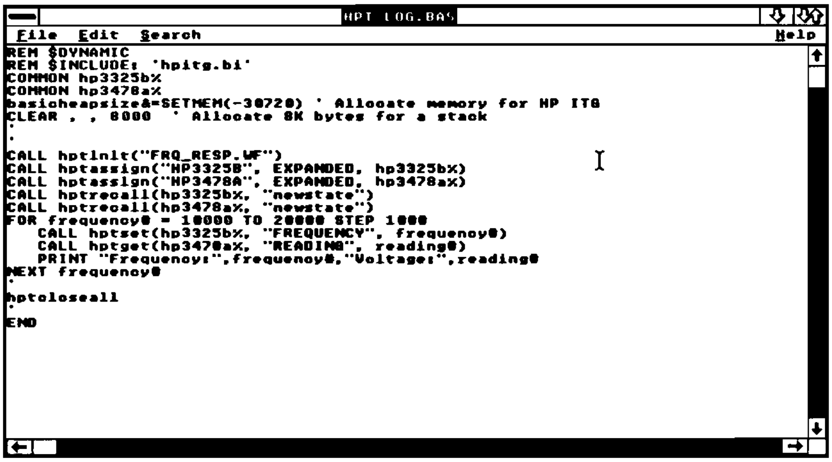
```
FOR frequency# = 10000 TO 20000 STEP 1000
```

2. Insert the following lines after the call to hptget:

```
PRINT "Frequency:",frequency#,"Voltage:",reading#
NEXT frequency#
```

3. In the line calling hptset, replace the number 10000 with frequency#.

Your QuickBASIC program should look similar to the following figure.



```
REM $DYNAMIC
REM $INCLUDE: 'hpitg.bi'
COMMON hp3325b%
COMMON hp3478a%
basicheapsize&=SETMEM(-30720) ' Allocate memory for HP ITG
CLEAR , , 8000  ' Allocate 8K bytes for a stack
'
'
CALL hptinit("FRQ_RESP.WF")
CALL hptassign("HP3325B", EXPANDED, hp3325b%)
CALL hptassign("HP3478A", EXPANDED, hp3478a%)
CALL hptrecall(hp3325b%, "newstate")
CALL hptrecall(hp3478a%, "newstate")
FOR frequency# = 10000 TO 20000 STEP 1000
   CALL hptset(hp3325b%, "FREQUENCY", frequency#)
   CALL hptget(hp3478a%, "READING", reading#)
   PRINT "Frequency:",frequency#,"Voltage:",reading#
NEXT frequency#
'
hptcloseall
'
END
```

Figure 5-9. The Final Program in QuickBASIC

QuickC

1. Click on the space just before the call to `hpt_set` and insert the following line:

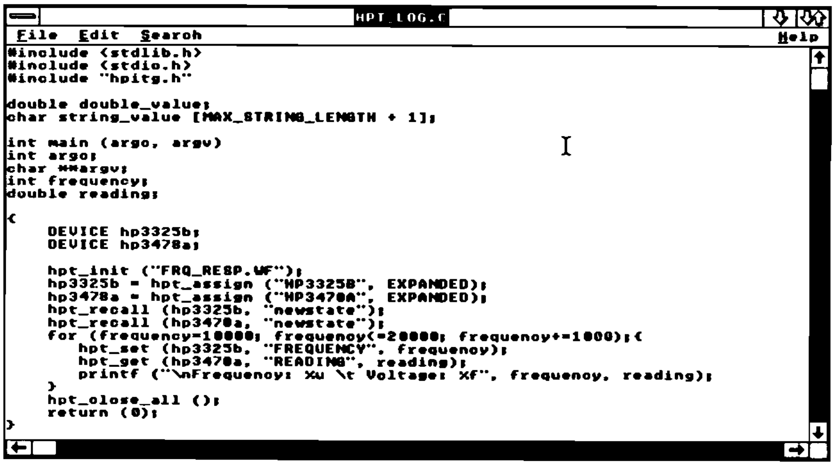
```
for (frequency=10000; frequency<=20000; frequency+=1000) {
```

2. Insert the following line after the call to `hpt_get`:

```
printf ("\nFrequency: %u \t Voltage: %f", frequency, reading); }
```

3. In the line calling `hpt_set`, replace the parameter `(double) 10000` with `frequency`.
4. In the line calling `hpt_get`, replace the parameter `&double_value` with `&reading`.
5. Add `int frequency` and `double reading` to the variable declarations at the beginning of the `main()` function.

Your C program should look similar to the following figure.



```
#include <stdlib.h>
#include <stdio.h>
#include "hpitg.h"

double double_value;
char string_value [MAX_STRING_LENGTH + 1];

int main (argc, argv)
int argc;
char **argv;
int frequency;
double reading;

{
    DEVICE hp3325b;
    DEVICE hp3478a;

    hpt_init ("FRQ_RESP.WF");
    hp3325b = hpt_assign ("HP3325B", EXPANDED);
    hp3478a = hpt_assign ("HP3478A", EXPANDED);
    hpt_recall (hp3325b, "newstate");
    hpt_recall (hp3478a, "newstate");
    for (frequency=10000; frequency<=20000; frequency+=1000);{
        hpt_set (hp3325b, "FREQUENCY", frequency);
        hpt_get (hp3478a, "READING", reading);
        printf ("\nFrequency: %u \t Voltage: %f", frequency, reading);
    }
    hpt_close_all ();
    return (0);
}
```

Figure 5-10. The Final Program in C

Step 7: Save Your Program

Now that you have generated the initialization and instrument-control code in HP ITG, you should save your program and exit HP ITG in preparation for further program development.

1. Click on `File` on the Editor menu bar.
2. Click on `Save`.
3. Click on the bidirectional arrows in the Editor window's upper right corner.
4. Exit HP ITG.
    a. Click on `File` on the System menu bar.
    b. Click on `Exit To DOS` to return to MS-DOS.

Step 8: Start Quick Environment

You are now ready to start the Quick environment for the language you are using so you can debug the program generated in HP ITG. This step explains how to use batch files supplied with HP ITG to start QuickBASIC or QuickC. If you are using Microsoft C, continue with Step 9.

Since you created a program using the Log HP ITG Calls mode, you will need to include the HP ITG Quick Library to use in QuickBASIC, or create a make file to use in QuickC. The batch files let you start either environment with the proper options so you can continue program development.

To start the QuickC environment, enter the batch file name with the program file name at the MS-DOS command line:

```
qcstart hpt_log.c
```

To start the QuickBASIC environment, enter the batch file name with the program file name at the MS-DOS command line:

```
qbstart hpt_log.bas
```

After starting either environment, refer to the product's manual for information about its operation.

Step 9: Create an Executable Program

When you have debugged your program and it is running correctly in the Quick environment, you can create an executable that you can run directly from the MS-DOS command line. If you are using Microsoft C, and have edited your program, you are ready to create an executable.

This step explains how to use a batch file supplied with HP ITG to create an executable. The batch file compiles your program into object code, then automatically links the object code against the correct HP ITG standalone library to create the executable, HPT_LOG.EXE.

To create the executable from your Microsoft C program, enter the batch file name with the program file name at the MS-DOS command line:

```
cmake hpt_log
```

To create the executable from your QuickC program, enter the batch file name with the program file name at the MS-DOS command line:

```
qcmake hpt_log
```

To create the executable from your QuickBASIC program, enter the batch file name with the program file name at the MS-DOS command line:

```
qbmake hpt_log
```

6

Creating and Using a Soft Test System

Overview

To develop test software, HP ITG lets you to create a soft test system onscreen that matches your actual test system. Your interactions with panels on the soft test system generate software that runs the actual test system. When you save a soft test system, HP ITG saves the panels, their instrument configurations, and instrument states in a workfile.

By using the HP ITG Instrument Drivers, you can create many workfiles to support different test systems. Each system would consist of the instruments for which drivers exist. You can add instruments to the system as new drivers become available, and you can delete instruments from the system.

Creating a New Soft Test System

At Startup

When you run HP ITG the first time, the Editor window is displayed at the bottom of the work area, and the icon area at the right is blank. You can immediately start building a test system by adding the instruments and applications you want to use.

During a Work Session

If you have been working with one test system and want to remove it to create another one, follow these instructions:

1. Click on `File` on the System menu bar.
2. Click on `New`.

This clears the work area and the icon area. Now you can start building a new test system by adding the instruments and applications you need.

You can also create a new test system by modifying an existing one, then saving the new test system under a new name. For example, assume the new test system you are planning includes instruments or application panels already contained in an existing test system. To remove an unwanted instrument panel:

1. Click on the panel's menu box (upper left corner).
2. Click on `Close` to remove the panel.
3. Begin adding new panels required for the new system by clicking on `Instruments ...` on the System menu bar.
4. To save the modified soft test system, click on `File` on the System menu bar, then click on `Save Workfile As ...` and enter the new workfile file name.

Opening an Existing Test System

If you or someone else have previously created and saved a workfile of a soft test system you want to use, you can open the existing workfile. HP ITG will ask you to save changes made to the current workfile before loading the next one.

1. Click on `File` on the System menu bar.
2. Click on `Open Workfile` ...
3. To change directories, click on `[..]`, then `Open`. Select the directory name.
4. Click on the correct workfile name or type it in.
5. Click on `Open`.

Note

You can open an existing workfile each time you start HP ITG by specifying the workfile's file name in the MS-DOS command line:

`hpitg` *workfile*

If you do not specify *workfile*, HP ITG automatically opens `DEFAULT.WF` if that workfile exists.

Saving a Soft Test System

For the First Time

If you built a new soft test system and you are saving it for the first time, you need to save it as a particular workfile:

1. Click on `File` on the System menu bar.
2. Click on `Save Workfile As` ...

3. Type in a new workfile name. To create the `DEFAULT.WF` workfile, type in **`default`**.
4. Click on `OK`.

When you type in a workfile name, HP ITG will add the `.WF` extension automatically. Be sure that the file name you enter meets MS-DOS file naming rules. The `.WF` file name extension is important because HP ITG displays only workfile names ending with `.WF` when you select the `Open` and `Save Workfile As ...` commands in the system `File` menu.

Resaving a Soft Test System

If you are resaving a soft test system, HP ITG saves the workfile under the same name you used to open the workfile.

1. Click on `File` on the System menu bar.
2. Click on `Save Workfile`.

Using the Default Workfile

If a `DEFAULT.WF` workfile exists, HP ITG opens it automatically when you type **`hpitg`** at the MS-DOS command line and press (Enter) to start HP ITG. If you want to work with only one soft test system, you may want to save it as the default workfile, `DEFAULT.WF`, instead of giving it a unique name.

Caution

If you have generated code using HP ITG's Log HP ITG Calls mode, you must save the soft test system used to generate this code or your program will not work.

Adding and Configuring Instruments

To add an instrument to your soft test system, you need to load the correct driver file for that instrument. The file names for the drivers shipped with HP ITG are listed in the online Help system under `Latest Information ...` You will notice both driver and help file names. These files must be kept in the same directory for the instrument help information to be available.

1. Click on `Help` on the System menu bar.
2. Click on `Latest Information ...`
3. Click on the topic `HP ITG Instrument Drivers`, then click on `OK`.
4. Scroll through the list using the scroll bar to the right to locate the instrument you want to add.
5. Note the file name for that instrument (file name extension is `.ID`).
6. Quit the Help window and proceed to add and configure the instrument.

You can add up to 254 instruments to your test system if your computer contains sufficient additional memory. The following instructions explain how to display the instrument selection list box so you can select and add instruments.

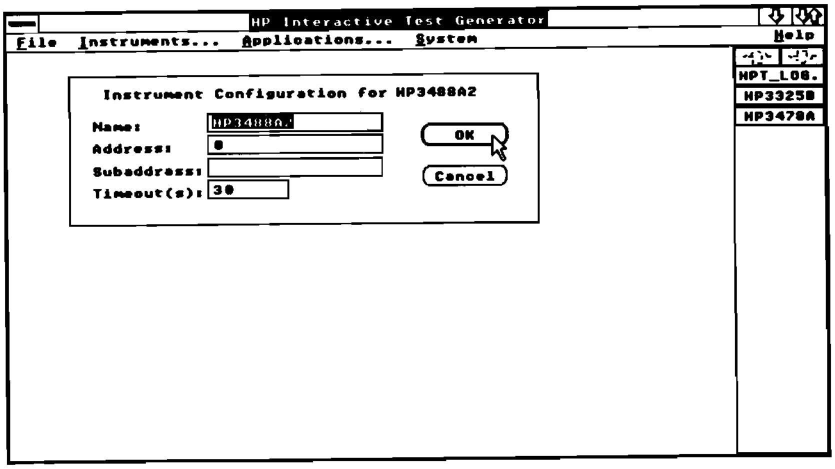
1. Click on `Instruments ...` on the System menu bar.
2. Click on the instrument of your choice or type in its driver file name.
3. Click on `OK`.

Though HP ITG allows you to add up to 254 instrument/application panels in one soft test system, the number of panels and icons that HP ITG can display at any one time depends on the size of your screen.

You can reduce panels to icons by clicking on the arrow in the top right corner of each panel. If there are more icons in the icon area than HP ITG can display at one time, click on the arrows at the top of the icon area to scroll through the list.

Configuring the Instrument

After you have selected an instrument, HP ITG displays the instrument configuration dialog box shown in the following figure. This configuration dialog box is also available from the panel menu, which allows you to make changes after you have added the instrument to your soft test system.



Instrument Configuration for HP3488A2

Name: HP3488A2

Address: 9

Subaddress:

Timeout(s): 30

Figure 6-1. Configuration Dialog Box for an Instrument

The configuration dialog box contains four fields: name, address, subaddress, and **timeout(s)**. To edit any field, press the Tab or click on the character in the field where you want to place the cursor and begin editing.

Name

In the `Name` field, enter the name you want to use to refer to this instrument/address combination.

- The default name is the driver file name with the `.ID` extension removed.

HP ITG uses `Name` to bind a driver to an instrument at a specific bus address in the current test system. This allows you to use the same driver to run another unit of the same instrument model at a different address. For example, if you have two HP 3478A multimeters in your test system, you must select `HP3478A.ID` twice and provide two different addresses and names.

Address

If the instrument is connected to the interface board, enter its address in the `Address` field. If the instrument is not connected to the interface board, leave `Address` set to 0.

- The default address is 0.

Note If you enter an address other than 0 but the instrument is not connected to the interface board or set to that address, HP ITG will generate an error. Click on `OK` to continue. HP ITG will reset the address to 0.

Subaddress

In the box labeled `Subaddress`, enter a subaddress if the instrument is a module that requires a subaddress.

- Subaddresses are usually only necessary for card-cage instruments.

HP ITG ignores subaddresses that are specified but not needed.

Timeout(s)

In the box labeled `Timeout(s)`, enter the amount of time in seconds you want HP ITG to wait for a response from an instrument when its panel is in Live mode. This value is also valid in the run-time environment. You can set Live mode from the instrument panel menu under `Modes` ... Live mode lets you control the instrument directly from the HP ITG development environment.

- The default time is 30 seconds.

Using HPIB.ID

If you want to include an instrument in your test system but an instrument driver does not exist for it, you can still control it by using the driver named `HPIB.ID`. This driver lets you control simpler instruments without having to write your own instrument driver. To add `HPIB.ID` to your work area:

1. Click on `Instruments` ... on the System menu bar.
2. Click on `HPIB.ID`, then click on `Open`.
3. Complete the instrument configuration box for the instrument you are using, then click on `OK`.

To use `HPIB.ID`, you need to know the instrument's HP-IB codes that control its features over the interface bus. Otherwise, use the panel as you would any other soft panel to generate your instrument-control code. See chapter 7, "Soft Panel Operation," and chapter 8, "Controlling an Instrument," for information.

Deleting Instruments

Caution

Delete an instrument panel from the soft test system only if you are sure you do not want that instrument panel. This deletes all stored states for that panel and removes the panel from the soft test system.

You can delete an instrument from your soft test system:

1. Click on the instrument panel's menu box (top left corner of the instrument panel).
2. Click on `Close`.

Adding Applications

You can add an **application** panel to your test system:

1. Click on `Applications` on the System menu bar.
2. Click on the application of your choice.

HP ITG lists all of the installed application drivers in the current directory. Applications must end with a `.AD` file name extension to be listed.

An application is anything you can control through an HP ITG soft panel that is not a standalone instrument. An application is a program written in the HP ITG driver style. A program written in this way provides a panel through which you can interact with the program just as you use an instrument panel to interact with instruments.

An application may perform simulation, modeling, or data analysis. Or, it may combine features of one or more instruments into a virtual instrument.

Printing the Display

To print a copy of the HP ITG display, be sure your printer is connected to the correct interface bus, turned on, and properly configured.

1. Click on `File` on the System menu bar.
2. Click on `Print` ...
3. Complete the dialog box, then click on `OK`.

If an error appears when you attempt to print, you may need to configure your printer. Printer configuration is a two-step process: first, the printer driver for your specific printer must be installed; then the printer must be set up. If you identified your printer during the HP ITG installation process, you should not have to install a printer driver, but you probably will need to set up the printer.

Configuring Your Printer

If you are adding a new printer, you will need to install the printer driver from the HP ITG distribution disks. Run HP ITG's Setup program again to install the printer driver. Be sure to use the same options and to identify the new printer, then quit Setup when prompted to identify the directories. After installing the new printer driver, you are ready to set up the printer.

1. Run HP ITG.
2. Click on `System` on the System menu bar.
3. Click on `Configure`.
4. When `Control Panel` appears, click on `Setup`, then click on `Connections` ...
5. Click on the printer model and the connection, then click on `OK`.

Note

Printers using the HP-IB interface must be connected to LPT1.

6. Click on `Setup` again, then click on `Printer` ...
7. Click on the printer model, then click on `OK`.
8. Complete the printer information, then click on `OK`.
9. To quit `Control Panel`, click on the menu control box in the panel's upper left corner, then on `Close`.

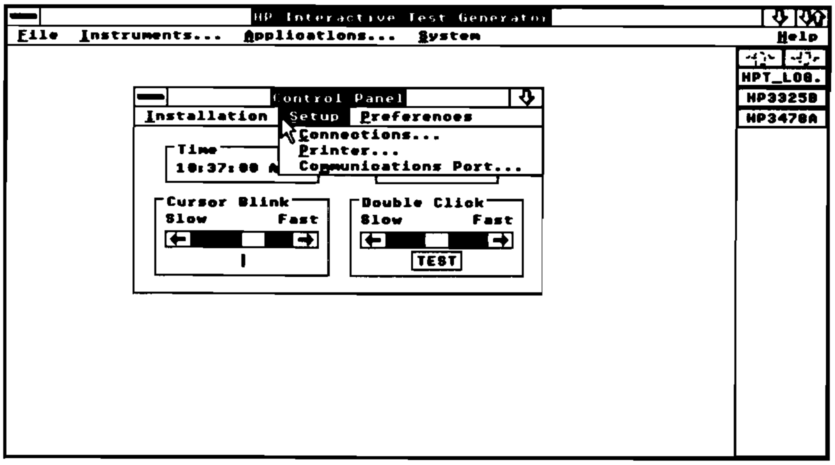


Figure 6-2. The Printer Configuration Control Panel

7

Soft Panel Operation

Overview

For each instrument or application, HP ITG displays a panel on the screen through which you can control an instrument or run an application program. The driver is a logical representation of the instrument. As such, it contains information that determines the panel's size, layout, operation, and initial state, among other things. Each driver supplied by Hewlett-Packard is documented in HP ITG's online Help system.

Panel Size

Panels provided by Hewlett-Packard come in three sizes:

- Quarter.
- Half.
- Full.

Panel Layout and Style

The figure on the following page shows a typical HP ITG panel.

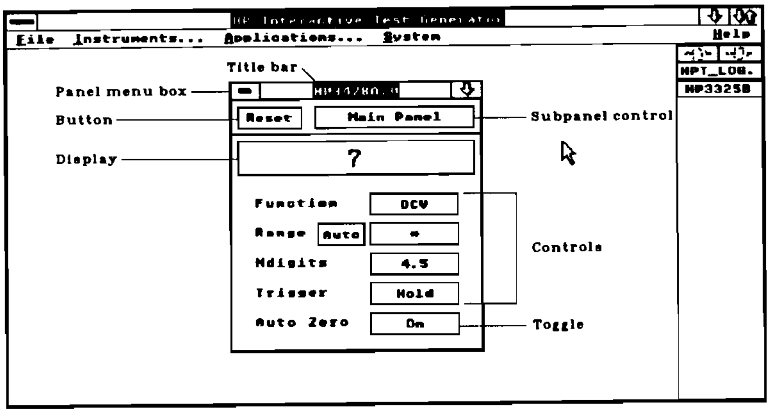


Figure 7-1. An Instrument Panel

All panels consist of one or more of the following elements: controls, buttons, subpanels, and displays.

Controls

Clicking on a control lets you change the panel settings in the following ways:

- Enter a numeric value.
- Select a value from a list.
- Toggle between two values—the current value is displayed in the control box.

Some controls provide a Query button. The Query button is part of the dialog box displayed when you click on the panel menu box (see following figure). The Query lets you request the instrument to update the panel with the current value of the instrument's control setting, if the panel is in Live mode. See chapter 8 for more information about Live mode.

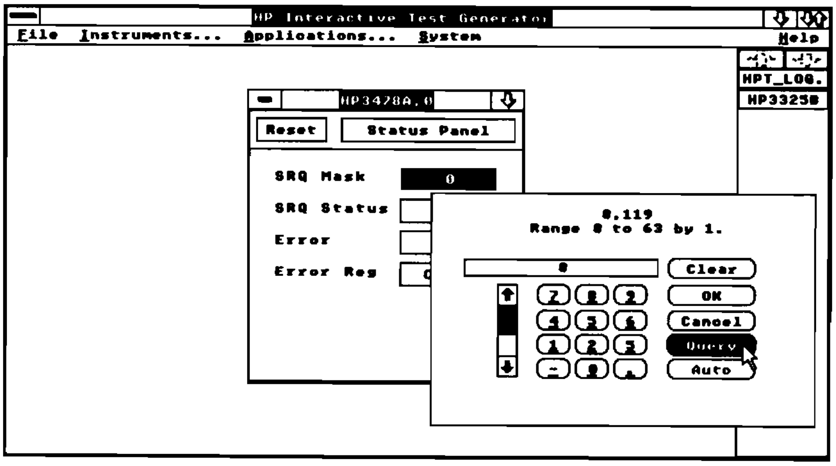


Figure 7-2. Click on the Query Button to Update the Panel

Note Some controls let you type in a value. Be sure to press (Enter) or click on `OK` after typing in the value.

Buttons

Clicking on a button causes an action. For example, a `Volt Reset` button resets the voltage control to a default value.

Subpanels

Many instruments contain more controls and buttons than can be displayed on one HP ITG panel. Most panels provided by Hewlett-Packard have a control in the top right corner that lets you select from a list of available **subpanels** when you click on it. The control displays the name of the currently selected subpanel.

Occasionally, several parameters must be set when using a particular feature, such as advanced triggering on the HP 3458A. In these cases, clicking on the control displays a dialog box that you must complete.

Displays

A display is either a numeric/string readout or an XY display. Clicking on a display makes a measurement and displays a reading.

Note An HP ITG XY display is not intended to duplicate the instrument's display. Rather, it provides a way for you to generate the code needed to make a measurement and send the measured data to the computer for storage and analysis.

Automatic Update. Some instrument drivers allow displays to be continuously updated after clicking on them. You can turn on this mode from the panel control menu:

1. Click on the panel menu box, then click on `Modes` ...
2. If the driver supports the mode, `Automatic Update` appears in black characters.
3. Click on the box next to `Automatic Update` so an `X` appears.
4. Click on `OK`.

This mode must also be turned on for the whole soft test system:

1. Click on `System` on the System menu bar.
2. If `Automatic Update` has a check mark beside it, the mode is on.
3. If no check mark exists, click on `Automatic Update` to turn the mode on.

To start the continuous updating for a panel, click on its display. To stop the updating, click on the display again. By controlling the mode from the System menu bar and from the panel menu, you can turn on Automatic

Update for individual instruments or for whole soft test systems.

Reset Button

Panels provided by Hewlett-Packard include a `Reset` button that returns the panel, and the instrument (if connected), to a default state defined in the driver. This default state may or may not match the instrument's power-on state.

Note

HP ITG creates a state called HPTINITIALSTATE when you load the panel into a soft test system. This state consists of the initial values of the instrument's functions as defined in the driver, and has the same effect as the panel's `Reset` button. You can recall HPTINITIALSTATE using the panel menu's `Recall State ...` command. You can overwrite HPTINITIALSTATE by changing the panel values, then clicking on `Store State ...` in the panel menu.

Panels and Icons

HP ITG can store the panels you are not currently using as **icons.** The icons are displayed along the right side of the HP ITG work area as shown in the figure on the following page.

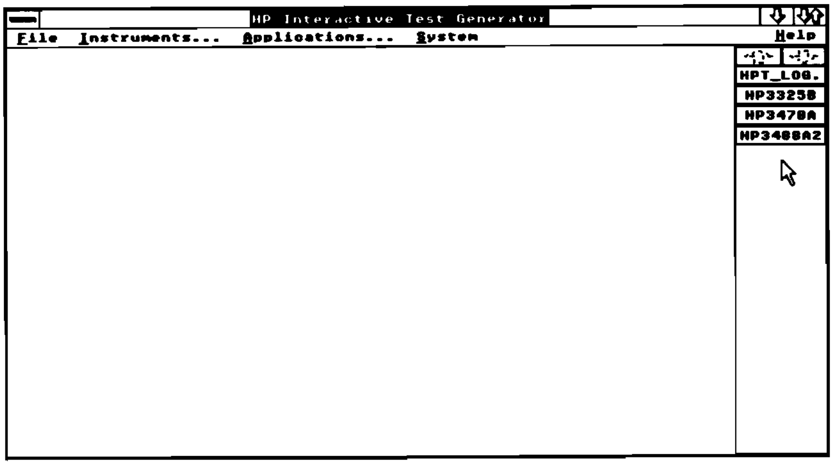


Figure 7-3. Panels Stored as Icons

Expanding an Icon

You can expand an instrument icon so that you can work with the panel by clicking on the icon.

Reducing a Panel

You can reduce a panel to an icon by clicking on the arrow in the top right corner of the panel.

Moving a Panel

You can move a panel around the work area:

1. Click and hold on the panel name on the panel's title bar.
2. Drag the panel to the new location, then release the mouse button.

Note

When you move a panel, it may be overlaid by another panel that you expand from an icon. To bring a hidden panel into better view, you can click anywhere on the panel to bring it to the top of the stack.

Getting Help on a Panel

Each instrument driver supplied by Hewlett-Packard is documented in HP ITG's online Help system. The online Help system provides a quick reference guide to the instrument and its panel. You can access the information from the `Help` command on the System menu bar and from the instrument panel menu.

From the Panel

You can access the online reference through the panel menu box:

1. Click on the panel menu box (top left corner of the panel).
2. Click on `Help` ...
3. Click on the preferred topic.
4. Click on `OK` to display the information.

From the System Menu Bar

You can access the online reference through `Help` on the System menu bar:

1. Click on `Help` on the System menu bar.
2. Click on `Instrument Help` ...
3. Click on the instrument file name from which you want information.
4. Click on `OK`.
5. Click on the preferred topic.
6. Click on `OK` to display the information.

8 Controlling an Instrument

Overview

One of the most powerful features of HP ITG is that it lets you verify your measurement procedure as you generate program code. With HP ITG, you can control an instrument directly as you adjust its soft panel while generating code. Then, you can verify every step of the measurement procedure as the code is generated.

Caution

You should not adjust the instrument's front panel manually while using HP ITG to control an instrument. When you adjust the instrument manually, HP ITG cannot detect the adjustment.

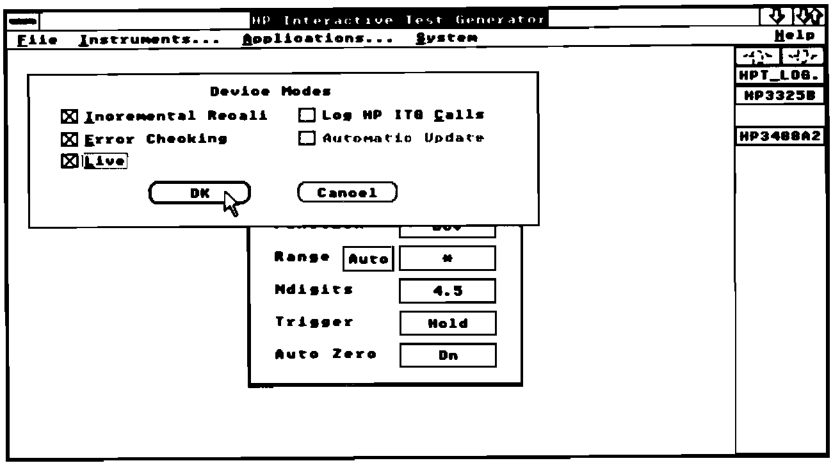
Making Adjustments

In a Panel

You can control an instrument through its panel:

1. Make sure the instrument is properly connected to the computer's interface, and that its address matches the one you assigned to the panel when you added it to your soft test system.

2. Turn on Live mode.
   a. Click on the panel menu box.
   b. Click on `Modes ...`
   c. Click on the box next to `Live` so an `X` appears in the box (figure 8-1).
   d. Click on `OK`.
3. Verify that everything is connected properly.
   a. Click on a panel control and either enter a value or select a value from the list box.
   b. Click on `OK`.
   c. Observe the instrument to confirm that the change you made on the panel also changed the instrument.
4. Click on the panel display to take a reading (if appropriate for the panel).


HP Interactive Test Generator
File Instruments... Applications... System Help
HPT_LOG.
HP3325B
HP3488A2
OK
Cancel

Device Modes

☒ Incremental Recall ☐ Log HP ITG Calls
☒ Error Checking ☐ Automatic Update
☒ Live

Range Auto *
Ndigits 4.5
Trigger Hold
Auto Zero On

Figure 8-1. Controlling an Instrument with Live Mode

In a Program

In a program containing calls to HP ITG subprograms, Live mode is automatically on. You can, however, insert a call to `hpt_livemode` to turn off Live mode for a given instrument. This is useful when you are developing other parts of the program and don't want to execute the instrument control section. For a complete description of `hpt_livemode`, see chapter 14 if you are using QuickBASIC, or chapter 15 if you are using C.

Checking for Errors

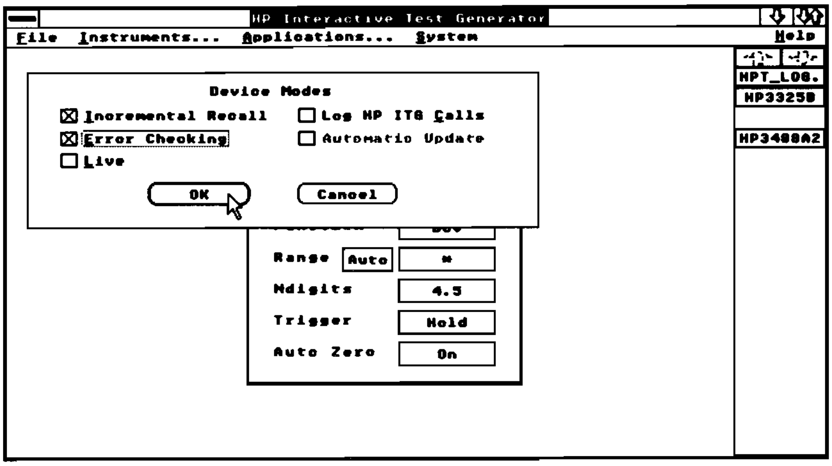
If an instrument has error-reporting abilities, and its driver supports that feature, you can use HP ITG's Error Checking mode for the instrument.

When the Error Checking mode is on, HP ITG checks the instrument for errors whenever you set a control or make a measurement. Many HP panels provide a display, typically labeled `Error`, in which HP ITG displays the number of the error that was detected. See `Instrument Help` ... in HP ITG's online Help system or the instrument's operating manual for error message information.

Controlling Instruments Directly

You can use the Error Checking mode while controlling an instrument directly with HP ITG (Live mode is on):

1. Click on the panel menu box.
2. Click on `Modes` ...
3. Click on the box next to `Error Checking` so an `X` appears in the box.
4. Click on `OK`.



Device Modes

☒ Incremental Recall ☐ Log HP ITG Calls
☒ Error Checking ☐ Automatic Update
☐ Live

Range Auto *
Ndigits 4.5
Trigger Hold
Auto Zero On

Figure 8-2. Using the Error Checking Mode

In a Program

To turn on Error Checking for an instrument within your program, insert a call to `hpt_errorcheck` into the program. Be selective about when you use the Error Checking mode in a program since it may slow down program execution. For a complete description of `hpt_errorcheck`, see chapter 14 if you are using QuickBASIC, or chapter 15 if you are using C.

Changing Configuration Information

You can change an instrument panel's name, address, subaddress, or timeout after adding the panel to the work area:

1. Click on the panel menu box.
2. Click on `Config` ...
3. Edit the fields in the instrument configuration box as necessary.

For a complete description about configuring instruments, see chapter 6 in this handbook.

9

Creating and Using Instrument States

Overview

After adjusting a panel as needed for a step in a measurement procedure, HP ITG lets you store that setup as an instrument state. This lets you use the same setup repeatedly without having to reset all of the controls.

When created, states are maintained in memory while the instrument driver is open in the HP ITG work area. The states are retained by the soft test system when you save the workfile. You can also save states for export to other soft test systems.

If many states have been saved in a workfile, you can recall one state after another in your program. This lets you change the instrument settings quickly. To optimize test throughput when recalling states, HP ITG provides **incremental state programming.** This lets HP ITG track the current state and send the minimum set of commands to put the instrument into the next recalled state.

Creating States

Each state is given a name when created. State names are unique to a given instrument. Many states can share the same name as long as each state belongs to a different instrument. For example, HPTINITIALSTATE exists for all instrument drivers. This lets you give the same state name to each instrument involved in a given measurement procedure.

Storing a State

You can store an instrument state after changing the panel setup:

1. Click on the instrument panel menu box.
2. Click on `Store State` ...
3. Type in a name or select from the list of names if you want to overwrite an existing state.
4. Click on `Store`.

Note

A name must begin with a letter; only letters, numbers, and underscores (_) are allowable characters for the rest of the name.

5. Be sure to save your soft test system before exiting HP ITG if you have stored any new states.

Caution

Closing a panel deletes all stored states associated with that panel if you then save the workfile. You should consider creating a **state library** as a permanent backup. The state library is described later in this chapter.

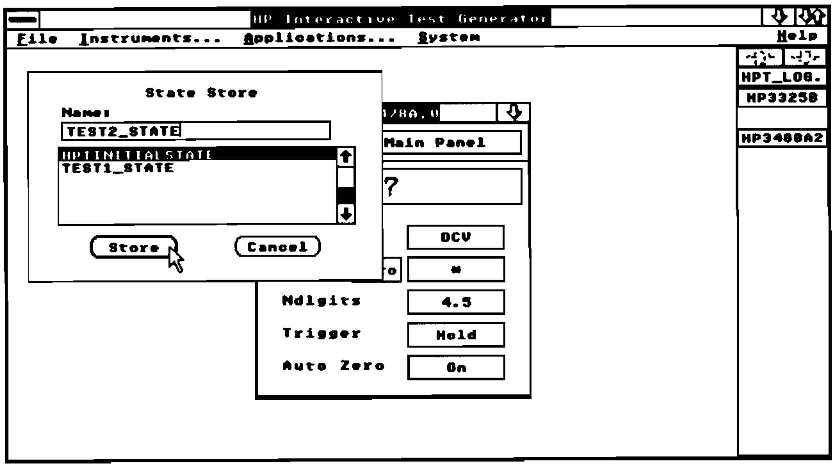
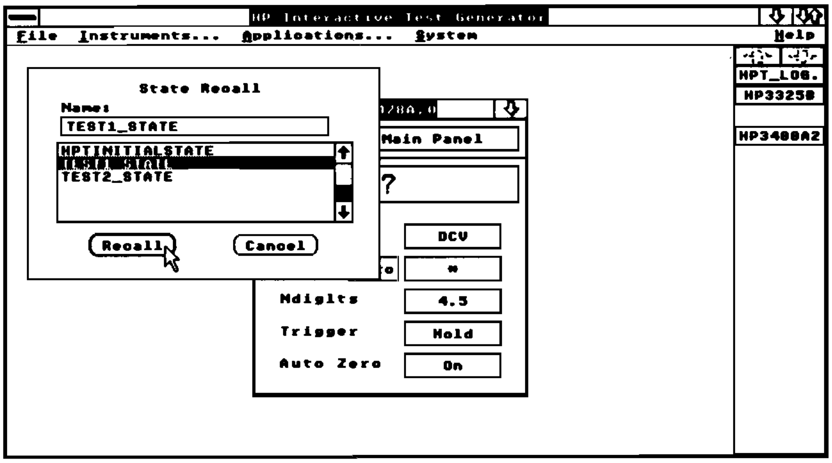


Figure 9-1. State Store Dialog Box

Recalling a State

You can recall a stored state from a panel:

1. Click on the panel menu box.
2. Click on `Recall State ...`
3. Select from the list of names.
4. Click on `Recall`.



State Recall

Name: TEST1_STATE

HPTINITIALSTATE
TEST1_STATE
TEST2_STATE

DCV

Ndigits 4.5

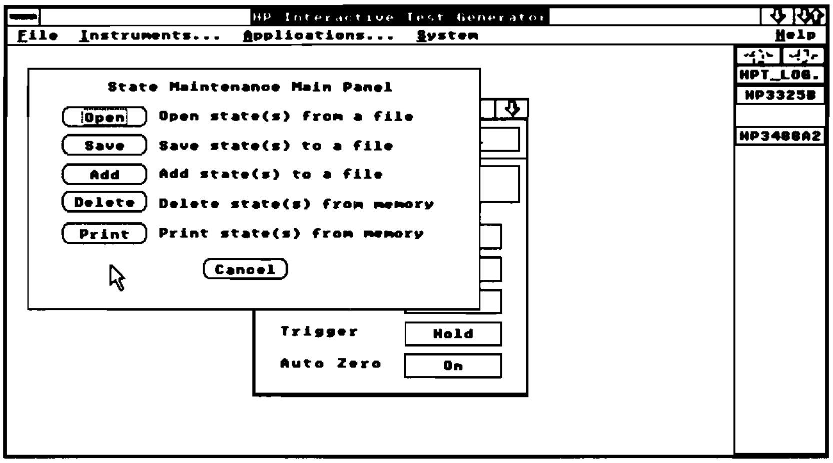
Trigger Hold

Auto Zero On

Figure 9-2. State Recall Dialog Box

Maintaining States

Once you have created instrument states, you can use the State Maintenance dialog box to increase each state's usefulness in various soft test systems.



State Maintenance Main Panel

Open — Open state(s) from a file

Save — Save state(s) to a file

Add — Add state(s) to a file

Delete — Delete state(s) from memory

Print — Print state(s) from memory

Trigger Hold

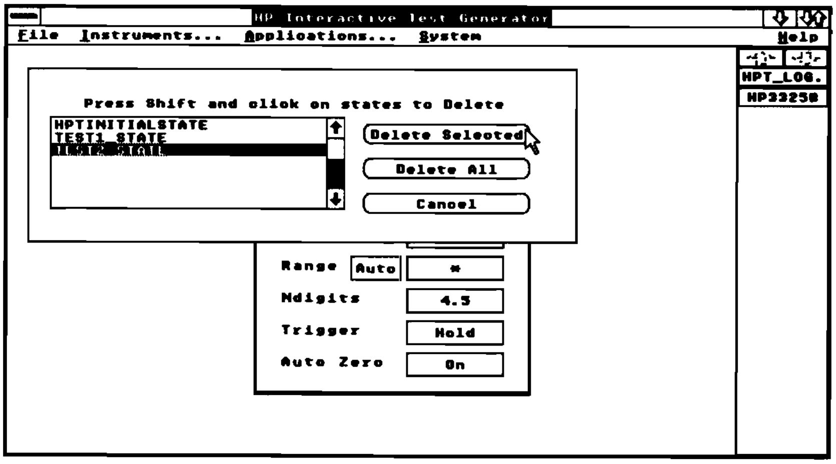
Auto Zero On

Figure 9-3. State Maintenance Dialog Box

Deleting a State

You can delete one or more stored states no longer needed for an instrument:

1. Click on the panel menu box.
2. Click on `State Maint ...`
3. Click on `Delete`.
4. Delete state(s).
   a. To delete selected states, press (Shift) and click on each state to be deleted, then click on `Delete Selected`.
   b. To delete all states, click on `Delete All`.



Press Shift and click on states to Delete

```
HPTINITIALSTATE
TEST1 STATE
TEST2 STATE
```

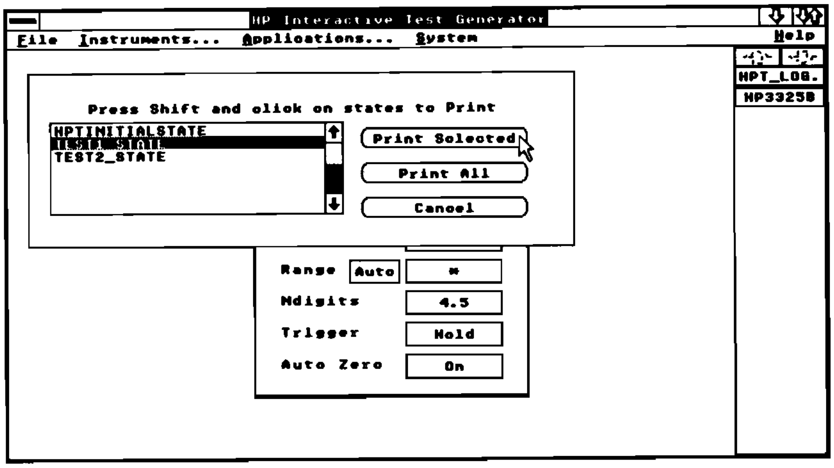
```
Range      Auto   *
Ndigits           4.5
Trigger           Hold
Auto Zero         On
```

Figure 9-4. The Delete State Dialog Box

Printing the Contents of a State

You can print the values for panel controls that HP ITG stores as part of an instrument state:

1. Click on the panel menu box.
2. Click on `State Maint ...`
3. Click on `Print`.
4. Print state(s).
   a. To print selected states, press (Shift) and click on each state to be printed; then click on `Print Selected`.
   b. To print all states, click on `Print All`.



```
Press Shift and click on states to Print
HPTINITIALSTATE
TEST1_STATE
TEST2_STATE

Range      Auto   *
Ndigits           4.5
Trigger           Hold
Auto Zero         On
```

Figure 9-5. The Print State Dialog Box

Making a State Library

Saving States to a File

You can create a state library file that can be exported to another soft test system:

1. Click on the panel menu box.
2. Click on `State Maint ...`
3. Click on `Save`.
4. Select the state(s) to save.
    a. To save selected states, press (Shift) and click on each state to be saved, then click on `Save Selected`.
    b. To save all states, click on `Save All`.
5. Type in the file name for the state file.
6. Click on `OK`.

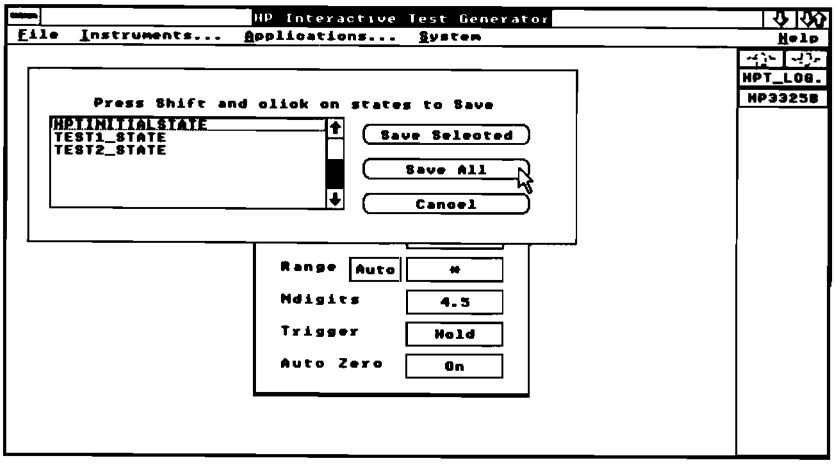
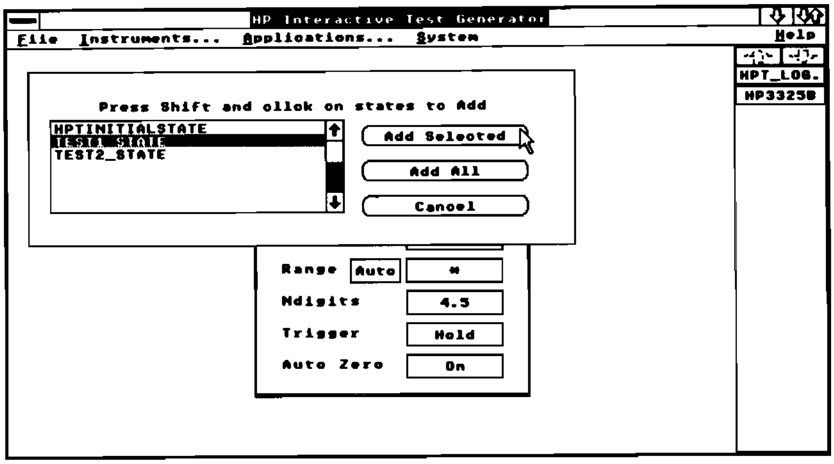


Figure 9-6. The Save State Dialog Box

Adding States to a File

You can add a state to an existing state library file:

1. Click on the panel menu box.
2. Click on `State Maint` ...
3. Click on `Add`.
4. Select state(s) to add.
   a. To add selected states, press (Shift) and click on each state to be added; then click on `Add Selected`.
   b. To add all states, click on `Add All`.
5. Type in the file name for the state file you are adding the state(s) to.
6. Click on `OK`.



Press Shift and click on states to Add

HPTINITIALSTATE
TEST1_STATE
TEST2_STATE

Range Auto *
Ndigits 4.5
Trigger Hold
Auto Zero On

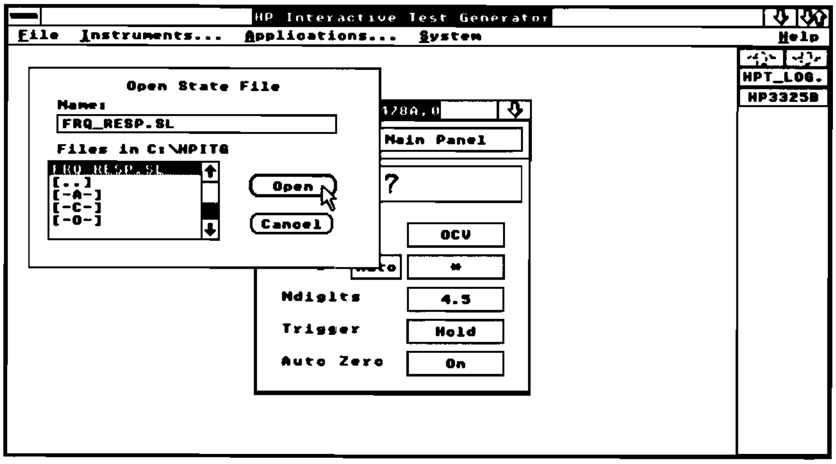
Figure 9-7. The Add State Dialog Box

Importing a State File

You can import a state file from another soft test system:

1. Click on the panel menu box.
2. Click on `State Maint` ...
3. Click on `Open`.
4. Click on the name of the state file you want to open, or type the file name into the name field.
5. Click on `Open`.

When you open a state file, you now have access to all states stored in that file.


HP Interactive Test Generator
File Instruments... Applications... System Help
Open State File
Open
Cancel
Main Panel
HPT_LOG.
HP3325B

Name: FRQ_RESP.SL

Files in C:\HPITG

```
FRQ_RESP.SL
[..]
[-A-]
[-C-]
[-D-]
```

	OCV
	*
Ndigits	4.5
Trigger	Hold
Auto Zero	On

Figure 9-8. The Open State List Box

Using States

To use the full power of HP ITG, you should create a state for as many instrument setups as you need to complete a measurement procedure. Not only does this streamline the process of stepping through the measurement procedure when you generate the code, but it greatly simplifies the code itself.

Figures 9-9 and 9-11 are programs generated by adjusting each function on each instrument. Figures 9-10 and 9-12 are programs generated by recalling states and adjusting controls only for looping purposes. Each figure presents the examples in the programming languages supported by HP ITG.

```
hpt_set_str (hp3325b, "FUNCTION", "SINE");
hpt_set (hp3325b, "FREQUENCY", (double) 10000);
hpt_set (hp3325b, "AMPLITUDE", (double) 1);
hpt_set_str (hp3325b, "AUNITS", "RMS");
hpt_set_str (hp3478a, "FUNCTION", "ACV");
hpt_set_str (hp3478a, "ARANGE", "ON");
hpt_set (hp3478a, "NDIG", 3.5);
hpt_get (hp3478a, "READING", &double_value);
```

Figure 9-9. In C, Each Function on Each Instrument is Set Individually

```
hpt_recall (hp3325b, "STATE1");
hpt_recall (hp3478a, "STATE1");
hpt_get (hp3478a, "READING", &double_value);
```

Figure 9-10.
In C, Function Settings are Stored as Instrument States and Then Recalled to Form a Procedure

```
CALL hptsetstr(hp3325b%, "FUNCTION", "SINE")
CALL hptset(hp3325b%, "FREQUENCY", 10000)
CALL hptset(hp3325b%, "AMPLITUDE", 1)
CALL hptsetstr(hp3325b%, "AUNITS", "RMS")
CALL hptsetstr(hp3478a%, "FUNCTION", "ACV")
CALL hptsetstr(hp3478a%, "ARANGE", "ON")
CALL hptset(hp3478a%, "NDIG", 3.5)
CALL hptget(hp3478a%, "READING", reading#)
```

Figure 9-11.
In QuickBASIC, Each Function on Each Instrument is Set Individually

```
CALL hptrecall(hp3325b%, "STATE1")
CALL hptrecall(hp3478a%, "STATE1")
CALL hptget(hp3478a%, "READING", reading#)
```

Figure 9-12.
In QuickBASIC, Function Settings are Stored as Instrument States and Then Recalled to Form a Procedure

Incremental State Programming

When HP ITG controls an instrument, it tracks the current instrument state and sends the minimum set of commands needed to put the instrument into the next recalled state. This is called incremental state programming. This improves test throughput by eliminating the time spent sending unnecessary commands over the HP-IB, as well as the time required for an instrument to interpret and respond to the commands.

When a panel's Incremental Recall mode is on, HP ITG performs incremental state programming on that instrument. HP ITG automatically turns on a panel's

Incremental Recall mode when you add it to your soft test system. When you want HP ITG to send all of the commands associated with a recalled state, you need to turn off that panel's Incremental Recall mode.

Controlling Instruments Directly

You can turn Incremental Recall mode on or off to control instruments directly as needed:

1. Click on the panel menu box.
2. Click on `Modes` ...
3. Click on the box beside `Incremental Recall`. An `X` in the box means the mode is on.

In a Program

To turn off Incremental Recall mode for an instrument within your program, insert a call to `hpt_incremental` into the program. For a complete description of `hpt_incremental`, see chapter 14 if you are using QuickBASIC, or chapter 15 if you are using C.

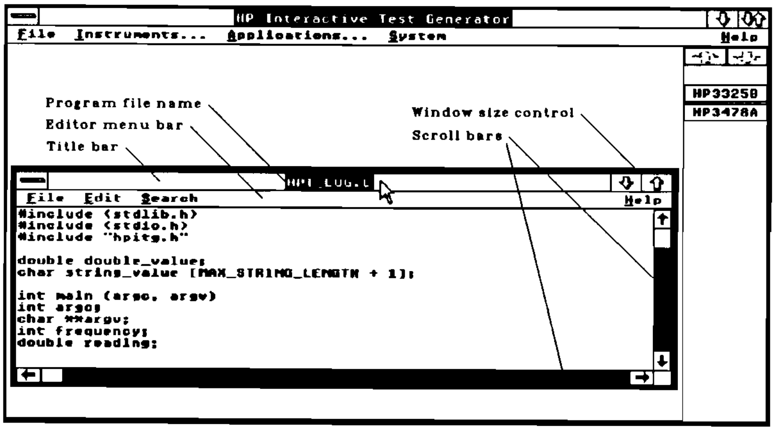
10

The HP ITG Editor

Overview

HP ITG provides an Editor window that lets you view and edit the code HP ITG generates. This chapter explains how to use the HP ITG Editor to generate your program code.

HP ITG provides one Editor window in every soft test system. You can add additional Editor windows as needed. An additional editor is useful for comparing files, for using one file as a reference while you create another, or breaking large programs into smaller modules.



```
#include <stdlib.h>
#include <stdio.h>
#include "hpitg.h"

double double_value;
char string_value [MAX_STRING_LENGTH + 1];

int main (argc, argv)
int argc;
char **argv;
int frequency;
double reading;
```

Figure 10-1. The Editor Window

Adjusting the Editor Window

You can adjust the Editor window's size and placement in the HP ITG work area. The adjustments can be made by clicking on various locations around the Editor window or using the the menu available from the Editor menu box at the left end of the window's title bar. The following instructions explain how to adjust the Editor window by clicking on various locations.

Storing the Editor as an Icon

You can reduce the Editor window to an icon if you need the extra room to display more panels.

- Click on the down arrow in the top right corner of the Editor title bar.

If you reduce the Editor window to an icon, HP ITG can still generate code in the Editor. You can view the code after expanding the icon back to a window.

Expanding the Editor Icon

You can expand the Editor window from an icon. This restores the window to the same location in the work area before it was reduced to an icon.

- Click on the Editor icon to expand the Editor window.

Expanding the Editor to Full Size

You can expand the Editor window to full size.

- Click on the up arrow in the top right corner of the Editor title bar.

Adjusting the Editor Window Size

When the Editor window is visible in the work area, you can adjust its size to more convenient proportions.

1. Place the mouse cursor on any Editor window border so a bidirectional arrow appears.
2. Click and hold on the border, dragging it in the arrow's direction.
3. Release the mouse button when the border is in the preferred location.

Moving the Window

You can move the Editor window to a more convenient location:

1. Click and hold on the Editor window title bar.
2. Drag the window to the preferred location.
3. Release the mouse button.

Using the File Commands

The Editor menu bar contains the `File` command that lets you open, save, and print your program files. See appendix A in this handbook, "Menus Index," for a description of each command.

Note

The default program name is `HPT_LOG` when the Editor window first appears. The `.BAS` extension is added if you are using the QuickBASIC language or `.C` for the C language.

Starting a New File

You can clear the Editor window to start a new program file:

1. Click on `File` on the Editor menu bar.
2. Click on `New`.

This deletes the contents of the HP ITG Editor window, and changes the file name to `untitled`.

Opening an Existing File

You can open an existing program to edit in the Editor window:

1. Click on `File` on the Editor menu bar.
2. Click on `Open ...`
3. To change directories, click on `[..]`, then `Open`. Select the directory name.

4. Select from the list of file names or enter the file name in the **Name:** field.
5. Click on Open.

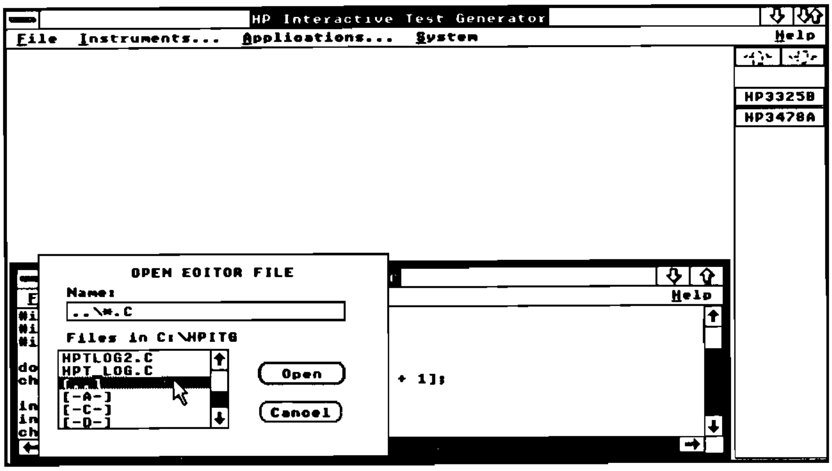


Figure 10-2. The Editor File List Box

Clearing the Editor and Starting Over

You can clear the contents of the Editor window to revise the current file:

1. Click on Edit on the Editor menu bar.
2. Click on Select All.
3. Click on Edit, then click on Clear.

The Select All/Clear commands work differently than New. Clear causes HP ITG to retain the current file name. For example, if you Open a file named FRQ_RESP, clear the screen, then select Save, the file named FRQ_RESP becomes empty. However, if you open the file and select New, the Editor window is cleared, and (untitled) appears in the title bar. The file FRQ_RESP remains unchanged.

Saving a File

There are two commands you can use to save your program, **Save** and **Save as ...** .

The Save Command

You can save your program under the current file name which is displayed on the Editor title bar:

1. Click on **File** on the Editor menu bar.
2. Click on **Save**.

The Save As ... Command

You can save your program under a file name other than the default file name, or the file name used to open it:

1. Click on **File** on the Editor menu bar.
2. Click on **Save As ...**
3. Type in the file name. Be sure to include the full path name if the program should be saved in a specific directory other than the current directory.
4. Click on **OK**. HP ITG saves your program as an ASCII file.

Note If you type an existing file name, HP ITG asks if you want to overwrite the existing file.

Printing a File

You can print the contents of the Editor window to a printer already configured for HP ITG. See chapter 6, "Creating and Using a Soft Test System," for information about printer configuration.

1. Click on **File** on the Editor menu bar.
2. Click on **Print**.

Using the Edit Commands

The edit commands let you enter and modify text in the Editor window.

Entering Text

To activate the Editor window so you can begin to enter text, click on the Editor's title bar. A flashing edit cursor appears in the window, indicating where text will appear when you begin typing or generating code. You can move the edit cursor's location by clicking on another character in the text or using the keyboard's edit keys.

Active Keys

All of the keyboard's character keys and the edit keys (Delete), (Home), (End), (Page Up), (Page Down), (Backspace), (Enter), and arrow keys are active in the Editor window. Since the Editor is always in the insert mode, the (Insert) key has no effect.

Scrolling

As you generate code and the Editor window fills with text, you can scroll through the text by using the scroll bars. The scroll bars are located at the right side and bottom of the Editor window. The right side scroll bar scrolls the text vertically, while the bottom scroll bar scrolls the text horizontally. For either scroll bar, use the scroll arrow and box to view different areas of text:

- Click on an arrow to move one line at a time.
- Click and hold on the scroll box, and drag it to a new location.
- Click in the scroll bar area outside the box to jump to a new area in the text.

The edit cursor remains in the same location of the text. To move it, click on a different character.

Generating Initialization Code

Every program that calls HP ITG subprograms must begin with standard initialization code. You can generate this standard code automatically from the Editor window. Be sure to save your workfile first.

1. Click on `Edit` on the Editor menu bar.
2. Click on `Generate Initialization Code`.

HP ITG generates initialization code for every panel that has Log HP ITG Calls mode turned on.

Note

Leave the edit cursor in the current position after generating the initialization code. HP ITG automatically positions the edit cursor at the line where you should continue code generation.

Undoing an Edit

You can cancel the last edit made if you undo the change immediately after the edit:

1. Click on `Edit` on the Editor menu bar.
2. Click on `Undo`.

Selecting and Replacing Text

You need to select specific text before you can replace it or use some of the Edit commands:

1. Click and hold on the character where the selection begins.
2. Drag the mouse cursor to the end of the selection, then release the button.
3. Replace the selected text by typing new text. When you type the first character, all of the selected text is deleted.

Using Select All

You can select all of the text in the Editor window at one time:

1. Click on `Edit`.
2. Click on `Select All`.

Deleting Text

You can delete text from the program:

1. Select text to be deleted.
2. Click on `Edit` on the Editor menu bar.
3. Click on `Clear`.

Moving Text

You can move portions of code to a new location:

1. Select text to be moved.
2. Click on `Edit` on the Editor menu bar.
3. Click on `Cut`.
4. Scroll to, then click on the new location to position the edit cursor where you want the text to appear.
5. Click on `Edit` again.
6. Click on `Paste`.

Copying Text

You can copy portions of code to a new location:

1. Select text to be copied.
2. Click on `Edit` on the Editor menu bar.
3. Click on `Copy`.
4. Scroll to, then click on the new location to position the edit cursor where you want the text to appear.
5. Click on `Edit` again.
6. Click on `Paste`.

Searching for Text

You can search for text in your program code by using the **Search** commands. A search always begins from the edit cursor's location.

Finding the First Occurrence

1. Click on the location from where you want the search to begin.
2. Click on **Search** on the Editor menu bar.
3. Click on **Find ...** and enter the information.
   a. Type in the text you want to search for.
   b. Click on the box next to **Match Case** to control searches for capitalization. An X means the search finds specific capitalization.
   c. Click on the box next to **Search Backward** to control the search direction. An X means the search goes backward through the text.
4. Click on OK.

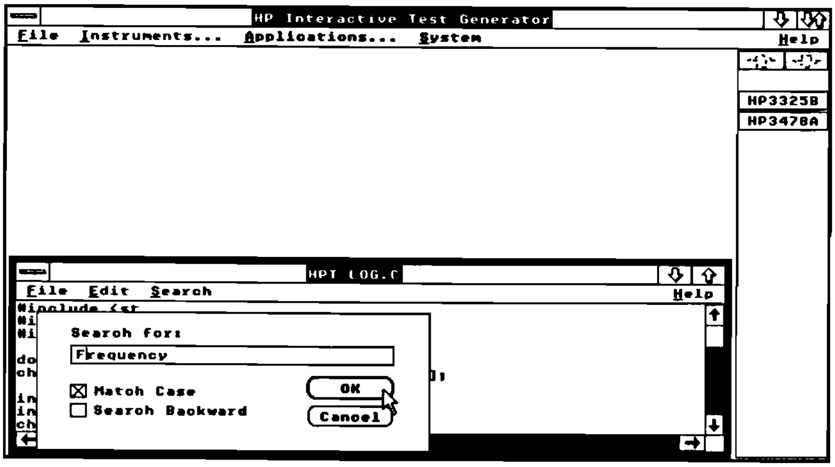


Figure 10-3. The Search Dialog Box

Finding the Next Occurrence

You can find additional occurrences of the text you last searched for:

1. Click on **Search** on the Editor menu bar.
2. Click on **Find Next**.

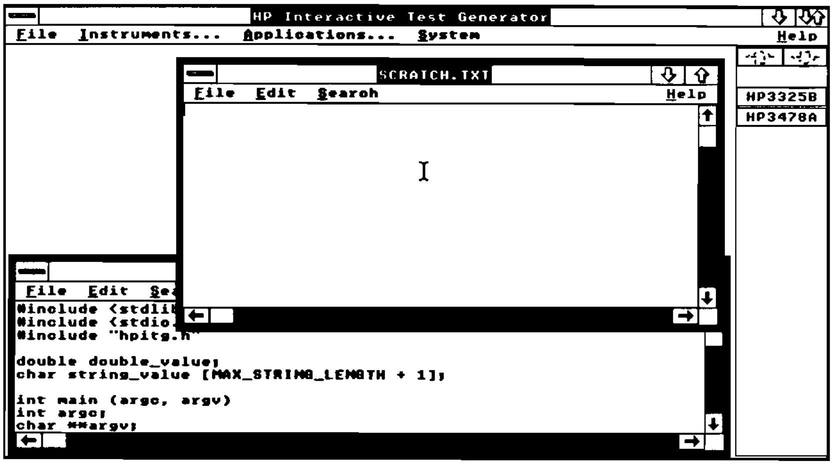
Getting Help on the Editor

You can get help information about how to use the Editor window:

1. Click on **Help** on the Editor menu bar.
2. Click on the topic name for which you need help.
3. Click on **OK**.

Adding Another Editor Window

You can add additional editor windows to your work area to use for file comparisons, for reference while you create another program, or to break your program into smaller modules. The default file name for these additional editor windows is `SCRATCH.TXT`.


HP Interactive Test Generator
File Instruments... Applications... System Help
SCRATCH.TXT
File Edit Search Help
HP3325B
HP3478A
File Edit Se

```
#include <stdli
#include <stdio.
#include "hpitg.h"

double double_value;
char string_value [MAX_STRING_LENGTH + 1];

int main (argc, argv)
int argc;
char **argv;
```

Figure 10-4. An Additional Editor Window

1. Click on `Applications ...` on the System menu bar.
2. Click on `EDITOR.AD`.
3. Click on `Open`.

Note

HP ITG does not generate code in the `EDITOR.AD` editor window, although you can load and save files, and type text into this window.

11

Generating Code Using Panels

Overview

Generating a measurement procedure using HP ITG involves several tasks, such as turning on the HP ITG log mode, generating initialization code, stepping through the measurement, and editing the code. This chapter explains how to do them. To help you edit the code generated with HP ITG, this chapter also explains how HP ITG subprograms are called and what they do in your programs.

Log Mode

After you have created the states you need for a particular measurement procedure, you are ready to turn on Log HP ITG Calls mode. This mode generates calls to subprograms in the HP ITG Library based on your interactions with the panels. You can turn this mode on or off as needed during program generation.

1. Click on the panel menu box.
2. Click on `Modes ...`
3. Click on the box beside the label `Log HP ITG Calls` so an `X` appears in the box.

The HP ITG Subprograms

The Basics

When you turn on Log HP ITG Calls for a panel, HP ITG generates calls to subprograms in the HP ITG Library. These subprograms control an instrument based on your interactions with its panel.

The three primary subprograms are `hpt_set`, `hpt_get`, and `hpt_recall`. There are also variations of `hpt_set` and `hpt_get` that pass string and array data (for example, `hpt_set_str` and `hpt_get_iary`).

Any control in a driver file is accessible by `hpt_set`, `hpt_get`, and their variations. A control's value is changed by `hpt_set`. To take readings, `hpt_get` triggers the instrument.

Instrument states you stored when working with the instrument's soft panel are recalled by using `hpt_recall`. Recalling a state sets the instrument controls to the new settings.

For a complete list and description of the HP ITG subprograms, please see chapter 14 and 15 in this handbook, "The HP ITG Library: QuickBASIC" and "The HP ITG Library: C." These chapters describe the library for each language HP ITG supports.

The Passed Parameters

Most HP ITG subprograms have a set of required parameters. When HP ITG generates a subprogram call, it generates variable names or uses values for the parameters based on how you have the panel set up. The following examples show the calls HP ITG generates when you enter the value 10000 for the frequency on the HP 3325B panel.

- In C:
  - `hpt_set(hp3325b,"FREQUENCY",10000)`
- In QuickBASIC:
  - `hptset(hp3325b,"FREQUENCY",10000)`

`hp3325b` is a variable name that HP ITG generates based on the name you assigned to the panel when you added the instrument to your soft test system. `FREQUENCY` is the control name located in the HP 3325B driver, and `10000` is the value entered from the panel.

Editing HP ITG Subprograms

To produce a more efficient program, you should determine if and how you will need to edit the code that HP ITG generates. In the previous examples calling `hpt_set`, you may want to replace the number `10000` with a variable. Then you can include the statement in a loop and change the variable to other frequency values.

In addition, there are several subprograms in the HP ITG Library that are not generated automatically. For a complete list and description of these subprograms, please see chapter 14 and 15 in this handbook, "The HP ITG Library: QuickBASIC" and "The HP ITG Library: C." These chapters describe the library for each language HP ITG supports.

Generating Code

You will need to complete several steps to prepare HP ITG to generate and save your instrument-control code. Be sure to save the current workfile before generating the initialization code.

Initialization

Every program that generates code using Log HP ITG Calls, must include calls to the following HP ITG subprograms.

hpt_init

This subprogram must be called before all other HP ITG subprograms. It identifies the workfile where the instruments and their states are located and puts the instruments controlled by your program into their initial states.

hpt_assign

Your program must call `hpt_assign` for each instrument included in a soft test system. You can use HP ITG to generate these calls in the HP ITG Editor window.

1. Save the workfile after adding all required instruments to the soft test system.
2. Turn on Log HP ITG Calls mode for each panel.
   a. Click on the panel menu box.
   b. Click on `Modes` ...
   c. Click on the box beside `Log HP ITG Calls` if there is no `X` in it.
   d. Click on `OK`.
3. Click on `Edit` on the Editor menu bar.
4. Click on `Generate Initialization Code`.

Note

Leave the edit cursor in the current position after generating the initialization code. HP ITG automatically positions the edit cursor at the line where you should continue code generation.

Stepping Through a Procedure

To continue your planned measurement procedure, do one or more of the following steps for each instrument in the soft test system. Log HP ITG Calls mode must be turned on for each instrument.

1. Recall a previously stored instrument state.
   - Click on the panel menu box, then click on `Recall State`.
2. Adjust the controls as needed.
   - Click on the box in the panel next to the control's name, then select or enter the new value.
3. Take a reading, if appropriate.
   - Click on the panel's display.

Editing the Procedure

You can use the HP ITG Editor for simple editing, such as substituting variable names for values and adding loops around HP ITG subprogram calls. However, if you have more complicated program needs and want to include calls to routines in other libraries, you should edit the program in the programming environment you selected when you installed HP ITG.

HP ITG declares variables in the initialization code and leaves the program statements as comments. You can uncomment them as needed. You will need to declare any other string or array variables that are used in HP ITG subprogram calls. All HP ITG arrays are two-dimensional, and you must declare them as such even if you only need one dimension.

To see more of your procedure as you edit it, enlarge the Editor window to full size:

- Click on the up arrow in the top right corner of the Editor window.

Saving Your Program

When you are ready to save your program, you can save it under the current file name or under a different file name.

1. Click on `File` on the Editor menu bar.
2. To save your program under the *current* file name and path, click on `Save`.
3. To save your program under a *different* file name or to change the path:
   a. Click on `Save As ...`
   b. Type in the file name and include the path if it is different from the path displayed in the dialog box.
   c. Click on `OK`.

Exiting HP ITG

When you have finished the code generation, and have saved your program, you can exit HP ITG.

1. Click on `File` on the System menu bar.
2. Click on `Exit To DOS`.

12

Running Your Program

Overview

This chapter explains how you can run your HP ITG-generated program to complete its development, then create a standalone executable for the run-time environment.

In the Development Environment

Using the same development system you used to generate instrument-control code, you can enhance your program by adding calls to routines in the HP ITG Library as well as other MS-DOS libraries. When the program is debugged, you can then create the executable to run on the development system, or to be installed on a run-time system.

In Microsoft C

If you are using Microsoft C, use a text editor you prefer to include additional code in your program. To run and debug the program, use the Microsoft C compiler making sure to link against the HP ITG Library, `HPITG.LIB`, and any other library you are using. The batch file, `CMAKE.BAT`, supplied with HP ITG runs the compiler on the program and links against `HPITG.LIB`. To use the batch file for a program named `HPT_LOG.C`, enter the following command at the MS-DOS prompt:

```
cmake hpt_log
```

HP ITG error messages can appear in MS-DOS as the program runs if HP ITG errors exist in the program. The messages include the error number, the HP ITG function name, and the message contents. See appendix E in this handbook, "Messages," for explanations. When the program is done, you are ready to use it in the run-time environment.

In QuickC

If you are using the QuickC environment, you can start QuickC using a batch file supplied with HP ITG, `QCSTART.BAT`. Enter the following command at the MS-DOS prompt including the program file name:

```
qcstart hpt_log.c
```

The batch file loads the program, `HPT_LOG.C` into the QuickC environment. It also creates a make file for the program. When QuickC starts a dialog box appears asking if you want to use the make file. Click on `Yes` to accept the file. Then you are ready to add calls to functions in the HP ITG Library. HP ITG error messages can appear in the QuickC environment if HP ITG errors exist in the program. The messages include the error number, the HP ITG function name, and the message contents. See appendix E in this handbook, "Messages," for explanations.

When the program is debugged, you are ready to create the executable. While you can create the executable within the QuickC environment, HP ITG supplies another batch file that can do the same task. The batch file, `QCMAKE.BAT`, compiles the program, and links it against the HP ITG Library, `HPITG.LIB`. This creates an executable with an `.EXE` file name extension. To use the batch file, enter the following command at the MS-DOS prompt including the program file name:

```
qcmake hpt_log
```

You can distribute the executable, `HPT_LOG.EXE`, to run-time systems along with the workfile (if required), instrument driver files, and `HPITG.ERR` which support the executable.

In QuickBASIC

If you are using the QuickBASIC environment, you can start QuickBASIC using a batch file supplied with HP ITG, QBSTART.BAT. Enter the following command at the MS-DOS prompt including the program file name:

```
qbstart hpt_log.bas
```

Note

QuickBASIC requires 512 Kbytes of free memory. If QuickBASIC reports an out-of-memory error, reduce the demand on the computer's memory. Modify CONFIG.SYS and AUTOEXEC.BAT, deleting device drivers for unnecessary peripheral devices or memory-resident applications such as network servers. Use the MS-DOS CHKDSK command to read the amount of free memory. Reboot the computer after modifying these files.

The batch file loads the program, HPT_LOG.BAS, and the HP ITG Quick Library, HPITGBAS.QLB. Then you are ready to add calls to subprograms in the HP ITG Library. HP ITG error messages can appear in the QuickBASIC environment if HP ITG errors exist in the program. The messages include the error number, the HP ITG subprogram name, and the message contents. See appendix E in this handbook, "Messages," for explanations.

When the program is debugged, you are ready to create the executable. While you can create the executable within the QuickBASIC environment, HP ITG supplies another batch file that can do the same task. The batch file, QBMAKE.BAT, compiles the program, and links it against the HP ITG Library, HPITGBAS.LIB. This creates an executable with an .EXE file name extension. To use the batch file, enter the following command at the MS-DOS prompt including the program file name:

```
qbmake hpt_log
```

You can distribute the executable, HPT_LOG.EXE, to run-time systems along with the workfile (if required), instrument driver files, and HPITG.ERR which support the executable.

In the Run-Time Environment

If you run the HP ITG-based program on the same system that you used to develop the program, your system already has the required support files. If you run the executable on a different run-time system, you must load the following onto the system controller:

- The executable program.
- The HP ITG error file, HPITG.ERR.
- The soft test system's workfile (if the program uses a workfile).
- The instrument driver files named in the workfile (if used) or added by calls to `hpt_add_device` in the program.

To run the executable, enter the program name at the MS-DOS prompt. If HP ITG errors exist, possibly due to a difference in systems, messages can appear on the screen as the program runs. The messages include the HP ITG error number, the subprogram or function name, and the message contents. See appendix E in this handbook, "Messages," for explanations.

13

Fine-Tuning Your Program

Overview

To develop finely-tuned, complex test programs using HP ITG, you will probably find yourself needing to edit your program directly to call subprograms in the HP ITG Library. The subprograms provide access to the instrument panel controls described by the instrument drivers. The real strength of HP ITG is that it provides instrument drivers and access to them.

HP ITG provides two ways that you can access these drivers. The first is the development environment. It is designed to let you access drivers automatically through the interactions with the instrument panels. The interactions generate calls to HP ITG Library subprograms.

The second way lets you access the drivers by adding calls to HP ITG Library subprograms when editing your program. In fact, calls to over half of the available HP ITG subprograms must be made by manually adding calls to them in your program since HP ITG cannot generate them.

The HP ITG Library Reference

Chapters 14 and 15 contain the reference to the HP ITG Library. Chapter 14 is the reference for the QuickBASIC subprograms. Chapter 15 is the reference for the C functions. These chapters describe the syntax, variables, and operation for each subprogram and function.

Certain naming conventions are used in this chapter to make explanations more clear. *Subprogram* is used in a general sense when referring to QuickBASIC subprograms and C functions. Similarly, subprogram names such as `hpt_recall` include underscores, though the QuickBASIC language does not allow underscores.

Using the Driver Documentation

To use the subprograms effectively, you need to have specific information about the structure of each instrument driver you will use. To help you find that information, each driver provided by Hewlett-Packard is documented in HP ITG's online Help system. You can access the information using the `Help` command on the System menu bar or the instrument panel menu.

From the System menu bar:

1. Click on `Help`.
2. Click on `Instrument Help` ...

From the instrument panel:

1. Click on the panel menu box.
2. Click on `Help` ...

The information summarizes instrument operation, status bytes, error messages, and driver components.

Most of the information you will need when adding calls to the HP ITG Library is listed under `Components`. The following table is a representative portion of the component table for the HP 3325B driver help information. As you see, it lists the component names used in the driver to control such features as amplitude and calibration, the range of values components can be set to, and the initial values.

COMPONENT NAME	ALLOWED VALUES	INITIAL VALUE
Am	Off, On	Off
Amplitude	.001 - 10	.001
Amptd_cal		
Assign_p		
Aunits	Vpp, Rms, dBm	Vpp
Calibrate	Disable, Enable	Enable

Working With Components

The concept of a component is important to your understanding about how a driver file interacts with HP ITG to control an instrument. In the driver, a **component** is a logical representation of an instrument's control or display. Values are listed for a control to name the possible settings. When many components are combined, the resulting driver represents the entire instrument. However, some instrument features may not be represented in a driver.

Most HP ITG subprograms require between one or more parameters. These parameters identify a particular driver, the component to change, and its value. Here is a generalized example that illustrates the type of component information a subprogram call needs:

- `hpt_set(`*inst_desc,comp_name,value*`)`
  - *inst_desc* is the instrument descriptor which is an internal number representing a particular driver. The number is generated by the `hpt_assign` subprogram.
  - *comp_name* is the name of the component in the driver. The component's value will be changed by the `hpt_set` subprogram.
  - *value* is a variable containing the new value for the component.

If you want more information about components, see chapter 3, "The Component Section," in the HP ITG manual, *How to Write an HP ITG Driver.*

Determining the Instrument Descriptor

The subprogram, `hpt_assign`, establishes the relationship between an actual instrument at a specific HP-IB address and the instrument driver. One of the parameters used by `hpt_assign` is the logical name entered for the instrument when you added it to the soft test system. The name is entered in the `Name` field of the instrument configuration dialog box.

The subprogram returns an integer that is unique to the driver with the particular name. This is an important distinction since, as you recall, you can add the same driver to your soft test system more than once if you give each one a different name. Various other subprograms use the returned integer as the instrument descriptor in their parameter lists.

If you generated the initialization code for your program using `Generate Initialization Code` (see `Edit` on the Editor menu bar), then HP ITG generated a default instrument descriptor for you to use. If you add an instrument by calling `hpt_add_device`, then you must also call `hpt_assign` to generate the descriptor.

Choosing the Right Subprogram Variation

Components use different types of data for the component values, depending on the component's function. In a driver, the TYPE statement identifies the data type, but you can identify the type by the way component values appear in the component table. The following list describes the data types and how they might be used in drivers:

- DISCRETE: An enumerated list of values for the function settings on a voltmeter.
- INTEGER: A value that is a 16-bit integer defining an error number.
- CONTINUOUS: A value that is a 64-bit real number for a function generator's frequency range.
- STRING: A string value that is not more than 256 characters for a displayed message.
- IARRAY: A value that is a two-dimensional array of integers used for calibration data.
- RARRAY: A value that is a two-dimensional array of real numbers for an oscilloscope's trace display.

Several subprograms are available in different variations to accommodate the various data types. As an example, the following table shows which **hpt_set** subprogram variation to use for each TYPE.

Table 13-1.
Component Types and Allowable Subprogram Variations

TYPE	Subprogram Variation
DISCRETE	hpt_set or hpt_set_str
INTEGER	hpt_set
CONTINUOUS	hpt_set
STRING	hpt_set_str
IARRAY	hpt_set_iary
RARRAY	hpt_set_ary

Putting It Together

To use the HP ITG subprograms, you will need to do the following:

1. Determine the instrument descriptor (see the previous subsections for details).
2. Refer to `Instrument Help ...` for the component name whose value you want to change.
3. Note the allowable values for the component.
4. Determine the component's TYPE.
5. Use the subprogram variation appropriate for the component TYPE (see table 13-1).

Uses for the HP ITG Subprograms

Creating and Recalling States

Instrument states can be created and recalled in a program.

`hpt_state_save`	Lets you create a new instrument state.
`hpt_recall`	Lets you recall an existing instrument state.

Turning On/Off HP ITG Modes

You can control many of HP ITG's modes from your program.

`hpt_incremental`	Controls incremental state programming.
`hpt_livemode`	Lets you disable Live mode within your program.
`hpt_errorcheck`	Lets you control Error Checking mode.
`hpt_monitor`	Lets you control Monitor mode for program debugging.

Controlling Component Values

Each component has a value and a status associated with it when it is being used in a program. The component usually represents an instrument's control, such as a function generator's frequency. The value, such as 10 kHz, represents the control's setting. A component's current value can be set to one of three status levels. Each status requires certain action from HP ITG.

- VALID:

  This means the component's value matches the instrument's corresponding value, which requires HP ITG to take no action.

- INVALID:

  This means the component's value does not match the instrument's corresponding value, requiring HP ITG to determine the correct value and send it to the instrument.

- DONTCARE:

  This means HP ITG should not send a value to the instrument, whether the instrument and component values match or not.

You can change any component's value and status.

`hpt_forget`	Lets you set the status of one or all of an instrument driver components to INVALID.
`hpt_getstate`	Lets you find out whether a component is currently VALID, INVALID, or DONTCARE.
`hpt_peek`	Along with its string and array variations, lets you find out the current value of the component.
`hpt_poke`	Along with its string and array variations, lets you change the value of a component without adjusting the instrument.
`hpt_setstate`	Lets you set a component's status to VALID, INVALID, or DONTCARE.

Improving Efficiency

The subprogram calls generated by HP ITG pass string parameters for state names, component names, and for the values of certain component types such as DISCRETE components. There are several subprograms that let you convert these string names to internal numbers. You can then use these internal numbers in alternate versions of many of the subprograms. Using the internal numbers in these *Version 2* subprograms can increase your program's execution speed.

hpt_assigncomp	Lets you assign a component name to a numeric variable. HP ITG supplies the number. If you do this, you will need to change all the subprogram calls associated with the component to their alternate version. For example, if you use a component name in a call to **hpt_get**, change the subprogram name to **hpt_get2** to use the assigned number as one of the arguments.
hpt_assignparm	Lets you replace a DISCRETE component's string values with numbers. HP ITG supplies the number. If you do use this, you will need to change all **hpt_set_str** calls that use the string values to **hpt_set** calls when you set the value of this component.
hpt_assignstate	Lets you assign a state name to a numeric variable. HP ITG supplies the number. If you do this, you will need to change all calls to hpt_recall to call hpt_recall2 instead.

Reading and Changing HP-IB Addresses

You can read and change the HP-IB address in HP ITG's instrument configuration.

`hpt_dev_addr`	Lets you read the HP-IB address of a particular instrument.
`hpt_devsubad`	Lets you read the HP-IB subaddress of a particular instrument.
`hpt_setdevaddr`	Lets you change the HP-IB address of a particular instrument in HP ITG's configuration.

Note

You cannot change the subaddress configuration programmatically.

Setting Array Values

HP ITG does not generate calls to the following subprograms because they are seldom used. If you do need them, you can include them in your program.

`hpt_set_iary`	Lets you set the value of an IARRAY component.
`hpt_set_ary`	Lets you set the value of an RARRAY component.
`hpt_compdims`	Lets you determine an array component's dimensions.

Note

Though HP ITG generates array variables in the initialization code, it does generate calls to `hpt_get_ary` or `hpt_get_iary` when you click on an XY display.

Checking for Errors

You can turn Error Checking mode on for instruments in the test system and have them display the messages.

`hpt_errorcheck`	Lets you specify that an instrument should notify HP ITG when an instrument error has occurred.
`hpt_errmsg`	Lets you display any messages that HP ITG receives while your program is running.

Adding Instruments

You can add instruments to your test system without having to include them in the workfile. This lets you develop a program without using the HP ITG development environment.

`hpt_add_device`	Lets you add an instrument to your test system outside of the workfile. The subprogram parameters require the same information as the instrument configuration dialog box does when adding an instrument to a soft test system. You must also call `hpt_assign` immediately after adding an instrument using this method.

14

The HP ITG Library: QuickBASIC

Overview

The HP ITG Library is a set of subprograms provided with HP ITG. Calls to several subprograms are generated automatically by HP ITG in the development environment when you interact with instrument panel controls. Calls to many other subprograms must be added to your program when you are ready to edit it.

This chapter describes the subprograms in the HP ITG Library for the QuickBASIC programming language. The subprograms are arranged in alphabetical order and are described using the following format:

- Syntax.
- Description.
- Parameters.
- Example.
- See Also.

To use these subprograms, you will need information from instrument drivers about component names and their values. That information helps you supply the arguments the parameter lists require. The component information is located in HP ITG's online Help system under `Instrument Help` ... The manual, *How to Write an HP ITG Driver,* describes how instrument drivers and their components are written.

HP ITG automatically generates calls to subprograms when an instrument panel is set to Log HP ITG Calls mode. There are other subprograms in the library to which HP ITG cannot generate calls. To use them, you must edit your program to add the call statements.

The file HPITG.BI is the include file that supports these subprograms. It declares the subprograms and the constants required as arguments. The specific constants (when required) are defined in the subprogram descriptions. The statement to include HPITG.BI in your program is added automatically when you generate the HP ITG initialization code.

Many subprograms have an alternate version, identified with a 2 appended to the main version subprogram names. These **Version 2** subprograms differ from the main versions in syntax, but return the same values with greater execution speed. These subprograms use HP ITG-assigned integer values for parameter arguments instead of strings to improve performance. The subprograms, **hptassigncomp**, **hptassignparm**, and **hptassignstate**, are used for the integer assignments.

The following tables list the HP ITG subprograms. Table 14-1 lists the subprograms that HP ITG can generate calls to automatically. Table 14-2 lists the subprograms for which calls cannot be generated automatically.

Table 14-1. Call Statements Generated by HP ITG for These QuickBASIC Subprograms

Main Version	Version 2
hptassign	
hptget	hptget2
hptgetary	hptgetary2
hptgetiary	hptgetiary2
hptgetstr	hptgetstr2
hptinit	
hptpush	hptpush2
hptrecall	hptrecall2
hptset	hptset2
hptsetstr	hptsetstr2

Table 14-2.
Additional HP ITG Subprograms for QuickBASIC

Main Version	Version 2
hptadddevice	
hptassigncomp	
hptassignparm	
hptassignstate	
hptclose	
hptcloseall	
hptcompdims	
hptdevaddr	
hptdevsubad	
hpterrmsg	
hpterrorcheck	
hptforget	hptforget2
hptgetstate	hptgetstate2
hptincremental	
hptlivemode	
hptlocal	
hptmeminfo	
hptmonitor	
hptpeek	hptpeek2
hptpeekary	hptpeekary2
hptpeekiary	hptpeekiary2
hptpeekstr	hptpeekstr2
hptpoke	hptpoke2
hptpokeary	hptpokeary2
hptpokeiary	hptpokeiary2
hptpokestr	hptpokestr2
hptremote	
hptsetary	hptsetary2
hptsetdevaddr	
hptseterrormode	
hptsetiary	hptsetiary2
hptsetstate	hptsetstate2
hptstatesave	

hptadddevice

Syntax

`hptadddevice` (*instr$, logical$, addr$, subaddr$, timeout#*)

Description

The `hptadddevice` subprogram lets you add an *instr$* to your test system without adding it to the workfile in the HP ITG development environment. The remaining parameters are identical to the entries you make in the instrument configuration dialog box when adding an instrument to a soft test system in the development environment. Specify the *logical$* name for the instrument, the HP-IB *addr$* and *subaddr$* (if needed), and the *timeout#*.

Note

The instrument's compiled instrument driver file (`.CID`) must exist for you to add the instrument using `hptadddevice`. Whether you add one or more instruments, you need to call the `hptassign` subprogram after each call to `hptadddevice`.

Parameters

instr$	The instrument driver file name (8 characters maximum). The `.ID` extension must be omitted.
logical$	The logical name for the instrument, such as `source` for the HP 3325B (25 characters maximum).
addr$	The instrument's HP-IB address (16 characters maximum).
subaddr$	The instrument's subaddress (16 characters maximum). Leave argument NULL ("") if none.

timeout#	The instrument's timeout period for the development and run-time environments. If the argument for this parameter is an integer variable, it must include parentheses and use the % specifier—*(val%)*.

Example

```
CALL hptadddevice("HP3488A3", "switch", "708", "2", 30.0)
```

See Also `hptassign`

hptassign

Syntax `hptassign` (*inst$*, *memory%*, *instdesc%*)

Description

Note A call to the `hptassign` subprogram is required in all programs that use HP ITG subprograms. To generate this code, click on `Generate Initialization Code` in the HP ITG Editor window's `Edit` menu. Generate this code at the beginning of your program. Be sure to save the workfile first.

The `hptassign` subprogram informs HP ITG you are using the instrument named *inst$* in your test system. You assign the instrument name when you add it to your test system. You may have added the instrument to your test system in the HP ITG development environment or by calling `hptadddevice`.

Use one of the following arguments for *memory%:*

- CONVENTIONAL.
- EXPANDED (default when generated with initialization code).

If the instrument driver exists, HP ITG opens the driver file and assigns a positive integer in the return variable, *instdesc%*. Use this number in other subprograms to control the instrument. If a 0 is returned, the assignment failed.

Note To deallocate the memory used by the instrument states, you must close all open instruments before exiting an application. Use `hptclose` or `hptcloseall` to close instruments.

Parameters

inst$	This variable contains the instrument name you gave the instrument when you added it to your test system.
memory%	A variable describing the type of memory where the instrument driver should be loaded. EXPANDED is recommended.
instdesc%	HP ITG assigns an integer value to this return variable. This is the instrument descriptor which you use to refer to the instrument in other subprograms that use this variable.

Example

```
CALL hptassign("scope", EXPANDED, scope%)
```

See Also

`hptadddevice`, `hptclose`, `hptcloseall`, `hptinit`, `hptmeminfo`

hptassigncomp

Syntax `hptassigncomp` (*instdesc%, comp$, compdesc&*)

Description The `hptassigncomp` subprogram assigns a component descriptor to *compdesc&*. *compdesc&* is a unique number for the *comp$* and *instdesc%* pair. *instdesc%* is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

Use the component descriptor in the Version 2 subprograms for higher performance.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the component in the instrument driver. This is the component the subprogram assigns an integer value to.
compdesc&	HP ITG assigns a long integer value to this return variable. This is the component descriptor which is unique to this component/instrument pair. Use this descriptor in the Version 2 subprograms for faster performance.

Example

```
CALL hptassigncomp(voltmeter%, "RANGE", range_number&)
.
.
.
CALL hptset2(range_number&, 2)
```

See Also `hptassign`, `hptassignstate`, `hptassignparm`

Many HP ITG subprograms have an alternate version, identified by the 2 appended to the main version subprogram names. These **Version 2** subprograms return the same values, and are usually used in conjunction with `hptassigncomp`. The following list names the subprograms that have both versions. See their descriptions in this chapter for details about each version.

```
hptforget
hptget
hptgetiary
hptgetstr
hptgetstate
hptpeek
hptpeekary
hptpeekiary
hptpeekstr
hptpoke
hptpokeary
hptpokeiary
hptpokestr
hptpush
hptrecall
hptset
hptsetary
hptsetiary
hptsetstr
hptsetstate
```

hptassignparm

Syntax `hptassignparm` (*instdesc%, comp$, disc$, val%*)

Description The `hptassignparm` subprogram assigns the internal ordinal number for *disc$,* a value name in a component, to the variable, *val%. comp$* names the component that contains *disc$. instdesc%* is the integer that identifies a specific instrument in your workfile, and is assigned by the subprogram, `hptassign`.

Use `hptassignparm` to assign numbers to the values listed in DISCRETE components. Use the returned number in calls to the subprogram, `hptset` or `hptset2`, rather than using the value's name in calls to the subprogram, `hptsetstr`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the component in the workfile's instrument driver.
disc$	The string variable that names the DISCRETE value in the component. The subprogram assigns an integer to the return variable for this value.
val%	HP ITG assigns an integer value to this return variable. The returned result is unique to the component/instrument pair specified in the parameter list. Use the returned value in calls to the subprogram, `hptset` or `hptset2` for faster performance.

Example

```
CALL hptassigncomp(hp3314a%, "Function", comp_desc&)
CALL hptassignparm(hp3314a%, "Function", "Sine", comp_val%)
.
.
.
CALL hptset2(comp_desc&, comp_val%)
```

See Also hptassigncomp, hptassignstate, hptset, hptset2, hptsetstr

hptassignstate

Syntax `hptassignstate` (*instdesc%, state$, statedesc&*)

Description The `hptassignstate` subprogram returns the internal number for an instrument state (*state$*) to the variable *statedesc&* for the instrument specified by *instdesc%*. *instdesc%* is the integer that identifies the specific instrument in your workfile, and is assigned by the subprogram, `hptassign`. The state name is created when you save a setup state for that instrument in the development environment.

Use the number returned to *statedesc&* in calls to the subprogram, `hptrecall2`. This improves processing time over using the state name in calls to the subprogram, `hptrecall`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
state$	The string variable that names the state created for the instrument in your workfile.
statedesc&	HP ITG assigns a value to this return variable. This is the state descriptor which is unique to the test system. Use this descriptor in calls to the subprogram, `hptrecall2` for faster performance.

Example

```
CALL hptassignstate(hp3437a%, "HPTINITIALSTATE", statedesc&)
.
.
.
CALL hptrecall2 (statedesc&)
```

See Also hptassign, hptassigncomp, hptassignparm, hptrecall, hptrecall2

hptclose

Syntax `hptclose` (*instdesc%*)

Description The `hptclose` subprogram closes and deallocates all memory used by the instrument driver and instrument states associated with *instdesc%*. It also closes the monitor file, if applicable. *instdesc%* is the integer that identifies a specific instrument in your workfile, and is assigned by the subprogram, `hptassign`. Use `hptclose` when an instrument is no longer needed in a program to free memory.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.

Example

```
CALL hptclose(hp3325b%)
```

See Also `hptassign`, `hptcloseall`, `hptmeminfo`, `hptmonitor`

hptcloseall

Syntax `hptcloseall`

Description The `hptcloseall` subprogram closes and deallocates all memory used by all instrument drivers and instrument states in the workfile. It also closes all monitor files, if applicable. HP ITG generates a call to this subprogram as part of the initialization code. It should appear at the end of the program to free memory.

Example `CALL hptcloseall`

See Also `hptassign`, `hptclose`, `hptmeminfo`, `hptmonitor`

hptcompdims

Syntax `hptcompdims` (*instdesc%, comp$, rows%, cols%*)

Description The `hptcompdims` subprogram returns the number of rows (*rows%*) and columns (*cols%*) in an array component. *comp$* names the component which should exist in the instrument driver associated with *instdesc%*. *instdesc%* is the integer that identifies a specific instrument in your workfile, and is assigned by the subprogram, `hptassign`.

Use this information to determine the array dimensions of subprogram variables in subsequent calls to other subprograms affecting arrays, such as `hptsetary` and `hptgetary`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the array component in the workfile's instrument driver.
rows%	HP ITG returns the number of rows in the array component in this variable.
cols%	HP ITG returns the number of columns in the array component in this variable.

Example

```
CALL hptcompdims(hp54501a%, "ENV_CH1", row%, col%)
```

See Also hptassign, hptgetary, hptgetiary, hptpeekary, hptpeekiary, hptpokeary, hptpokeiary, hptsetary, hptsetiary

hptdevaddr

Syntax `hptdevaddr` (*instdesc%, addr$*)

Description The `hptdevaddr` subprogram returns the current HP-IB address of the instrument associated with *instdesc%*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, **`hptassign`**. See appendix B, "I/O Interfaces," for information about interface board select codes.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, **`hptassign`**.
addr$	HP ITG returns the instrument's address in this string variable.

Example

```
CALL hptdevaddr(hp3478a%, addr$)
```

See Also `hptassign`, `hptsetdevaddr`, `hptdevsubad`

hptdevsubad

Syntax `hptdevsubad` (*instdesc%, subaddr$*)

Description The `hptdevsubad` subprogram returns the current subaddress of the instrument module associated with *instdesc%*. The instrument descriptor is the integer that identifies a specific module in your workfile, and is returned by the subprogram, **`hptassign`**.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, **`hptassign`**.
subaddr$	HP ITG returns the instrument's subaddress in this string variable.

Example

```
CALL hptdevsubad(hp3488a3%, subaddr$)
```

See Also `hptassign`, `hptdevaddr`

hpterrmsg

Syntax `hpterrmsg` (*error$*)

Description The `hpterrmsg` subprogram returns an error message, which corresponds to the most recent HP ITG error. The error can be from an instrument or from HP ITG. For information about specific error codes and messages, see appendix E, "Messages," in this handbook.

Note Error Check mode must be enabled for each instrument that you want to receive an error message from. See the subprogram, `hpterrorcheck`.

Parameters

error$	HP ITG returns the error message in this string variable.

Example

```
CALL hpterrmsg(message$)
```

See Also `hpterrorcheck`

hpterrorcheck

Syntax `hpterrorcheck` (*instdesc%, switch%*)

Description The `hpterrorcheck` subprogram uses the values ON or OFF for *switch%* to enable or disable Error Checking mode for the instrument specified by *instdesc%.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

To use `hpterrorcheck`, the instrument must be able to report error information, and the instrument driver must contain an error component. Error Check mode is disabled by default when running a program using HP ITG subprograms. You must include `hpterrorcheck` in your program to enable error checking. HP ITG then checks for errors after each call to the subprograms, `hptget`, `hptrecall`, `hptset`, or `hptpush` (including the related subprograms for strings and arrays, and their Version 2 subprograms).

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
switch%	The variable that controls Error Check mode. Use the argument ON to enable error checking, or OFF to disable error checking.

Example

```
CALL hpterrorcheck(hp3325a%, ON)
```

See Also

hptassign, hptget, hptget2, hptgetary, hptgetary2, hptgetiary, hptgetiary2, hptgetstr, hptgetstr2, hptrecall, hptset, hptset2, hptsetary, hptsetary2, hptsetiary, hptsetiary2, hptsetstr, hptsetstr2

hptforget, hptforget2

Syntax

`hptforget` (*instdesc%, comp$*)

`hptforget2` (*compdesc&*)

Description

The `hptforget` subprogram sets the status of one or all of the components of the instrument specified by *instdesc%* to INVALID. If *comp$* is specified, the single component's status is invalidated. You must call this subprogram for each specific component you want to invalidate. If *comp$* is NULL (""), then all of the instrument's components are invalidated. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

The `hptforget2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

Use `hptforget` or `hptforget2` if you have changed the instrument's setup either manually or with a program's I/O statements.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the component in the workfile's instrument driver that you want to invalidate.

compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example `CALL hptforget(hp3478a%, "TRIGGER")`

See Also `hptassign`, `hptassigncomp`

hptget, hptget2

Syntax

`hptget` (*instdesc%, comp$, val#*)

`hptget2` (*compdesc&, val#*)

Description

The `hptget` subprogram returns the current value (*val#*) of an INTEGER or CONTINUOUS component (*comp$*) directly from the instrument specified as *instdesc%.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

The `hptget2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

HP ITG generates a call to the `hptget` subprogram when you click on an instrument display. To use the `hptget2` subprogram, you will have to modify your test program after you generate it.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the INTEGER or CONTINUOUS component in the workfile's instrument driver.
val#	HP ITG returns the current value of the instrument in this real variable.

compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example

```
CALL hptget(hp3437a%, "READING", reading#)
```

See Also `hptassign`, `hptassigncomp`

hptgetary, hptgetary2

Syntax

`hptgetary` (*instdesc%, comp$, val#()*)

`hptgetary2` (*compdesc&, val#()*)

Description

The `hptgetary` subprogram returns the current value (*val#()*) of an RARRAY component (*comp$*) directly from the instrument specified as *instdesc%*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

The `hptgetary2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

HP ITG generates a call to the `hptgetary` subprogram when you click on an instrument display. To use the `hptgetary2` subprogram, you will have to modify your test program after you generate it.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the RARRAY component in the workfile's instrument driver.

val#()	HP ITG returns the current value of the instrument in this real array variable. This variable must be properly dimensioned in the program. Use `hptcompdims` to read the array dimensions.
compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example

```
CALL hptgetary(hp54501a%, "ENV_CH2", meas_values#())
```

See Also `hptassign`, `hptassigncomp`, `hptcompdims`

hptgetiary, hptgetiary2

Syntax

`hptgetiary` (*instdesc%*, *comp$*, *val%*())

`hptgetiary2` (*compdesc&*, *val%*())

Description

The **`hptgetiary`** subprogram returns the current value (*val%*()) of an IARRAY component (*comp$*) directly from the instrument specified as *instdesc%*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, **`hptassign`**.

The **`hptgetiary2`** subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, **`hptassigncomp`**.

HP ITG generates a call to the **`hptgetiary`** subprogram when you click on an instrument display. To use the **`hptgetiary2`** subprogram, you will have to modify your test program after you generate it.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, **`hptassign`**.
comp$	The string variable that names the IARRAY component in the workfile's instrument driver.

val%()	HP ITG returns the current value of the instrument in this integer array variable. This variable must be properly dimensioned in the program. Use `hptcompdims` to read the array dimensions.
compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example

```
CALL hptgetiary(hp5334a%, "CalibrationData", cal_data%())
```

See Also `hptassign`, `hptassigncomp`, `hptcompdims`

hptgetstate, hptgetstate2

Syntax

`hptgetstate` (*instdesc%, comp$, state%*)

`hptgetstate2` (*compdesc&*, *state%*)

Description

The `hptgetstate` subprogram returns the current component (*comp$*) status for the instrument specified as *instdesc%* in the variable, *state%*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

The subprogram returns one of the following status values in *state%:*

- COMP_VALID (component is VALID).
- COMP_INVALID (component is INVALID).
- COMP_DONTCARE (component is DONTCARE).

The `hptgetstate2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the component in the workfile's instrument driver.
state%	HP ITG returns the current component status in this variable.

compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example

```
CALL hptgetstate(hp3325b%, "Function", current_state%)
```

See Also

`hptassign`, `hptassigncomp`

hptgetstr, hptgetstr2

Syntax

`hptgetstr` (*instdesc%, comp$, val$*)

`hptgetstr2` (*compdesc&*, *val$*)

Description

The `hptgetstr` subprogram returns the current value (*val$*) of a STRING or DISCRETE component (*comp$*) directly from the instrument specified as *instdesc%.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

The `hptgetstr2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

HP ITG generates a call to the `hptgetstr` subprogram when you click on an instrument display. To use the `hptgetstr2` subprogram, you will have to modify your test program after you generate it.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the STRING or DISCRETE component in the workfile's instrument driver.
val$	HP ITG returns the current value of the instrument in this string variable.

compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example

```
CALL hptgetstr(hp3325b%, "Function", current_function$)
```

See Also `hptassign`, `hptassigncomp`

hptincremental

Syntax

`hptincremental` (*instdesc%, switch%*)

Description

The `hptincremental` subprogram uses the values ON or OFF for *switch%* to enable or disable Incremental mode for the instrument specified by *instdesc%.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

The default setting is ON for this mode in the HP ITG development environment. Use this subprogram when you have changed the instrument's setup manually.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
switch%	The variable that controls Incremental mode. Use the argument ON to enable Incremental mode, or OFF to disable Incremental mode.

Example

```
CALL hptincremental(hp3478a%, OFF)
```

See Also

`hptassign`

hptinit

Syntax `hptinit` (*workfile$*)

Description

Note A call to the `hptinit` subprogram is required when you use a workfile, and the program calls HP ITG subprograms affecting the instruments in that workfile. This subprogram must be called before any other HP ITG subprogram is called. You can generate this code automatically from the HP ITG Editor window's menu bar. Click on `Edit`, then `Generate Initialization Code`. Be sure you have saved the workfile and the editor cursor is at the beginning of the file.

The `hptinit` subprogram names the current workfile for HP ITG, and initializes the data used by the rest of HP ITG. If the argument for *workfile$* does not specify an absolute path name, HP ITG uses the current directory to find the workfile name. The file name for *workfile$* does not need to include the `.WF` extension used to identify workfiles.

Parameters

workfile$	The string variable that specifies the soft test system for HP ITG to read into the work space. The argument may include a complete path name.

Example

```
CALL hptinit("FRQ_RESP.WF")
```

See Also `hptassign`

hptlivemode

Syntax `hptlivemode` (*instdesc%, switch%*)

Description The `hptlivemode` subprogram uses the values ON or OFF for *switch%* to enable or disable Live mode for the instrument specified by *instdesc%*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

The default setting for this mode in a program is ON. HP ITG sends HP-IB commands directly to instruments when Live mode is on. Use `hptlivemode` to turn Live mode OFF for some instruments while you are developing other parts of your program.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
switch%	The variable that controls Live mode. Use the argument ON to enable Live mode, or OFF to disable Live mode.

Example

```
CALL hptlivemode(hp3325b%, OFF)
```

See Also `hptassign`

hptlocal

Syntax `hptlocal` (*instdesc%*)

Description The `hptlocal` subprogram puts the instrument specified by *instdesc%* into local mode, preventing the program's I/O commands from controlling the instrument. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, **`hptassign`**.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, **`hptassign`**.

Example

```
CALL hptlocal(oscilloscope%)
```

See Also `hptassign`, `hptremote`

hptmeminfo

Syntax `hptmeminfo` (*instdesc%, conv%, ems%*)

Description The `hptmeminfo` subprogram reports the amount of conventional (*conv%*) and expanded (*ems%*) memory being used by the instrument driver, states, and configuration information specified by *instdesc%.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
conv%	HP ITG returns the amount of conventional memory (in Kbytes) being used in this integer variable. Typically, this is between 2-3 Kbytes depending on the instrument driver size.
ems%	HP ITG returns the amount of expanded memory (in Kbytes) being used in this integer variable. Expanded memory usage is expressed as a multiple of 16 Kbytes, even if less memory is actually used.

Example

```
CALL hptmeminfo(voltmeter%, conv_mem_usage%, exp_mem_usage%)
```

See Also `hptassign`, `hptclose`, `hptcloseall`

hptmonitor

Syntax `hptmonitor` (*instdesc%, mode, filename$*)

Description The `hptmonitor` subprogram controls HP ITG's Monitor mode for program debugging. If Monitor mode is on, the type of debugging information specified by *mode* is written to *filename$*. All of the information is written during program execution for the instrument specified by *instdesc%*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, **hptassign**.

To report results more immediately, set *filename$* to a printer port. As an example, if a printer is configured to the computer's LPT1 port, use `"LPT1"` for *filename$*.

Use the following constants as arguments for *mode:*

- OFF—Monitor mode is off.
- ON—Log I/O transactions to *filename$*.
- DRIVER_DEBUG—Log I/O transactions and additional information to *filename$*.

The default setting is OFF in a program. Use this subprogram to turn on Monitor mode for debugging purposes.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, **hptassign**.
mode	Specifies the mode setting.
filename$	Specifies the destination file for the debugging information.

Example

```
CALL hptmonitor(scope%, DRIVER_DEBUG, "DEBUG.ERR")
```

See Also

hptassign, hptclose, hptcloseall

hptpeek, hptpeek2

Syntax

hptpeek (*instdesc%, comp$, val#*)

hptpeek2 (*compdesc&, val#*)

Description

The hptpeek subprogram returns the current value (*val#*) of the INTEGER or CONTINUOUS component (*comp$*) for the instrument specified by *instdesc%.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, hptassign. The returned value is read directly from the results of the most recent call to the hptget or hptpoke subprograms.

The hptpeek2 subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, hptassigncomp.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, hptassign.
comp$	The string variable that names the INTEGER or CONTINUOUS component in the workfile's instrument driver.
val#	HP ITG returns the component's value in this real variable.
compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, hptassigncomp.

Example `CALL hptpeek(hp3325b%, "Phase", current_val#)`

See Also hptassign, hptassigncomp, hptget, hptpoke

hptpeekary, hptpeekary2

Syntax

`hptpeekary` (*instdesc%, comp$, val#()*)

`hptpeekary2` (*compdesc&, val#()*)

Description

The `hptpeekary` subprogram returns the current value (*val#()*) of the RARRAY component (*comp$*) for the instrument specified by *instdesc%*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`. The returned value is read directly from the results of the most recent call to the `hptgetary` or `hptpokeary` subprograms.

The `hptpeekary2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the RARRAY component in the workfile's instrument driver.
val#()	HP ITG returns the component's value in this real array variable. This variable must be properly dimensioned in the program. Use `hptcompdims` to read the array dimensions.

compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example

```
CALL hptpeekary(hp54501a%, "ENV_CH1", comp_val#())
```

See Also

`hptassign`, `hptassigncomp`, `hptcompdims`, `hptgetary`, `hptpokeary`

hptpeekiary, hptpeekiary2

Syntax `hptpeekiary` (*instdesc%, comp$, val%()*)

`hptpeekiary2` (*compdesc&*, *val%()*)

Description The `hptpeekiary` subprogram returns the current value (*val%()*) of the IARRAY component (*comp$*) for the instrument specified by *instdesc%.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`. The returned value is read directly from the results of the most recent call to the `hptgetiary` or `hptpokeiary` subprograms.

The `hptpeekiary2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the IARRAY component in the workfile's instrument driver.
val%()	HP ITG returns the component's value in this integer array variable. This variable must be properly dimensioned in the program. Use `hptcompdims` to read the array dimensions.

compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example

```
CALL hptpeekiary(hp5334a%, "CalibrationData", cal_data%())
```

See Also

`hptassign`, `hptassigncomp`, `hptcompdims`, `hptgetiary`, `hptpokeiary`

hptpeekstr, hptpeekstr2

Syntax

`hptpeekstr` (*instdesc%, comp$, val$*)

`hptpeekstr2` (*compdesc&*, *val$*)

Description

The `hptpeekstr` subprogram returns the current value (*val$*) of the STRING component (*comp$*) for the instrument specified by *instdesc%.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

The `hptpeekstr2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the STRING component in the workfile's instrument driver.
val$	HP ITG returns the component's value in this string variable.
compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example

```
CALL hptpeekstr(source%, "Function", src_comp_func$)
```

See Also hptassign, hptassigncomp, hptgetstr, hptpokestr

hptpoke, hptpoke2

Syntax

`hptpoke` (*instdesc%, comp$, val#*)

`hptpoke2` (*compdesc&, val#*)

Description

The `hptpoke` subprogram sets the internal variable of the CONTINUOUS or INTEGER component (*comp$*) for the instrument specified by *instdesc%* to a value (*val#*). The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`. The `hptpoke` subprogram adjusts only the component's value, not the instrument.

The `hptpoke2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the INTEGER or CONTINUOUS component in the workfile's instrument driver.
val#	HP ITG changes the component to this value. If the argument for this parameter is an integer variable, it must include parentheses and use the % specifier—*(val%)*.

compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example `CALL hptpoke(voltmeter%, "RANGE", new_range#)`

See Also `hptassign`, `hptassigncomp`, `hptset`

hptpokeary, hptpokeary2

Syntax

`hptpokeary` (*instdesc%, comp$, val#()*)

`hptpokeary2` (*compdesc&, val#()*)

Description

The `hptpokeary` subprogram sets the internal variable of the RARRAY component (*comp$*) for the instrument specified by *instdesc%* to a value (*val#*). The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`. The `hptpokeary` subprogram adjusts only the component's value, not the instrument.

The `hptpokeary2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the RARRAY component in the workfile's instrument driver.
val#()	HP ITG changes the component to this value. This variable must be properly dimensioned in the program. Use `hptcompdims` to read the array dimensions.

compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example

```
CALL hptpokeary(voltmeter%, "READINGS", new_readings#())
```

See Also

`hptassign`, `hptassigncomp`, `hptcompdims`, `hptsetary`

hptpokeiary, hptpokeiary2

Syntax

`hptpokeiary` (*instdesc%, comp$, val%()*)

`hptpokeiary2` (*compdesc&, val%()*)

Description

The `hptpokeiary` subprogram sets the internal variable of the IARRAY component (*comp$*) for the instrument specified by *instdesc%* to a value (*val%*). The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`. The `hptpokeiary` subprogram adjusts only the component's value, not the instrument.

The `hptpokeiary2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the IARRAY component in the workfile's instrument driver.
val%()	HP ITG changes the component to this value. This variable must be properly dimensioned in the program. Use `hptcompdims` to read the array dimensions.

compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example

```
CALL hptpokeiary(switch%, "SET_CHAN", new_chans%())
```

See Also

`hptassign`, `hptassigncomp`, `hptcompdims`, `hptsetiary`

hptpokestr, hptpokestr2

Syntax

`hptpokestr` (*instdesc%, comp$, val$*)

`hptpokestr2` (*compdesc&, val$*)

Description

The `hptpokestr` subprogram sets the internal variable of the STRING component (*comp$*) for the instrument specified by *instdesc%* to a value (*val$*). The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`. The `hptpokestr` subprogram adjusts only the component's value, not the instrument.

The `hptpokestr2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the STRING component in the workfile's instrument driver.
val$	HP ITG changes the component to this value.

compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example

```
CALL hptpokestr(voltmeter%, "FUNCTION", "ACV")
```

See Also `hptassign`, `hptassigncomp`, `hptsetstr`

hptpush, hptpush2

Syntax

`hptpush` (*instdesc%, comp$*)

`hptpush2` (*compdesc&*)

Description

The **hptpush** subprogram executes the set actions associated with the component named by *comp$* for the instrument specified by *instdesc%*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, **hptassign**.

The **hptpush2** subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, **hptassigncomp**.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, **hptassign**.
comp$	The variable that names the component in the instrument driver.
compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, **hptassigncomp**.

Example

```
CALL hptpush(scope%, "RESET")
```

See Also

hptassign, hptassigncomp

hptrecall, hptrecall2

Syntax

`hptrecall` (*instdesc%, state$*)

`hptrecall2` (*statedesc&*)

Description

The `hptrecall` subprogram recalls the instrument state specified by *state$* for the instrument specified by *instdesc%.* The subprogram changes the instrument settings to the values contained in the recalled state. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

The `hptrecall2` subprogram produces the same results. As a Version 2 subprogram, the parameter *statedesc&* replaces *instdesc%* and *state$* for higher performance. Its value can be assigned by the subprograms, `hptassignstate` and `hptstatesave`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
state$	The name of the instrument state to be recalled.
statedesc&	The state descriptor that replaces *instdesc%* and *state$* in the Version 2 subprogram. Its value is unique to the instrument/state pair, and can be assigned by the subprograms, `hptassignstate` and `hptstatesave`.

Example

```
CALL hptrecall(analyzer%, "HPTINITIALSTATE")
```

See Also hptassign, hptassignstate, hptstatesave

hptremote

Syntax `hptremote` (*instdesc%*)

Description The `hptremote` subprogram puts the instrument specified by *instdesc%* into remote mode. An instrument in remote mode can be controlled by a program's I/O commands. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.

Example

```
CALL hptremote(voltmeter%)
```

See Also `hptassign`, `hptlocal`

hptset, hptset2

Syntax

`hptset` (*instdesc%, comp$, val#*)

`hptset2` (*compdesc&, val#*)

Description

The **hptset** subprogram sets the CONTINUOUS or INTEGER component (*comp$*) and the instrument specified by *instdesc%* to a value (*val#*). The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, **hptassign**.

The **hptset2** subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, **hptassigncomp**.

In the HP ITG development environment, HP ITG automatically generates a call to **hptset** when you adjust an instrument's soft panel control. To use **hptset2**, you have to call **hptassigncomp** and edit the call to **hptset** to change the parameter list's arguments.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, **hptassign**.
comp$	The string variable that names the INTEGER or CONTINUOUS component in the workfile's instrument driver.

val#	HP ITG sets the component and instrument to this value. If the argument for this parameter is an integer variable, it must include parentheses and use the % specifier—*(val%)*. For a list of allowed values, see the component summary table for the instrument in HP ITG's online Help system under `Instrument Help` ...
compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example

```
CALL hptset(voltmeter%, "RANGE", 5)
```

See Also `hptassign`, `hptassigncomp`, `hptpoke`

hptsetary, hptsetary2

Syntax

`hptsetary` (*instdesc%, comp$, val#()*)

`hptsetary2` (*compdesc&*, *val#()*)

Description

The `hptsetary` subprogram sets the RARRAY component (*comp$*) and the instrument specified by *instdesc%* to a value (*val#()*). The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

The `hptsetary2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the RARRAY component in the workfile's instrument driver.
val#()	HP ITG adjusts the component and instrument to this value. This variable must by properly dimensioned in the program. For a list of allowed values, see the component summary table in HP ITG's online Help system under `Instrument Help ...`

compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example `CALL hptsetary(hp54501a%, "ENV_CH2", values#())`

See Also `hptassign`, `hptassigncomp`, `hptcompdims`, `hptpokeary`

hptsetdevaddr

Syntax `hptsetdevaddr` (*instdesc%, addr$*)

Description The `hptsetdevaddr` subprogram changes the HP ITG instrument configuration specified by *instdesc%* to the HP-IB address, *addr$*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`. See appendix B, "I/O Interfaces," for information about interface board select codes.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
addr$	The string variable containing the HP-IB address for the instrument. Only 16 characters are significant.

Example

```
CALL hptsetdevaddr(counter%, "703")
```

See Also `hptadddevice`, `hptassign`, `hptdevaddr`, `hptdevsubad`

hptseterrormode

Syntax `hptseterrormode` (*mode%*)

Description The `hptseterrormode` subprogram sets the error mode (*mode%*) for generated programs. Three modes are allowed:

- PRINTITGERROR prints HP ITG errors and maintains program execution.
- IGNOREITGERROR ignores HP ITG errors.
- STOPITGERROR prints HP ITG errors and suspends program execution while in the QuickBASIC environment.

Parameters

mode%	Use one of the following arguments for the parameter: ▪ PRINTITGERROR ▪ IGNOREITGERROR ▪ STOPITGERROR (default)

Example

```
CALL hptseterrormode(PRINTITGERROR)
```

See Also `hpterrmsg`

hptsetiary, hptsetiary2

Syntax

`hptsetiary` (*instdesc%, comp$, val%()*)

`hptsetiary2` (*compdesc&, val%()*)

Description

The `hptsetiary` subprogram sets the IARRAY component (*comp$*) and the instrument specified by *instdesc%* to a value (*val%()*). The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

The `hptsetiary2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the IARRAY component in the workfile's instrument driver.
val%()	HP ITG sets the component and instrument to this value. This variable must be properly dimensioned in the program. For a list of the allowed values, see the instrument's component summary table in HP ITG's online Help system under `Instrument Help` ...

compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example

```
CALL hptsetiary(switch%, "SET_CHAN", new_chans%())
```

See Also

`hptassign`, `hptassigncomp`, `hptcompdims`, `hptpokeiary`

hptsetstate, hptsetstate2

Syntax

`hptsetstate` (*instdesc%, comp$, state%*)

`hptsetstate2` (*compdesc&, state%*)

Description

The `hptsetstate` subprogram lets you change the status (*state%*) of the component (*comp$*) in the instrument specified as *instdesc%*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

Set the component status using one of the following values for *state%:*

- COMP_VALID (sets component status to VALID).
- COMP_INVALID (sets component status to INVALID).
- COMP_DONTCARE (sets component status to DONTCARE).

The `hptsetstate2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the component in the workfile's instrument driver.

state%	HP ITG changes the component to the status defined by the parameter's argument.
compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, **hptassigncomp**.

Example

```
CALL hptsetstate(hp3325a%, "Function", COMP_DONTCARE)
```

See Also `hptassign`, `hptassigncomp`

hptsetstr, hptsetstr2

Syntax

`hptsetstr` (*instdesc%, comp$, val$*)

`hptsetstr2` (*compdesc&, val$*)

Description

The `hptsetstr` subprogram sets the STRING component (*comp$*) and the instrument specified by *instdesc%* to a value (*val$*). The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

The `hptsetstr2` subprogram produces the same results. As a Version 2 subprogram, the parameter *compdesc&* replaces *instdesc%* and *comp$* for higher performance. Its value is assigned by the subprogram, `hptassigncomp`.

In the HP ITG development environment, HP ITG automatically generates a call to `hptsetstr` when you adjust an instrument's soft panel control. To use `hptsetstr2`, you have to call `hptassigncomp` and edit the call to `hptsetstr` to change the parameter list's arguments.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
comp$	The string variable that names the STRING component in the workfile's instrument driver.
val$	HP ITG adjusts the component and instrument to this value.

compdesc&	The component descriptor that replaces *instdesc%* and *comp$* in the Version 2 subprogram. Its value is unique to the instrument/component pair, and is assigned by the subprogram, `hptassigncomp`.

Example `CALL hptsetstr(hp3325b%, "Function", "DCV")`

See Also `hptassign`, `hptassigncomp`, `hptpokestr`

hptstatesave

Syntax `hptstatesave` (*instdesc%, state$, statedesc&*)

Description The `hptstatesave` subprogram lets you create a new instrument state with the state name *state$* for the instrument specified as *instdesc%*. An internal number corresponding to the new state is returned in the variable, *statedesc&*. This state descriptor can be used in calls to `hptrecall2`. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the subprogram, `hptassign`.

Parameters

instdesc%	The instrument descriptor, an integer that specifies the instrument in your workfile. Its value is assigned by the subprogram, `hptassign`.
state$	The string variable containing the name for the newly created state.
statedesc&	HP ITG returns a descriptor in this variable corresponding to the new state.

Example

```
CALL hptstatesave(analyzer%, "read_response", state_descriptor&)
```

See Also `hptassign`, `hptassignstate`, `hptrecall`, `hptrecall2`

15

The HP ITG Library: C

Overview

The HP ITG Library is a set of functions provided with HP ITG. Calls to several functions are generated automatically by HP ITG in the development environment when you interact with instrument panel controls. Calls to many other functions must be added to your program when you are ready to edit it.

This chapter describes the functions in the HP ITG Library for the Microsoft C and QuickC languages. The functions are arranged in alphabetical order and are described using the following format:

- Syntax.
- Description.
- Return Value.
- Error Conditions.
- See Also.
- Example.

To use these functions, you will need information from instrument drivers about component names and their values. That information helps you supply the arguments the function parameter lists require. The component information is located in HP ITG's online Help system under `Instrument Help` ... The manual, *How to Write an HP ITG Driver,* describes how instrument drivers and their components are written.

HP ITG automatically generates function calls when an instrument panel is set to `Log HP ITG Calls` mode. There are other functions in the library to which

HP ITG cannot generate calls. To use them, you must edit your program to include the function calls.

The file, `HPITG.H`, is the include file that supports these functions. It declares the functions and the constants required as arguments. The specific constants (when required) are defined in the function descriptions. The statement to include `HPITG.H` in your program is added automatically when you generate the HP ITG initialization code.

Many functions have an alternate version, identified with a 2 appended to the main version function names. These **`Version 2`** functions differ from the main versions in syntax, but return the same values with greater execution speed. HP ITG-assigned integer values for parameter arguments are used instead of strings to improve performance. The functions, **`hpt_assigncomp`** and **`hpt_assignstate`**, are used for the integer assignments.

The following tables list the HP ITG subprograms. Table 15-1 lists the functions to which HP ITG can generate calls. Table 15-2 lists the functions to which calls cannot be generated.

**Table 15-1.
Calls Generated by HP ITG for These C Functions**

Main Version	Version 2
hpt_assign	
hpt_get	hpt_get2
hpt_get_ary	hpt_get_ary2
hpt_get_iary	hpt_get_iary2
hpt_get_str	hpt_get_str2
hpt_init	
hpt_push	hpt_push2
hpt_recall	hpt_recall2
hpt_set	hpt_set2
hpt_set_str	hpt_set_str2

Table 15-2. Additional HP ITG Functions for C

Main Version	Version 2
hpt_add_device	
hpt_assigncomp	
hpt_assignparm	
hpt_assignstate	
hpt_close	
hpt_close_all	
hpt_compdims	
hpt_devaddr	
hpt_devsubad	
hpt_errmsg	
hpt_errorcheck	
hpt_forget	hpt_forget2
hpt_getstate	hpt_getstate2
hpt_incremental	
hpt_livemode	
hpt_local	
hpt_mem_info	
hpt_monitor	
hpt_peek	hpt_peek2
hpt_peek_ary	hpt_peek_ary2
hpt_peek_iary	hpt_peek_iary2
hpt_peek_str	hpt_peek_str2
hpt_poke	hpt_poke2
hpt_poke_ary	hpt_poke_ary2
hpt_poke_iary	hpt_poke_iary2
hpt_poke_str	hpt_poke_str2
hpt_remote	
hpt_set_ary	hpt_set_ary2
hpt_set_error_handler	
hpt_set_iary	hpt_set_iary2
hpt_setdevaddr	
hpt_setstate	hpt_setstate2
hpt_state_save	

hpt_add_device

Syntax

`int hpt_add_device(`*instr, logical, addr, subaddr, timeout*`);`

`char far *`*instr*	Names the instrument driver file name (8 character maximum). The `.ID` extension must be omitted.
`char far *`*logical*	The logical name for the instrument, such as `source` (25 character maximum).
`char far *`*addr*	The instrument's HP-IB address (16 character maximum).
`char far *`*subaddr*	The instrument's subaddress (16 character maximum).
`double` *timeout*	The instrument's timeout period for the development and run-time environments.

Description

The `hpt_add_device` function lets you add an *instr* to your test system during run-time without adding it to the workfile in the HP ITG development environment. The remaining parameters are identical to the entries you specify in the instrument configuration dialog box when adding an instrument to a soft test system in the development environment. You must not include the `.ID` file name extension for *instr*. Specify the *logical* name for the instrument, the HP-IB *addr* and *subaddr* (if needed), and the *timeout*. If there is no *subaddr*, specify "".

Note

The instrument's compiled instrument driver file (.CID) must exist for you to add the instrument using **hpt_add_device**. Whether you add one or more instruments, you must call the **hpt_assign** function for each instrument.

Return Value

The **hpt_add_device** function returns a 0 if it could not add the instrument.

Error Conditions

A return value of 0 indicates one of the following error conditions:

- *instr* is an incorrect driver file name.
- The compiled instrument driver (.CID) file does not exist or could not be opened.

See Also

hpt_assign

Example

```
hpt_add_device("HP54501A", "SCOPE", "704", "", 30.0);
```

hpt_assign

Syntax

`int hpt_assign(`*inst, mem*`);`

`char far *`*inst*	Instrument soft panel name in workfile.
`int` *mem*	Names memory type where instrument driver file should be loaded.

Description

Note

A call to `hpt_assign` is required in all programs that use HP ITG functions. To generate this code in HP ITG, click on `Generate Initialization Code` in the Editor window's `Edit` menu. Generate this code at the beginning of your program. Be sure to save the workfile first.

The `hpt_assign` function informs HP ITG you are using the instrument named *inst* in your test system. You assign the instrument name when you add it to your test system. You may have added the instrument to the test system in the HP ITG development environment, or by calling `hpt_add_device`.

If the instrument driver file and associated states exist, HP ITG attempts to open the instrument driver file. The environment variable, HPITG, defines the directory that contains the driver file. Otherwise, HP ITG looks in the directory, `C:\HPITG`. If the file is opened successfully, HP ITG assigns an integer descriptor to the instrument. Other HP ITG functions use this number to control the instrument.

Use one of the following arguments to specify where the driver should be loaded:

- CONVENTIONAL.
- EXPANDED (default when generated with initialization code).

Note To deallocate computer memory, you must close all open instrument drivers before exiting an application. Use `hpt_close` or `hpt_close_all` to close instrument drivers.

Return Value

The `hpt_assign` function returns an instrument descriptor (an integer in the range from 1 through 254) that is unique to *inst.* The function returns 0 if an error occurs.

Error Conditions

A return value of 0 indicates one of the following error conditions:

- *inst* is not in the workfile specified by `hpt_init`, or has not been added by `hpt_add_device`.
- The instrument driver could not be opened.
- The workfile was not loaded successfully.

See Also

`hpt_add_device`, `hpt_close`, `hpt_close_all`, `hpt_init`, `hpt_mem_info`

Example

```
inst_desc = hpt_assign("SCOPE",EXPANDED);
```

hpt_assigncomp

Syntax

`long hpt_assigncomp(`*inst_desc, comp*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.

Description

The `hpt_assigncomp` function assigns a unique component descriptor to *comp*. *inst_desc* is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

The component descriptor is useful when *comp* is referenced many times.

Return Value

`hpt_assigncomp` returns a component descriptor that is unique to *comp* for the instrument driver referenced by *inst_desc*. The function returns 0 if it could not assign a component descriptor.

Error Conditions

A return value of 0 indicates one of the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc*.

See Also

`hpt_assign`

Many HP ITG functions have an alternate version, identified by the 2 appended to the main version function names. These **Version 2** functions return the same values, and are usually used in conjunction with `hpt_assigncomp`. The following list names the functions that have both versions. See their descriptions in this chapter for details about each.

hpt_forget
hpt_get
hpt_get_ary
hpt_get_iary
hpt_get_str
hpt_getstate
hpt_peek
hpt_peek_ary
hpt_peek_iary
hpt_peek_str
hpt_poke
hpt_poke_ary
hpt_poke_iary
hpt_poke_str
hpt_push
hpt_recall
hpt_set
hpt_set_ary
hpt_set_iary
hpt_set_str
hpt_setstate

Example

```
function_comp = hpt_assigncomp(inst_desc, "FUNCTION");
.
.
.
hpt_get2(function_comp, &value);
```

hpt_assignparm

Syntax

`int hpt_assignparm(`*inst_desc, comp, parm, discrete*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`char far *`*parm*	Component value name.
`int far *`*discrete*	Receives return value.

Description

The `hpt_assignparm` function returns the internal ordinal number for *parm,* a value name in a component, in the return variable, *discrete. comp* names the component that contains *parm. inst_desc* is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`. *comp* must exist in the instrument driver.

Use `hpt_assignparm` to assign numbers to the values listed in DISCRETE components. For improved performance, use the returned number in calls to the `hpt_set` function rather than using the value's name in calls to the `hpt_set_str` function.

Return Value

`hpt_assignparm` returns an ordinal number in *discrete.* This number is unique to *parm* which is a component value in the instrument driver referenced by *inst_desc.* The function returns a 0 if it could not assign a number.

Error Conditions

A return value of 0 indicates one of the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc.*
- *parm* does not exist as a DISCRETE value for this component.

See Also

`hpt_assign`, `hpt_set`, `hpt_set_str`

Example

```
status= hpt_assignparm(inst_desc, "WAVEFORM", "SINE", &discrete);
```

hpt_assignstate

Syntax

`long hpt_assignstate(`*inst_desc*, *state*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*state*	Instrument state name.

Description

The `hpt_assignstate` function returns the internal number for *state*. The state name is created when an instrument setup is saved as a state in the development environment. *inst_desc* is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

Use the returned number in calls to the function, `hpt_recall2`. This improves performance compared to using the state name in calls to the function, `hpt_recall`.

Return Value

`hpt_assignstate` returns a number that is unique to *state* for the instrument driver referenced by *inst_desc*. The function returns a 0 if it could not assign a number.

Error Conditions

A return value of 0 indicates one of the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *state* does not exist as a saved state for the instrument specified by *inst_desc*.

See Also hpt_assign, hpt_recall, hpt_recall2, hpt_state_save

Example

```
setup_state = hpt_assignstate(inst_descriptor, "SETUP");
.
.
.
hpt_recall2(setup_state);
```

hpt_close

Syntax

`int hpt_close(`*inst_desc*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.

Description

The `hpt_close` function closes the instrument driver associated with *inst_desc.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`. This activity frees up all conventional and expanded memory used by the instrument.

Return Value

`hpt_close` returns a 0 if the function could not close a file.

Error Conditions

A return value of 0 indicates one of the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- System memory problems exist.

See Also

`hpt_assign`, `hpt_close_all`, `hpt_mem_info`, `hpt_monitor`

Example

```
hpt_close(inst_descriptor);
```

hpt_close_all

Syntax `int hpt_close_all();`

Description The `hpt_close_all` function closes all instrument drivers opened by `hpt_assign`. Closing files frees up all conventional and expanded memory used by the instrument drivers.

Return Value `hpt_close_all` returns a 0 if the function could not close all of the instrument drivers.

Error Conditions A return value of 0 indicates the following error condition:

- System memory problems exist.

See Also `hpt_assign`, `hpt_close`, `hpt_mem_info`, `hpt_monitor`

Example

```
hpt_close_all();
```

hpt_compdims

Syntax

`int hpt_compdims(`*inst_desc, comp, rows, cols*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`int far *`*rows*	Receives return value.
`int far *`*cols*	Receives return value.

Description

The `hpt_compdims` function returns the number of *rows* and *cols* dimensioned for an array component. The variable *comp* names the component which should exist in the instrument driver associated with *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

Use this information to determine the array dimensions of function variables in subsequent calls to other functions affecting arrays, such as `hpt_set_ary` and `hpt_get_ary`.

Return Value

`hpt_compdims` returns the number of array rows in *rows* and the number of array columns in *cols*. The function returns a 0 if an error condition exists.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc*.

See Also hpt_assign, hpt_get_ary, hpt_get_iary, hpt_peek_ary, hpt_peek_iary, hpt_poke_ary, hpt_poke_iary, hpt_set_ary, hpt_set_iary

Example

```
hpt_compdims(inst_descriptor, "DISPLAY", &rows, &columns);
```

hpt_devaddr

Syntax

`int hpt_devaddr(`*inst_desc, address*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*address*	Receives return value.

Description

The `hpt_devaddr` function returns the current *address* of the instrument specified by *inst_desc.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`. See appendix B, "I/O Interfaces," for information about interface board select codes.

The program must allocate sufficient space for the address. Include a statement in the program using the constant, ADDRESS_LENGTH, which is defined in the include file, `HPITG.H`.

Return Value

`hpt_devaddr` returns a number into the string pointed to by *address* that is the interface bus address for the instrument referenced by *inst_desc.* The function returns a 0 if it could not read the address.

Error Conditions

A return value of 0 indicates the following error condition:

- *inst_desc* does not match any assigned instruments in the workfile.

See Also

`hpt_assign`, `hpt_devsubad`, `hpt_setdevaddr`

Example

```
char address[ADDRESS_LENGTH];
hpt_devaddr(inst_descriptor, address);
```

hpt_devsubad

Syntax

`int hpt_devsubad(`*inst_desc, address*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*address*	Receives return value.

Description

The `hpt_devsubad` function returns the current subaddress (*address*) of the instrument specified by *inst_desc*. The instrument descriptor is the integer that identifies the specific instrument in your workfile, and is returned by the function, `hpt_assign`.

The program must allocate sufficient space for the subaddress. Include a statement in the program using the constant, ADDRESS_LENGTH, which is defined in the include file, `HPITG.H`.

Return Value

`hpt_devsubad` returns a number into the string pointed to by *address*. The number is the device subaddress for the instrument module referenced by *inst_desc*. The function returns a 0 if it could not read the subaddress.

Error Conditions

A return value of 0 indicates the following error condition:

- *inst_desc* does not match any assigned instruments in the workfile.

See Also

`hpt_assign`, `hpt_devaddr`, `hpt_setdevaddr`

Example

```
char subaddress[ADDRESS_LENGTH];
hpt_devsubad(inst_descriptor, subaddress);
```

hpt_errmsg

Syntax

`int hpt_errmsg(error);`

`char far *error`	Receives error message.

Description

The **hpt_errmsg** function sets the error pointer to *error*, a null-terminated string. The string contains the most recent HP ITG error message. The message can be from an instrument or from HP ITG.

The program must allocate sufficient space for the message. Include a statement in the program using the constant, MAX_ERR_MSG, which is defined in the include file, `HPITG.H`.

Return Value

hpt_errmsg sets a pointer; it does not return a value.

Error Conditions

There are no conditions that can cause this function to return other than a NULL value or a pointer to an error message.

See Also

hpt_errorcheck, **hpt_set_error_handler**

Example

```
char error[MAX_ERR_MSG]; hpt_errmsg(error);
printf("%s\n",error);
```

hpt_errorcheck

Syntax `int hpt_errorcheck(`*inst_desc, switch*`);`

`int` *inst_desc*	Instrument descriptor assigned by **`hpt_assign`**.
`int` *switch*	Boolean switch turns Error Check mode ON or OFF.

Description The **`hpt_errorcheck`** function turns the Error Checking mode ON or OFF for the instrument specified by *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, **`hpt_assign`**. *switch* is a Boolean value that controls Error Check mode. Use the following constants as arguments for *switch:*

- OFF (turns Error Check mode OFF).
- ON (turns Error Check mode ON).

To use **`hpt_errorcheck`**, the instrument must be able to report error information, and the instrument driver must contain an error component. Error Check mode is OFF by default when running a program using HP ITG functions. You must add **`hpt_errcheck`** to your program to turn Error Check mode ON. HP ITG then checks for errors after each call to the functions, **`hpt_get`**, **`hpt_set`**, or **`hpt_recall`** (including the related functions for strings and arrays, and their Version 2 functions).

Return Value **`hpt_errorcheck`** returns a 0 if it could not switch the Error Check mode for the specified instrument.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- The specified instrument cannot report errors.

See Also

`hpt_assign`, `hpt_get`, `hpt_set`, `hpt_recall`

Example

```
/* Turn Error Check mode ON */
hpt_errorcheck(inst_descriptor, ON);
```

hpt_forget, hpt_forget2

Syntax

Standard Version

`int hpt_forget(`*inst_desc, comp*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.

Version 2

`int hpt_forget2(`*inst_desc, comp_desc*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.

Description

The `hpt_forget` and `hpt_forget2` functions set component state(s) of the instrument specified by *inst_desc* to INVALID. If *comp* is specified, the single component's state is invalidated. You must call these functions for each specific component state you want to invalidate. If *comp* or *comp_desc* are left NULL (""), then all of the instrument's component states are invalidated. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

Use `hpt_forget` or `hpt_forget2` to set component states to INVALID that might have been set to VALID. Component states can be set to VALID using `hpt_setstate` if you change an instrument's setup either manually or with a program's I/O statements.

Return Value

hpt_forget and hpt_forget2 return a 0 if error conditions prevent them from setting a component state to INVALID.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc.*
- *comp_desc* is an invalid descriptor.

See Also

hpt_assign, hpt_assigncomp, hpt_setstate

Example

```
/*  Standard version  */
hpt_forget(inst_descriptor, "READING");

/*  Version 2  */
read_descriptor=hpt_assigncomp(inst_descriptor,"READING");
hpt_forget2(inst_descriptor, read_descriptor);
```

hpt_get, hpt_get2

Syntax

Standard Version

`int hpt_get(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`double far *`*val*	Receives return value.

Version 2

`int hpt_get2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`double far *`*val*	Receives return value.

Description

The `hpt_get` function returns the current *val* of an INTEGER or CONTINUOUS *comp* directly from the instrument specified as *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

The `hpt_get2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

HP ITG generates a call to the `hpt_get` function when you click on an instrument display. To use the `hpt_get2` function, you will have to modify your test program after you generate it.

Return Value

hpt_get and hpt_get2 return the current value of the instrument's component in *val.* Both functions return a 0 if error conditions prevent them from returning the component value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc.*
- *comp_desc* is not a valid component descriptor.
- An HP-IB error occurred.

See Also

hpt_assign, hpt_assigncomp

Example

```
/*  Standard version  */
double frequency;
.
.
.
hpt_get(inst_descriptor, "FREQUENCY", &frequency);

/*  Version 2  */
double frequency;
freq_comp = hpt_assigncomp(inst_descriptor, "FREQUENCY");
.
.
.
hpt_get2(freq_comp, &frequency);
```

hpt_get_ary, hpt_get_ary2

Syntax

Standard Version

`int hpt_get_ary(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`double far *`*val*	Receives returned array contents.

Version 2

`int hpt_get_ary2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`double far *`*val*	Receives returned array contents.

Description

The `hpt_get_ary` function returns the current *val* of an RARRAY *comp* for the instrument specified as *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

The program must allocate sufficient space for *val*. Use `hpt_compdims` to read the array component's dimensions.

The `hpt_get_ary2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

HP ITG generates a call to the `hpt_get_ary` function when you click on an instrument display. To use the `hpt_get_ary2` function, you will have to modify your test program after you generate it.

Return Value

`hpt_get_ary` and `hpt_get_ary2` return the current value of the instrument's component in *val.* Both functions return a 0 if error conditions prevent them from returning the component value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc.*
- *comp_desc* is not a valid component descriptor.
- An HP-IB error occurred.

See Also

`hpt_assign`, `hpt_assigncomp`, `hpt_compdims`

Example

```
/*  Standard version  */
hpt_compdims (inst_descriptor, "FREQUENCIES", &rows, &cols);

frequencies=(double far *) malloc(rows * cols * sizeof(double));
.
.
.
hpt_get_ary(inst_descriptor, "FREQUENCIES", frequencies));

/*  Version 2  */
double frequencies[NUM_VALUES];
freq_comp = hpt_assigncomp(inst_descriptor, "FREQUENCIES");
.
.
.
hpt_get_ary2(freq_comp, frequencies);
```

hpt_get_iary, hpt_get_iary2

Syntax

Standard Version

`int hpt_get_iary(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`int far *`*val*	Receives returned array contents.

Version 2

`int hpt_get_iary2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`int far *`*val*	Receives returned array contents.

Description

The `hpt_get_iary` function returns the current *val* of an IARRAY *comp* for the instrument specified as *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

The program must allocate sufficient space for *val*. Use `hpt_compdims` to read the array component's dimensions.

The `hpt_get_iary2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

HP ITG generates a call to the `hpt_get_iary` function when you click on an instrument display. To use the `hpt_get_iary2` function, you will have to modify your test program after you generate it.

Return Value

`hpt_get_iary` and `hpt_get_iary2` return the current value of the instrument's component in *val.* Both functions return a 0 if error conditions prevent them from returning the component value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc.*
- *comp_desc* is not a valid component descriptor.
- An HP-IB error occurred.

See Also

`hpt_assign`, `hpt_assigncomp`, `hpt_compdims`

Example

```
/*  Standard version  */
hpt_compdims (inst_descriptor, "POINTS", &rows, &cols);

points=(int far *) malloc(rows * cols * sizeof(int));
.
.
.
hpt_get_iary(inst_descriptor, "POINTS", points);

/*  Version 2  */
int points[NUM_POINTS];
point_comp = hpt_assigncomp(inst_descriptor, "POINTS");
.
.
.
hpt_get_iary2(point_comp, points);
```

hpt_get_str, hpt_get_str2

Syntax

Standard Version

`int hpt_get_str(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`char far *`*val*	Receives return value.

Version 2

`int hpt_get_str2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`char far *`*val*	Receives return value.

Description

The `hpt_get_str` function returns the current *val* of a STRING or DISCRETE *comp* for the instrument specified as *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

Be sure to dimension the string *val* large enough to hold the returned value. Strings can have a maximum length of 256 bytes.

The `hpt_get_str2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

HP ITG generates a call to the `hpt_get_str` function when you click on a DISCRETE or STRING panel control. To use the hpt_get_str2 function, you will have to edit your test program.

Return Value

hpt_get_str and hpt_get_str2 return the current value of the instrument's component in *val.* Both functions return a 0 if error conditions prevent them from returning the component value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc.*
- *comp_desc* is not a valid component descriptor.
- An HP-IB error occurred.

See Also

hpt_assign, hpt_assigncomp

Example

```
/*  Standard version  */
char string[STRING_SIZE];
.
.
.
hpt_get_str(inst_descriptor, "STRING_COMP", string);

/*  Version 2  */
char string[STRING_SIZE];
string_comp = hpt_assigncomp(inst_descriptor, "STRING");
.
.
.
hpt_get_str2(string_comp, string);
```

hpt_getstate, hpt_getstate2

Syntax

Standard Version

`int hpt_getstate(`*inst_desc, comp*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.

Version 2

`int hpt_getstate2(`*comp_desc*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.

Description

The `hpt_getstate` function returns the status of *comp* for the instrument specified as *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

The `hpt_getstate2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

Return Value

Both functions return the following status values:

- COMP_VALID (component is VALID).
- COMP_INVALID (component is INVALID).
- COMP_DONTCARE (component is DONTCARE).

Both functions return a 0 if error conditions prevent them from reading the component's status.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc*.
- *comp_desc* is not a valid component descriptor.

See Also

hpt_assign, hpt_assigncomp

Example

```
/*  Standard version  */
status = hpt_getstate(inst_descriptor, "MODE");
switch (status) {
     case COMP_VALID:    printf("Valid\n");
     break;
     case COMP_INVALID:    printf("Invalid\n");
     break;
     case COMP_DONTCARE:    printf("Don't Care\n");
}

/*  Version 2  */
mode_comp = hpt_assigncomp(inst_descriptor, "MODE");
int status;
.
.
.
status = hpt_getstate2(mode_comp);
switch (status) {
     case COMP_VALID:    printf("Valid\n");
     break;
     case COMP_INVALID:    printf("Invalid\n");
     break;
     case COMP_DONTCARE:    printf("Don't Care\n");
}
```

hpt_incremental

Syntax `int hpt_incremental(`*inst_desc, switch*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`int` *switch*	Boolean switch turns Incremental mode ON or OFF.

Description The `hpt_incremental` function turns Incremental mode ON or OFF for the instrument specified by *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`. *switch* is a Boolean value that controls Incremental mode. Use the following constants as arguments for *switch:*

- OFF (turns Incremental mode OFF).
- ON (turns Incremental mode ON).

The default setting is ON for this mode in both the HP ITG development environment and in a generated program. When Incremental mode is ON, HP ITG sends only the specific commands needed to put the instrument into the state recalled by the `hpt_recall` function.

Use this function to turn Incremental mode OFF if you have changed the instrument's setup manually or with I/O program statements. HP ITG cannot track such instrument setting changes. When Incremental mode is OFF, HP ITG sends all of the commands needed to put the instrument into the state recalled by the `hpt_recall` function.

Return Value

`hpt_incremental` returns a 0 if an error condition exists.

Error Conditions

A return value of 0 indicates the following error condition:

- *inst_desc* does not match any assigned instruments in the workfile.

See Also

`hpt_assign`, `hpt_recall`

Example

```
/* Turn on Incremental mode */
hpt_incremental(inst_descriptor, ON);
```

hpt_init

Syntax `int hpt_init(`*workfile*`);`

`char far *`*workfile*	Path name for soft test system workfile.

Description

Note A call to the `hpt_init` function is required when you use a workfile, and the program calls HP ITG functions affecting the instruments in that workfile. This function must be called in the program before any other HP ITG function. You can generate this code automatically from the HP ITG Editor window's menu bar. Click on `Edit`, then `Generate Initialization Code`. Be sure you have saved the workfile, and the editor cursor is at the beginning of the file.

The `hpt_init` function names the current workfile for HP ITG and initializes the data used by the rest of HP ITG. If the argument for *workfile* does not specify an absolute path name, the current directory is used to find the file name. The file name used as the argument for *workfile* does not need to include the `.WF` extension.

Return Value `hpt_init` returns a non-zero value if the workfile is loaded successfully. The function returns a 0 if the workfile could not be loaded.

Error Conditions A return value of 0 indicates the following error conditions:

- *workfile* does not exist in the current or specified directory.
- *workfile* file name exists but is not a valid workfile.

See Also `hpt_assign`

Example

```
hpt_init("WORK.WF");
```

hpt_livemode

Syntax `int hpt_livemode(`*inst_desc, switch*`);`

`int` *inst_desc*	Instrument descriptor assigned by **`hpt_assign`**.
`int` *switch*	Boolean switch turns Live mode ON or OFF.

Description The `hpt_livemode` function turns Live mode ON or OFF for the instrument specified by *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, **`hpt_assign`**. *switch* is a Boolean value that controls Live mode. Use the following constants as arguments for *switch:*

- OFF (turns Live mode OFF).
- ON (turns Live mode ON).

The default setting is ON for this mode in a program. This means all instruments addressed in your program will answer commands as they are received. To eliminate interface bus activity for a particular instrument, use `hpt_livemode` to turn Live mode OFF for that instrument while you are developing other parts of your program.

Return Value `hpt_livemode` returns a 0 if the operation failed.

Error Conditions A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *switch* is not a legal value.

See Also hpt_assign

Example

```
/* Turn on Live mode */
hpt_livemode(inst_descriptor, ON);
```

hpt_local

Syntax `int hpt_local(`*inst_desc*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.

Description The `hpt_local` function puts the instrument specified by *inst_desc* into local mode, which prevents the program's I/O commands from controlling the instrument. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

Return Value `hpt_local` returns a 0 if it could not set the instrument to local mode.

Error Conditions A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- The instrument does not respond to the interface bus.

See Also `hpt_assign`, `hpt_remote`

Example

```
hpt_local(inst_descriptor);
```

hpt_mem_info

Syntax

`int hpt_mem_info(`*inst_desc, conv, ems*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`int far *`*conv*	Receives return variable.
`int far *`*ems*	Receives return variable.

Description

The `hpt_mem_info` function reports the amount of conventional (*conv*) and expanded (*ems*) memory being used by the instrument driver, states, and configuration information specified by *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

Return Value

`hpt_mem_info` returns the amount of conventional memory (in Kbytes) being used by an instrument driver in *conv*. Typically, this is between 2-3 Kbytes depending on the instrument driver size. The function returns the amount of expanded memory (in Kbytes) being used by an instrument driver in *ems*. Expanded memory usage is expressed as a multiple of 16 Kbytes, even if less memory is actually used. A 0 is returned if memory usage could not be reported.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- The function could not read the current memory usage.

See Also hpt_assign, hpt_close, hpt_close_all

Example

```
hpt_mem_info(instr_descriptor, &conv_mem_usage, &ems_mem_usage);
```

hpt_monitor

Syntax `int hpt_monitor(`*inst_desc, mode, filename*`);`

`int` *inst_desc*	Instrument descriptor assigned by **hpt_assign**.
`int` *mode*	Controls Monitor mode status.
`char far *`*filename*	Specifies destination for debugging information.

Description The **hpt_monitor** function controls HP ITG's Monitor mode for program debugging. The information type specified by *mode* is written to *filename* during program execution. All of the resulting debugging information applies only to the instrument specified by *inst_desc.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, **hpt_assign**.

To report results more immediately, set *filename* to a printer port. As an example, if a printer is configured to the computer's LPT1 port, use `"LPT1:"` for *filename.*

You can set the Monitor mode status using the following arguments for *mode:*

- OFF (Turn Monitor mode OFF).
- ON (Log I/O transactions to *filename*).
- DRIVER_DEBUG (Log I/O transactions and additional information to *filename*).

The default setting is OFF for this mode in a program. This means HP ITG will not produce any debugging information while your program is running. Use this function to turn Monitor mode on for debugging purposes.

Return Value

`hpt_monitor` returns a 0 if it could not change the Monitor mode status.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *filename* could not be opened to receive debugging information.
- *mode* is not one of the allowed values.

See Also

`hpt_assign`, `hpt_close`, `hpt_close_all`

Example

```
/*  Send information to line printer */

hpt_monitor(inst_descriptor, DRIVER_DEBUG, "LPT1:");
```

hpt_peek, hpt_peek2

Syntax

Standard Version

`int hpt_peek(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`double far *`*val*	Receives return value.

Version 2

`int hpt_peek2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`double far *`*val*	Receives return value.

Description

The `hpt_peek` function returns the current *val* of the INTEGER or CONTINUOUS *comp* for the instrument specified by *inst_desc.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`. The returned value is read directly from the results of the most recent call to the `hpt_get` or `hpt_poke` functions.

The `hpt_peek2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

Return Value

hpt_peek and hpt_peek2 return the current value of the instrument's component in *val.* Both functions return a 0 if error conditions prevent them from returning the component value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc.*
- *comp* is not an INTEGER or CONTINUOUS type component.
- *comp_desc* is not a valid component descriptor.

See Also

hpt_assign, hpt_assigncomp, hpt_get, hpt_poke

Example

```
/*  Standard version  */
double voltage;
.
.
.
hpt_peek(inst_descriptor, "VOLTAGE", &voltage);
printf("Voltage currently is %lfV\n",voltage);

/*  Version 2  */
double voltage;
voltage_comp = hpt_assigncomp(inst_descriptor, "VOLTAGE");
.
.
.
hpt_peek2(voltage_comp, &voltage);
printf("Voltage currently is %lfV\n",voltage);
```

hpt_peek_ary, hpt_peek_ary2

Syntax

Standard Version

`int hpt_peek_ary(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`double far *`*val*	Receives returned array contents.

Version 2

`int hpt_peek_ary2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`double far *`*val*	Receives returned array contents.

Description

The `hpt_peek_ary` function returns the current *val* of the RARRAY *comp* for the instrument specified by *inst_desc.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`. The returned value is read directly from the results of the most recent call to the `hpt_get_ary` function.

The program must allocate sufficient space for *val.* Use `hpt_compdims` to read the array component's dimensions.

The `hpt_peek_ary2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

Return Value

`hpt_peek_ary` and `hpt_peek_ary2` return the current value of the instrument's component in *val.* Both functions return a 0 if error conditions prevent them from returning the component value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc.*
- *comp* is not an RARRAY type component.
- *comp_desc* is not a valid component descriptor.

See Also

`hpt_assign`, `hpt_assigncomp`, `hpt_compdims`, `hpt_get_ary`

Example

```
/*  Standard version  */
double reading[20];
.
.
.
hpt_peek_ary(inst_descriptor, "READING", reading);
for (i = 0; i < 20; i++)
     printf("Reading %d currently is %lf\n",i,reading[i]);

/*  Version 2  */
double voltage[20];
voltage_comp = hpt_assigncomp(inst_descriptor, "VOLTAGE");
.
.
.
hpt_peek_ary2(voltage_comp, voltage);
for (i = 0; i < 20; i++)
     printf("Voltage %d currently is %lfV\n",i,voltage[i]);
```

hpt_peek_iary, hpt_peek_iary2

Syntax

Standard Version

`int hpt_peek_iary(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`int far *`*val*	Receives returned array contents.

Version 2

`int hpt_peek_iary2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`int far *`*val*	Receives returned array contents.

Description

The `hpt_peek_iary` function returns the current *val* of the IARRAY *comp* for the instrument specified by *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`. The returned value is read directly from the results of the most recent call to the `hpt_get_iary` function.

The program must allocate sufficient space for *val*. Use `hpt_compdims` to read the array component's dimensions.

The `hpt_peek_iary2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

Return Value

`hpt_peek_iary` and `hpt_peek_iary2` return the current value of the instrument's component in *val.* Both functions return a 0 if error conditions prevent them from returning the component value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc.*
- *comp* is not an IARRAY type component.
- *comp_desc* is not a valid component descriptor.

See Also

`hpt_assign`, `hpt_assigncomp`, `hpt_compdims`, `hpt_get_iary`

Example

```
/*  Standard version  */
int ireading[10];
.
.
.
hpt_peek_iary(inst_descriptor, "IREADING", ireading);
for (i = 0; i < 10; i++)
     printf("IREADING %d currently is %d\n",i,ireading[i]);

/*  Version 2  */
int   count[5];
count_comp = hpt_assigncomp(inst_descriptor, "COUNT");
.
.
.
hpt_peek_iary2(count_comp, count);
for (i = 0; i < 5; i++)
     printf("Count %d currently is %dV\n",i,count[i]);
```

hpt_peek_str, hpt_peek_str2

Syntax

Standard Version

`int hpt_peek_str(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`char far *`*val*	Receives return value.

Version 2

`int hpt_peek_str2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`char far *`*val*	Receives return value.

Description

The `hpt_peek_str` function returns the current *val* of the STRING *comp* for the instrument specified by *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`. The returned value is read directly from the results of the most recent call to the `hpt_get_str` function.

Be sure to dimension the string *val* large enough to hold the returned value. Strings can have a maximum length of 256 bytes.

The `hpt_peek_str2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

Return Value

`hpt_peek_str` and `hpt_peek_str2` return the current value of the instrument's component in *val.* Both functions return a 0 if error conditions prevent them from returning the component value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc.*
- *comp* is not a STRING type component.
- *comp_desc* is not a valid component descriptor.

See Also

`hpt_assign`, `hpt_assigncomp`, `hpt_get_str`, `hpt_poke_str`

Example

```
/*  Standard version  */
char  error_string[100];
.
.
.
hpt_peek_str(inst_descriptor, "ERROR_STRING", error_string);
printf("Instrument reports \"%s\"\n",error_string);

/*  Version 2  */
char error_string[100];
string_comp = hpt_assigncomp(inst_descriptor, "ERROR_STRING");
.
.
.
hpt_peek_str2(string_comp, error_string);
printf("Instrument reports \"%s\"\n",error_string);
```

hpt_poke, hpt_poke2

Syntax

Standard Version

`int hpt_poke(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`double` *val*	New component value.

Version 2

`int hpt_poke2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`double` *val*	New component value.

Description

The `hpt_poke` function sets the internal variable of the CONTINUOUS or INTEGER *comp* to *val* for the instrument specified by *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`. `hpt_poke` changes only the component's value, not the instrument.

The `hpt_poke2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

Return Value

hpt_poke and hpt_poke2 return a 0 if error conditions prevent them from setting the component's value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc*.
- *comp* is not an INTEGER or CONTINUOUS type component.
- *comp_desc* is not a valid component descriptor.

See Also

hpt_assign, hpt_assigncomp, hpt_set

Example

```
/*  Standard version  */
hpt_poke(inst_descriptor, "FREQUENCY", 1000.0);

/*  Version 2  */
double frequency;
frequency_comp = hpt_assigncomp(inst_descriptor, "FREQUENCY");
.
.
.
hpt_poke2(frequency_comp, 1000.0);
```

hpt_poke_ary, hpt_poke_ary2

Syntax

Standard Version

`int hpt_poke_ary(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`double far *`*val*	New component value.

Version 2

`int hpt_poke_ary2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`double far *`*val*	New component value.

Description

The `hpt_poke_ary` function sets the internal variable of the RARRAY *comp* to *val* for the instrument specified by *inst_desc.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`. `hpt_poke_ary` sets only the component's value, not the instrument.

The program must allocate sufficient space for *val.* Use `hpt_compdims` to read the array component's dimensions.

The `hpt_poke_ary2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

Return Value

`hpt_poke_ary` and `hpt_poke_ary2` return a 0 if error conditions prevent them from setting the component's value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc*.
- *comp* is not an RARRAY type component.
- *comp_desc* is not a valid component descriptor.

See Also

`hpt_assign`, `hpt_assigncomp`, `hpt_compdims`, `hpt_set_ary`

Example

```
/*  Standard version  */
double points[40];
.
.
.
hpt_poke_ary(inst_descriptor, "POINTS", points);

/*  Version 2  */
double points[40];
points_comp = hpt_assigncomp(inst_descriptor, "POINTS");
.
.
.
hpt_poke_ary2(points_comp, points);
```

hpt_poke_iary, hpt_poke_iary2

Syntax

Standard Version

`int hpt_poke_iary(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`int far *`*val*	New component value.

Version 2

`int hpt_poke_iary2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`int far *`*val*	New component value.

Description

The `hpt_poke_iary` function sets the internal variable of the IARRAY *comp* to *val* for the instrument specified by *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`. `hpt_poke_iary` changes only the component's value, not the instrument.

The program must allocate sufficient space for *val*. Use `hpt_compdims` to read the array component's dimensions.

The `hpt_poke_iary2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

Return Value

hpt_poke_iary and hpt_poke_iary2 return a 0 if error conditions prevent them from setting the component's value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc*.
- *comp* is not an IARRAY type component.
- *comp_desc* is not a valid component descriptor.

See Also

hpt_assign, hpt_assigncomp, hpt_compdims, hpt_set_iary

Example

```
/*  Standard version  */
int points[40];
.
.
.
hpt_poke_iary(inst_descriptor, "POINTS", points);

/*  Version 2  */
int points[40];
points_comp = hpt_assigncomp(inst_descriptor, "POINTS");
.
.
.
hpt_poke_iary2(points_comp, points);
```

hpt_poke_str, hpt_poke_str2

Syntax

Standard Version

`int hpt_poke_str(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`char far *`*val*	New component value.

Version 2

`int hpt_poke_str2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`char far *`*val*	New component value.

Description

The `hpt_poke_str` function sets the internal variable of the STRING *comp* to *val* for the instrument specified by *inst_desc.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`. `hpt_poke_str` changes only the component's value, not the instrument.

The `hpt_poke_str2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

Return Value

hpt_poke_str and hpt_poke_str2 return a 0 if error conditions prevent them from setting the component's value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc*.
- *comp* is not an STRING type component.
- *comp_desc* is not a valid component descriptor.

See Also

hpt_assign, hpt_assigncomp, hpt_set_str

Example

```
/*  Standard version  */
char name[] = "STRING";
.
.
.
hpt_poke_str(inst_descriptor, "INST_STRING", name);

/*  Version 2  */
char name[] = "STRING";
string_comp = hpt_assigncomp(inst_descriptor, "INST_STRING");
.
.
.
hpt_poke_str2(string_comp, name);
```

hpt_push, hpt_push2

Syntax

Standard Version

`int hpt_push(`*inst_desc, button*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*button*	Button component name from instrument driver.

Version 2

`int hpt_push2(`*button_desc*`);`

`long` *button_desc*	Button descriptor assigned by `hpt_assigncomp`.

Description

The `hpt_push` function executes a set of actions associated with the component named, *button,* for the instrument specified by *inst_desc.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

The `hpt_push2` function produces the same results. As a Version 2 function, the parameter *button_desc* replaces *inst_desc* and *button* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

Return Value

`hpt_push` and `hpt_push2` return a 0 if error conditions prevent them from executing the set of actions.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *button* does not exist in the instrument driver.
- *button_desc* is not a valid component descriptor.

See Also

hpt_assign, hpt_assigncomp

Example

```
/*  Standard version  */
hpt_push(voltmeter, "RESET");

/*  Version 2  */
button_comp = hpt_assigncomp(inst_descriptor, "RESET");
.
.
.
hpt_push2(button_comp);
```

hpt_recall, hpt_recall2

Syntax

Standard Version

`int hpt_recall(`*inst_desc, state*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*state*	Instrument state name to be recalled.

Version 2

`int hpt_recall2(`*state_desc*`);`

`long` *state_desc*	State descriptor assigned by `hpt_assignstate`.

Description

The `hpt_recall` function recalls the *state* for the instrument specified by *inst_desc.* The instrument components are set according to the recalled instrument state. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

The `hpt_recall2` function produces the same results. As a Version 2 function, the parameter *state_desc* replaces *inst_desc* and *state* for higher performance. Its value can be assigned by the functions, `hpt_assignstate` and `hpt_state_save`.

Return Value

`hpt_recall` and `hpt_recall2` return a 0 if error conditions prevent them from recalling the instrument state.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *state* does not exist for the instrument.
- *state_desc* is not a valid descriptor.
- An HP-IB error occurred.

See Also

hpt_assign, hpt_assignstate, hpt_state_save

Example

```
/*  Standard version  */
hpt_recall(inst_descriptor, "INIT_STATE");

/*  Version 2  */
init_state = hpt_assignstate(inst_descriptor, "INIT_STATE");
.
.
.
hpt_recall2(init_state);
```

hpt_remote

Syntax

`int hpt_remote(`*inst_desc*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.

Description

The `hpt_remote` function puts the instrument specified by *inst_desc* into remote mode. An instrument in remote mode can be controlled by a program's I/O commands. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

Return Value

`hpt_remote` returns a 0 if the function could not put the instrument into remote mode.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- An HP-IB error occurred.

See Also

`hpt_assign`, `hpt_local`

Example

```
hpt_remote(inst_descriptor);
```

hpt_set, hpt_set2

Syntax

Standard Version

`int hpt_set(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by **`hpt_assign`**.
`char far *`*comp*	Component name from instrument driver.
`double` *val*	New value for component and instrument.

Version 2

`int hpt_set2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by **`hpt_assigncomp`**.
`double` *val*	New value for component and instrument.

Description

The **`hpt_set`** function sets the CONTINUOUS or INTEGER *comp* and the instrument specified by *inst_desc* to *val*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, **`hpt_assign`**.

The **`hpt_set2`** function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, **`hpt_assigncomp`**.

In the HP ITG development environment, HP ITG automatically generates a call to **`hpt_set`** when you adjust an instrument's soft panel control. To use **`hpt_set2`**, you have to call **`hpt_assigncomp`** and edit the call to **`hpt_set`** to change the parameter list's arguments.

Return Value

hpt_set and hpt_set2 return a 0 if error conditions prevent them from setting the component and instrument to the value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc*.
- *comp* is not a CONTINUOUS or INTEGER type component.
- *comp_desc* is not a valid component descriptor.
- An HP-IB error occurred.

See Also

hpt_assign, hpt_assigncomp, hpt_poke

Example

```
/*  Standard version  */
hpt_set(inst_descriptor, "FREQUENCY", 1.5E6);

/*  Version 2  */
freq_comp = hpt_assigncomp(inst_descriptor, "FREQUENCY");
hpt_set2(freq_comp, 1.5E6);
```

hpt_set_ary, hpt_set_ary2

Syntax

Standard Version

`int hpt_set_ary(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`double far *`*val*	New value for component and instrument.

Version 2

`int hpt_set_ary2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`double far *`*val*	New value for component and instrument.

Description

The `hpt_set_ary` function sets the RARRAY *comp* and the instrument specified by *inst_desc* to *val.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

The program must allocate sufficient space for *val.* Use `hpt_compdims` to read the array component's dimensions.

The `hpt_set_ary2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

Return Value

hpt_set_ary and hpt_set_ary2 return a 0 if error conditions prevent them from setting the component and instrument to the value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc*.
- *comp* is not an RARRAY type component.
- *comp_desc* is not a valid component descriptor.
- An HP-IB error occurred.

See Also

hpt_assign, hpt_assigncomp, hpt_compdims, hpt_poke_ary

Example

```
/*  Standard version  */
double cont_array[SIZE] = {0};
.
.
.
hpt_set_ary(inst_descriptor, "REAL_DATA", cont_array);

/*  Version 2  */
double cont_array[SIZE] = {0};
array_comp= hpt_assigncomp(inst_descriptor, "REAL_DATA");
.
.
.
hpt_set_ary2(array_comp, cont_array);
```

hpt_set_error_handler

Syntax

`int hpt_set_error_handler(`*function*`);`

`int (*`*function*`)()`	Pointer to user-written error handler function.

Description

The `hpt_set_error_handler` function sets the error handler for HP ITG functions. This overrides the default handler which prints the error message to STDERR and exits the program.

Return Value

`hpt_set_error_handler` returns a 0 if the function could not change the program's error mode.

Error Conditions

A return value of 0 indicates the following error condition:

- *function* is not a valid function.

See Also

`hpt_errmsg`

Example

```
/*  User-written error handler function  */
int handler() {
char buffer[256];
     hpt_errmsg(buffer);
     fprintf(stderr, "%s\n",buffer);
     return 0;
}
.
.
.
/*  Program initialization code goes here  */
.
.
.
hpt_set_error_handler(handler);
```

hpt_set_iary, hpt_set_iary2

Syntax

Standard Version

`int hpt_set_iary(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`int far *`*val*	New value for component and instrument.

Version 2

`int hpt_set_iary2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`int far *`*val*	New value for component and instrument.

Description

The `hpt_set_iary` function sets the IARRAY *comp* and the instrument specified by *inst_desc* to *val.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

The `hpt_set_iary2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

The program must allocate sufficient space for *val.* Use `hpt_compdims` to read the array component's dimensions.

Return Value hpt_set_iary and hpt_set_iary2 return a 0 if error conditions prevent them from setting the component and instrument to the value.

Error Conditions A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc*.
- *comp* is not an IARRAY type component.
- *comp_desc* is not a valid component descriptor.
- An HP-IB error occurred.

See Also hpt_assign, hpt_assigncomp, hpt_compdims, hpt_poke_iary

Example

```
/*  Standard version  */
int int_array[SIZE] = {0};
.
.
.
hpt_set_iary(inst_descriptor, "INT_DATA", int_array);

/*  Version 2  */
int int_array[SIZE] = {0};
array_comp= hpt_assigncomp(inst_descriptor, "INT_DATA");
.
.
.
hpt_set_iary2(array_comp, int_array);
```

hpt_set_str, hpt_set_str2

Syntax

Standard Version

`int hpt_set_str(`*inst_desc, comp, val*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`char far *`*val*	New value for component and instrument.

Version 2

`int hpt_set_str2(`*comp_desc, val*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`char far *`*val*	New value for component and instrument.

Description

The `hpt_set_str` function sets the STRING or DISCRETE *comp* and the instrument specified by *inst_desc* to *val.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

The `hpt_set_str2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

In the HP ITG development environment, HP ITG automatically generates a call to `hpt_set_str` when you adjust an instrument's soft panel control. To use `hpt_set_str2`, you have to call `hpt_assigncomp` and edit the call to `hpt_set_str` to change the parameter list's arguments.

Return Value

`hpt_set_str` and `hpt_set_str2` return a 0 if error conditions prevent them from setting the component and instrument to the value.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc.*
- *comp* is not a STRING or DISCRETE type component.
- *comp_desc* is not a valid component descriptor.
- An HP-IB error occurred.

See Also

`hpt_assign`, `hpt_assigncomp`, `hpt_poke_str`

Example

```
/*  Standard version  */
char *string = "STRING";
.
.
.
hpt_set_str(inst_descriptor, "STRING_DATA", string);

/*  Version 2  */
char *string = "STRING";
string_comp = hpt_assigncomp(inst_descriptor, "STRING_DATA");
.
.
.
hpt_set_str2(string_comp, string);
```

hpt_setdevaddr

Syntax

`int hpt_setdevaddr(`*inst_desc, address*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*address*	New address for HP ITG instrument configuration.

Description

The `hpt_setdevaddr` function changes the HP ITG instrument configuration specified by *inst_desc* to the new *address.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

Return Value

`hpt_setdevaddr` returns a 0 if the function could not change the address configuration.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *address* is not a valid instrument address.

See Also

`hpt_add_device`, `hpt_assign`, `hpt_devaddr`, `hpt_devsubad`

Example

```
hpt_setdevaddr(inst_descriptor, "703");
```

hpt_setstate, hpt_setstate2

Syntax

Standard Version

`int hpt_setstate(`*inst_desc, comp, state*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*comp*	Component name from instrument driver.
`int` *state*	Component's new status.

Version 2

`int hpt_setstate2(`*comp_desc, state*`);`

`long` *comp_desc*	Component descriptor assigned by `hpt_assigncomp`.
`int` *state*	Component's new status.

Description

The `hpt_setstate` function lets you change the status of *comp* for the instrument specified as *inst_desc* to *state.* The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

Use one of the following constants as the argument for *state:*

- COMP_VALID (component is VALID).
- COMP_INVALID (component is INVALID).
- COMP_DONTCARE (component is DONTCARE).

The `hpt_setstate2` function produces the same results. As a Version 2 function, the parameter *comp_desc* replaces *inst_desc* and *comp* for higher performance. Its value is assigned by the function, `hpt_assigncomp`.

Return Value

hpt_setstate and hpt_setstate2 return a 0 if error conditions prevent them from changing the component's status.

Error Conditions

A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *comp* does not exist in the instrument driver specified by *inst_desc*.
- *state* is not a valid value.
- *comp_desc* is not a valid component descriptor.

See Also

hpt_assign, hpt_assigncomp, hpt_forget, hpt_getstate

Example

```
/*  Standard version  */
hpt_setstate(inst_descriptor, "FREQUENCY", COMP_VALID);

/*  Version 2  */
frequency_comp = hpt_assigncomp(inst_descriptor, "FREQUENCY");
.
.
.
hpt_setstate2(frequency_comp, COMP_VALID);
```

hpt_state_save

Syntax `int hpt_state_save(`*inst_desc, state*`);`

`int` *inst_desc*	Instrument descriptor assigned by `hpt_assign`.
`char far *`*state*	State name to create and save.

Description The `hpt_state_save` function lets you create a new instrument state named *state* for the instrument specified as *inst_desc*. The instrument descriptor is the integer that identifies a specific instrument in your workfile, and is returned by the function, `hpt_assign`.

Return Value `hpt_state_save` returns an internal number (state descriptor) corresponding to the new state. The state descriptor can be used in calls to `hpt_recall2`. The function returns a 0 if it could not create the new state.

Error Conditions A return value of 0 indicates the following error conditions:

- *inst_desc* does not match any assigned instruments in the workfile.
- *state* could not be created.

See Also `hpt_assign`, `hpt_assignstate`, `hpt_recall`, `hpt_recall2`

Example

```
state_descriptor = hpt_state_save(inst_descriptor, "NEWSTATE");
```

A

Menus Index

Overview

HP ITG is controlled through commands contained in pull-down menus. Menus are grouped into three sets:

- System menus

  The system menu names are displayed in HP ITG's System menu bar located across the top of the window. Click on a menu name, and HP ITG displays either a pull-down menu from which you can select commands that control system operations, or a dialog box. The box on the left side of the title bar (above the System menu bar) also contains a window system menu.

- Editor menus

  The editor menu names are displayed across the top of HP ITG's Editor window. Click on a menu name or the box in the left corner, and HP ITG displays pull-down menus from which you can select commands that control window and Editor operations.

- Instrument panel menu

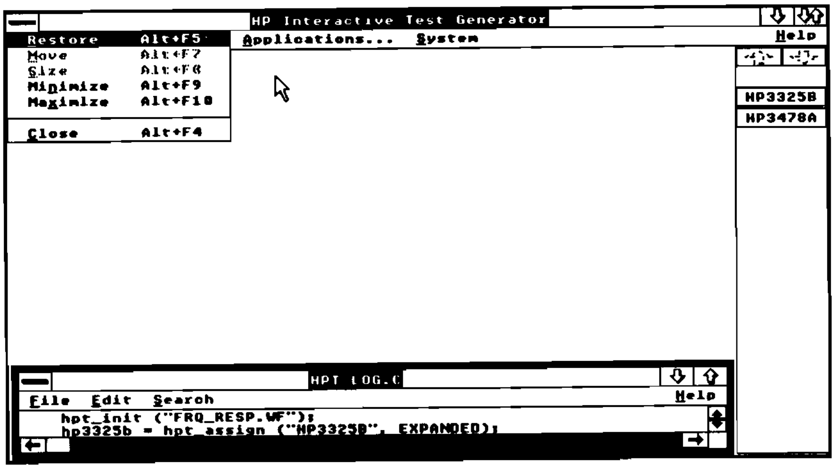
  Click on the box in the top left corner of the instrument panel so HP ITG displays the panel menu. It contains commands that control panel operation.

Note Menu commands that appear grayed cannot be used.

System Menus

These menus control HP ITG operation as well as the soft test systems you create.

System Box

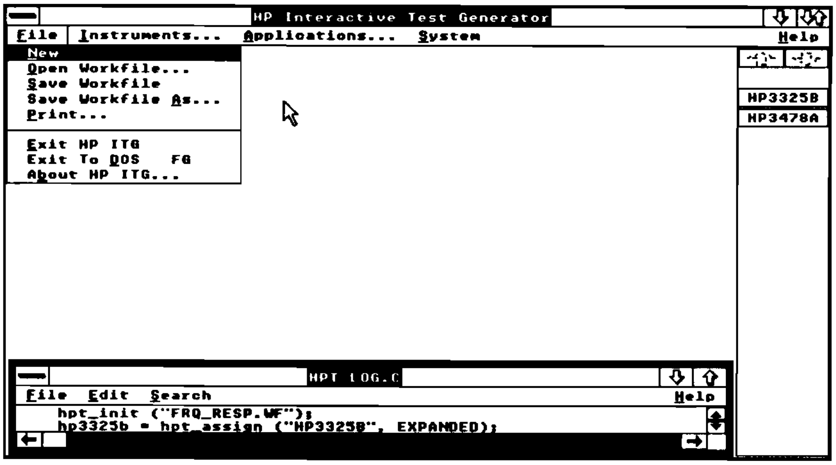


```
    hpt_init ("FRQ_RESP.WF");
    hp3325b = hpt_assign ("HP3325B", EXPANDED);
```

Figure A-1. The System Box Pull-Down Menu

- `Restore` sets the window back to its previous size.
- `Move` lets you reposition the window.
- `Size` lets you resize the window.
- `Minimize` lets you reduce the window to an icon.
- `Maximize` lets you enlarge the window to its maximum size.
- `Close` exits the window application.

File



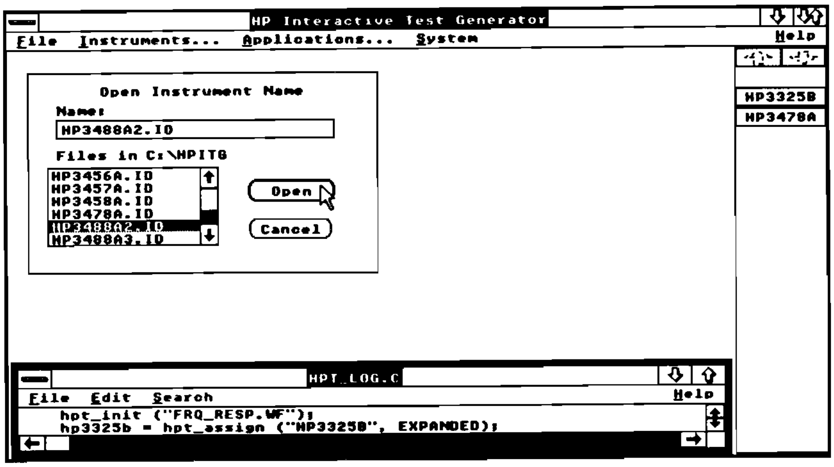
```
hpt_init ("FRQ_RESP.WF");
hp3325b = hpt_assign ("HP3325B", EXPANDED);
```

Figure A-2. The System File Pull-Down Menu

- `New` clears the test system from the screen and lets you build a new soft test system.
- `Open Workfile` ... lets you open an existing workfile. HP ITG displays the instruments and applications that constitute the soft test system contained in the workfile.
- `Save Workfile` replaces the existing test system file with the current test system data. All workfile file names include the `.WF` extension.
- `Save Workfile As` ... saves the current soft test system's instruments and their states in the designated file. A dialog box prompts you for a file name.
- `Print` ... prints the contents of the screen.
- `Exit HP ITG` exits the HP ITG program. This command is available only if you are using the full Microsoft Windows application.

- `Exit To DOS` exits the Microsoft Windows graphics environment that supports HP ITG and returns to MS-DOS.
- `About HP ITG ...` displays copyright information, version number, and memory usage. (For more detailed expanded memory usage information, run the `HPEMSTAT.EXE` program from the MS-DOS command line.)

Instruments ...



Open Instrument Name

Name: HP3488A2.ID

Files in C:\HPITG

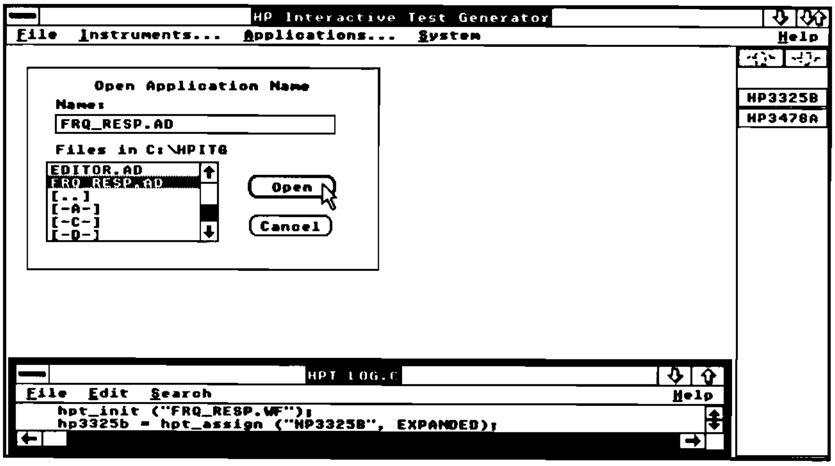
HP3456A.ID
HP3457A.ID
HP3458A.ID
HP3478A.ID
HP3488A2.ID
HP3488A3.ID

```
hpt_init ("FRQ_RESP.WF");
hp3325b = hpt_assign ("HP3325B", EXPANDED);
```

Figure A-3. The Instrument List Box

Clicking on this command displays a list box of all available instrument drivers. You add an instrument to your test system by selecting a driver and providing instrument configuration information. Instrument drivers must use the `.ID` file name extension to be listed here.

Applications . . .

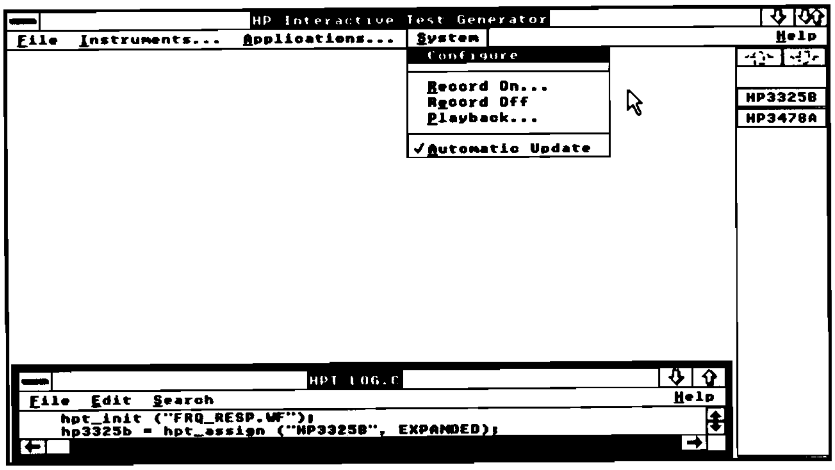


```
hpt_init ("FRQ_RESP.WF");
hp3325b = hpt_assign ("HP3325B", EXPANDED);
```

Figure A-4. The Applications List Box

This menu displays a list box with all installed applications. You add an application to your test system by selecting an application and providing configuration information. Application drivers must use the .AD file name extension to be listed here.

System



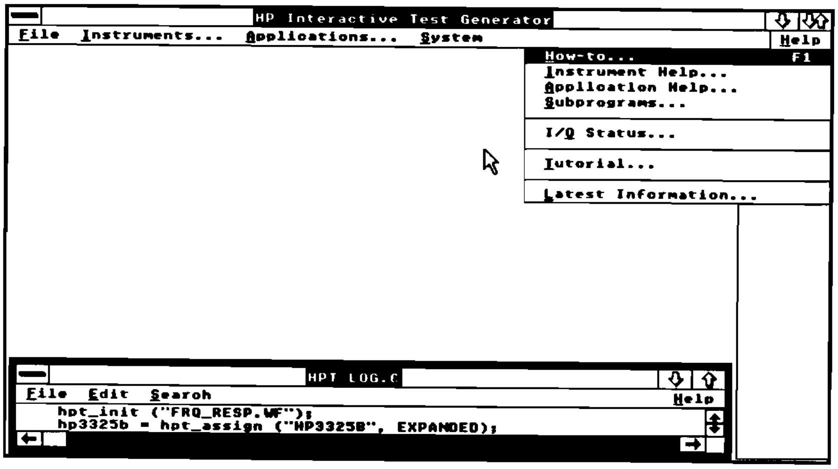
```
    hpt_init ("FRQ_RESP.WF");
    hp3325b = hpt_assign ("HP3325B", EXPANDED);
```

Figure A-5. The System Pull-Down Menu

- `Configure` lets you select and configure system entities such as a printer.
- `Record On ...` turns on HP ITG's Record mode. Clicking on `Record On ...` displays a dialog box. Enter the file name for the script file using the `.DS` extension, and click on `OK`. HP ITG then begins recording your interactions in this script file.
- `Record Off` turns off HP ITG's Record mode. All the movements you made with the mouse after you turn on Record mode until you turn it off are saved in the file you created.
- `Playback ...` lets you select from the `.DS` files in the current directory so that you can see the movements that were recorded and saved.
- `Automatic Update` controls the Automatic Update mode for instruments in your soft test system. Click on the menu command to turn the mode on or off. A check mark beside the command means the mode

is turned on. The mode must also be controlled for each instrument in the soft test system. Click on the instrument panel menu box, then Modes ...

Help



```
    hpt_init ("FRQ_RESP.WF");
    hp3325b = hpt_assign ("HP3325B", EXPANDED);
```

Figure A-6. The Help Pull-Down Menu

- How-to ... describes how to perform common tasks with HP ITG.
- Instrument Help ... provides a quick reference on all instrument panels installed with HP ITG.
- Application Help ... provides a quick reference on the available applications.
- Subprograms ... provides the syntax and a brief description of the subprograms in the HP ITG Library for the programming language you are using.
- I/O Status ... displays the select code setting for each IEEE-488 interface installed in your computer.
- Tutorial ... demonstrates HP ITG's basic operation.

- `Latest Information` ... lists the instrument drivers currently available with HP ITG, and provides information about HP ITG features added after this handbook was printed.

Editor Menus

These menus control your interaction with the contents of the HP ITG Editor window and the editor available from `Applications` ... The HP ITG Editor displays the current program file name in the center of its title bar. `HPT_LOG` is the default file name with a `.C` or `.BAS` extension, depending on the programming language you are using. The default application editor file name is `SCRATCH.TXT`.

Box Menu

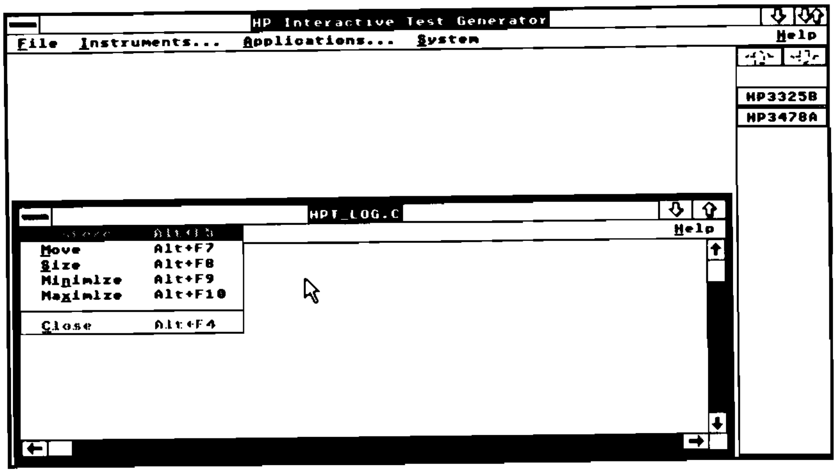


Figure A-7. The Editor Box Pull-Down Menu

- **Restore** sets the Editor window back to its previous size.
- **Move** lets you reposition the Editor window in the work area.
- **Size** lets you resize the Editor window.
- **Minimize** reduces the Editor window to an icon, placing it in the icon area at the far right of the HP ITG window.
- **Maximize** enlarges the Editor window to full size.
- **Close** closes the Editor window, removing it from the work area. This works only in the editor application window.

File

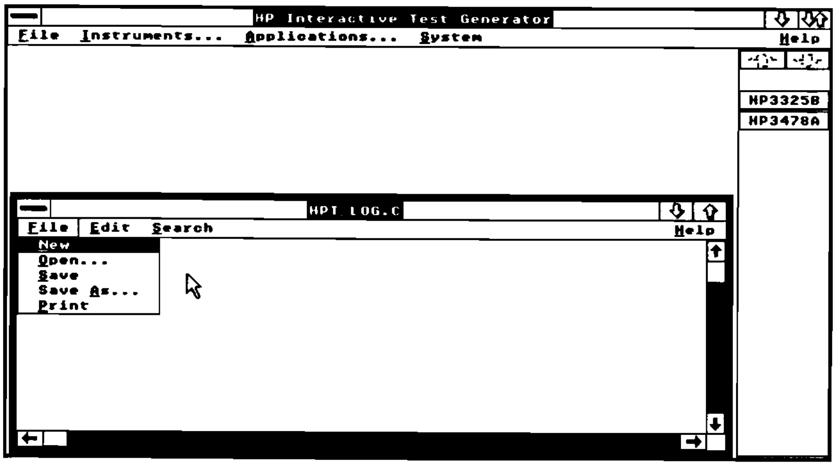


Figure A-8. The Editor File Pull-Down Menu

- **New** deletes all text in the Editor window, and changes the current file name to `untitled`.

- Open ... loads a file into the Editor window. To select a file, click on the file name in the list box and click on OK.
- Save replaces the file that was opened with the current contents of the Editor window.
- Save As ... saves the contents of the Editor window in the designated file. A dialog box prompts you for the file name.
- Print prints the contents of the Editor window.

Edit

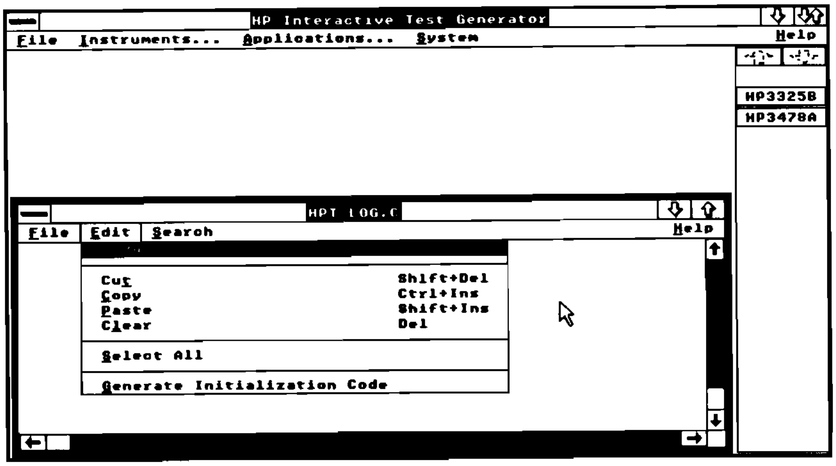


Figure A-9. The Edit Pull-Down Menu

- Undo cancels the last edit made if you undo the change immediately after the edit.
- Cut removes selected text from the edit window and places it in a storage buffer. You can Paste this text in another location in the edit window.

- Copy copies the selected text into a storage buffer. You can Paste this text in another location in the edit window.
- Paste places text from the storage buffer in the current location of the edit cursor. Text is placed in the storage buffer using Cut or Copy.
- Clear deletes all selected text from the edit window but retains the file name.
- Select All selects all text in the file whether it currently appears in the edit window or not. You can then Cut, Copy, or Clear the text.
- Generate Initialization Code generates the initialization code for all instruments in the current test system that have Log HP ITG Calls mode on.

Search

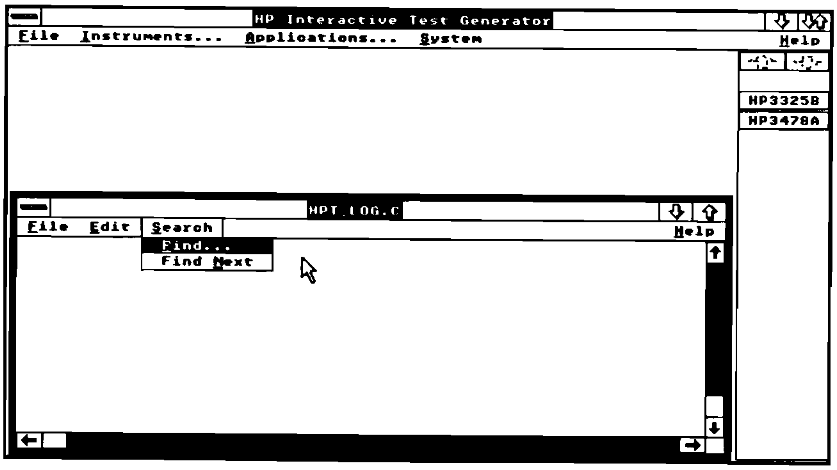
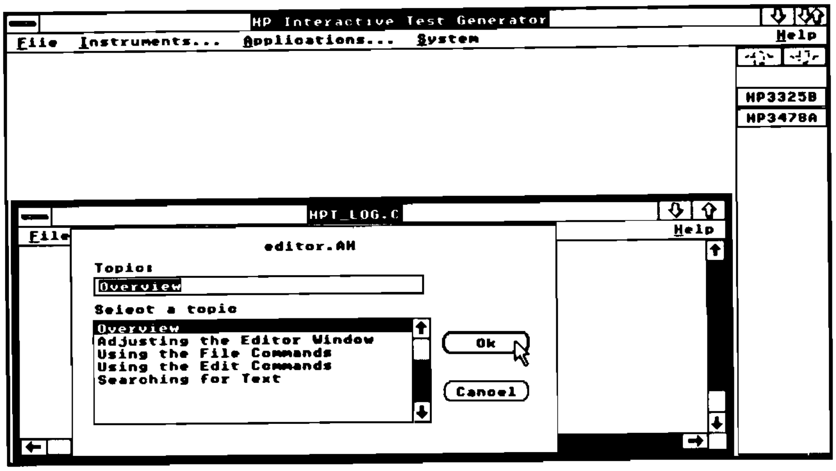


Figure A-10. The Search Pull-Down Menu

- `Find ...` lets you search the current file for a number, word, or phrase starting from the cursor's current position. A dialog box lets you specify the criteria including the search direction and capitalization. HP ITG puts the cursor at the beginning of the first occurrence it finds.
- `Find Next` lets you repeat the previous search operation to find the next occurrence.

Help


HP Interactive Test Generator
File Instruments... Applications... System
Help
HP3325B
HP3478A
HPT_LOG.C
File
Help
editor.AH
Ok
Cancel

Topic:
Overview

Select a topic
Overview
Adjusting the Editor Window
Using the File Commands
Using the Edit Commands
Searching for Text

Figure A-11. The Editor Help List Box

Clicking on this command displays a list box listing topics that describe the Editor operation. Click on a topic, then click on `OK` to display the information.

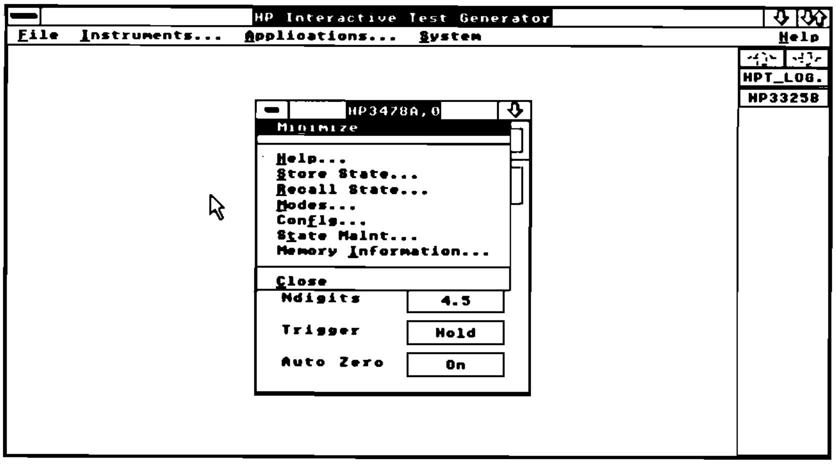
Arrows

- Clicking on the up arrow increases the Editor window to full screen.
- Clicking on the down arrow reduces the Editor window to an icon.
- When the Editor window is full screen, clicking on the bidirectional arrow icon reduces the Editor window from full screen to partial screen.

Instrument Panel Menu

The instrument panel title bar contains the panel menu box, the panel name and address, and an arrow.

Instrument Panel Box



Ndigits 4.5

Trigger Hold

Auto Zero On

Figure A-12. The Instrument Panel Pull-Down Menu

Click on the panel menu box in the top left corner of the panel and HP ITG displays the panel menu.

- `Minimize` reduces the soft panel to an icon.

- **Help ...** displays a list box with topics of information about the instrument and its driver.
- **Store State ...** saves the current panel setup as a single instrument state. HP ITG prompts you for a name. If the name selected already exists, the old data are overwritten with the current data.
- **Recall State ...** recalls an instrument state from the stored set of states.
- **Modes ...** displays a dialog box with the different modes that can be toggled on/off for the individual instrument panel. A mode is on when an **X** is displayed in the box next to the given mode. The modes include:

`Incremental Recall`	When on, Incremental Recall minimizes the number of commands sent to an instrument to cause it to go from one state to the next recalled state.
`Error Checking`	When on, Error Checking checks the instrument for errors whenever a control is set or a measurement is made. Not all instruments have an Error Checking mode.
`Live`	When on, Live mode sends HP-IB commands directly to the instrument, allowing you to control an instrument through HP ITG.
`Log HP ITG Calls`	When on, Log HP ITG Calls generates calls to HP ITG subprograms that you can use as the basis of a measurement procedure. HP ITG displays this code in the Editor window.

`Automatic Update`	When on, Automatic Update lets an instrument that provides a display to continuously update that display with measurement readings. The mode name is grayed if the instrument driver does not support Automatic Update. To start updating a panel's display, click on the display, then click on it again to stop updating. You must also turn on **`Automatic Update`** (in the System menu) for the HP ITG development environment.

- **`Config ...`** displays a dialog box containing configuration information for the instrument, including its HP ITG name, HP-IB address, card cage subaddress, and timeout.

  The **`Name`** and **`Address`** you select for the instrument are displayed in the middle of the panel's title bar.

  `Timeout` is the amount of time allowed for the instrument to respond to commands in the development and run-time environments. The default time is 30 seconds.

- **`State Maint ...`** displays a dialog box that allows you to further control states (that is, in addition to Store State and Recall State). The dialog box presents the following options:

`Open`	This lets you open a state file whether created in the current test system or in a different test system.

`Save`	You can save one or all of an instrument's stored states from the current soft test system into a state file. The name you create for the state file must end with the `.SL` file name extension. You can use this state file in other test systems.
`Add`	You can add one or all of an instrument's stored states to an existing state file. State files are created using the `Save` command.
`Delete`	You can delete a stored state from your current soft test system. It does not affect states saved in a state file.
`Print`	You can obtain a printout of the values HP ITG has stored for one or all of an instrument's stored states in your current soft test system
`Cancel`	This works like all other HP ITG `Cancel` buttons. Clicking on it removes the dialog box from the screen and no action is taken.

- `Memory Information`... displays the amount of conventional and expanded memory being used by the instrument driver, states, and configuration information. The values are rounded to the nearest 1 Kbyte.
- `Close` closes the panel, which removes it from the current test system. Closing the panel deletes all the stored states associated with it, but does not affect the states saved in a state file.

Arrow Clicking on the arrow in the upper right corner reduces the panel to an icon, and places it in the icon area at the far right of the HP ITG work area.

B

I/O Interfaces

Overview

HP ITG currently supports the following IEEE-488 and VXI interfaces:

- Hewlett-Packard's HP-IB (Models HP 82335A and HP 82990A)
- National Instruments Corporation's GPIB-PCII and GPIB-PCIIA
- Radix MicroSystems, Inc. VXIbus

Your test system may need two interface boards. One board should be reserved for instrument control. Use the other to support computer peripherals such as printer, plotter, or external disk drive.

Note If you install more than one interface board in your computer, each board requires a separate select code.

Hewlett-Packard HP-IB Interface

The HP-IB interface is addressed using the following numeric scheme:

SDA[SA]

Where:

S = Select code of the HP-IB interface (HP ITG supports select codes 3 - 7).

DA = Device Address of instrument or mainframe (allowed range is 00 - 29).

SA = Optional Secondary Address for module (allowed range is 00 - 31).

With this scheme, you can control instruments from HP ITG through an HP-IB board. You can use more than one HP-IB interface using the range of addresses listed in the following table for each supported select code. Select code 7 is the default switch setting and is recommended for HP ITG. If another interface in your computer is using that code, use 6.

Select Code	Address [Secondary Address] Range	PC Address Segment
3	300[00] - 329[31]	CC00
4	400[00] - 429[31]	D000
5	500[00] - 529[31]	D400
6	600[00] - 629[31]	D800
7	700[00] - 729[31]	DC00

Caution

The LIM 4.0 EMS driver may conflict with address space for HP-IB select codes 3 - 7. See the installation guide for your HP-IB interface for information about switching settings to prevent address space conflict.

Changing the HP-IB Configuration

The configuration switches for the HP-IB interface shipped with your HP ITG are factory set for select code 7. If you need information about changing the configuration switches and installing the interface, see the guide, “Installing the HP-IB Interface” that comes with the interface. Read the sections, “Setting Switches,” “Installing the Interface,” and “Connecting Peripherals.”

National Instruments Corporation GPIB-PCII/IIA

The National Instruments Corporation GPIB-PCII and GPIB-PCIIA interfaces are addressed using the following numeric scheme:

SDA[SA]

Where:

S = Select code of the GPIB-PCII/IIA interface (HP ITG supports select codes 8 and 9).

DA = Device Address of instrument or mainframe (allowed range is 01 - 30).

SA = Optional Secondary Address for module (allowed range is 00 - 31).

With this scheme, you can control instruments from HP ITG through a GPIB-PCII/IIA interface. You can use more than one GPIB-PCII/IIA interface using the following range of addresses for each supported select code:

Select Code	Port Address	Address [Secondary Address] Range
8	0x2B8	801[00] - 830[31]
9	0x2C0	901[00] - 930[31]

You should use select code 8 for the first GPIB-PCII/IIA interface you install in your computer. Use select code 9 for the second interface. See the GPIB-PCII/IIA interface's installation manual for details on setting the port address.

Radix MicroSystems, Inc. EPC-2 System

The EPC-2 system from Radix MicroSystems, Inc. uses an alpha-numeric addressing scheme for instruments using the VXIbus interface standard. When the EPC-2 is turned on, it assigns default names to any VXI instruments it recognizes.

The default name is **vdev** followed by unique digits representing the instrument's position in the card cage. If there are several VXI instruments in the card cage, the EPC-2 assigns the name **vdev0** to the first instrument, **vdev1** to the second, and so on. HP ITG uses these addresses to control the VXI instruments. You can view and change the default addresses by using the Radix MicroSystems Start-up Resource Manager program, SURM.EXE.

The EPC-2 system also provides an interface compatible with National Instruments Corporation's GPIB-PCII interface. HP ITG supports that interface at select code 8, but only with the EPC-2 system designated as *Controller in Charge.*

C

Expanded and Extended Memory

Overview

The type of computer you use, whether i286-based or i386-based, affects the way you configure the additional memory in the computer and the memory manager you use to control it. Chapter 3 in this handbook, "Installing HP ITG," explains the basic installation requirements for HP ITG running in the single application environment (SAE) for Microsoft Windows. This appendix explains memory configuration information for development systems that may use the complete Windows application.

i286-Based Computer

The complete Windows version for an i286-based computer is Windows/286. Whether you use Windows/286 or continue using the SAE, the additional memory in the computer should remain configured as expanded, and use the same memory manager shipped with the memory board.

i386-Based Computer

The complete Windows version for an i386-based computer is Windows/386. For the SAE, the additional memory (usually preinstalled) should be configured as extended memory and use the memory manager shipped with it.

If you use Windows/386, you will need to remove the original memory manager. Windows/386 provides its own memory manager. It is important that you remove any other memory manager, or Windows/386 will not run correctly. The memory itself should remain configured as extended memory. By using the Windows/386 memory manager, you gain the advantage of running HP ITG and other applications in the Windows/386 multitasking environment.

Creating a Script

HP ITG provides a way for you to record and playback your interactions with HP ITG. Interactions include keystrokes as well as the movements you make with the mouse. This lets you demonstrate a measurement or procedure.

To create a script:

1. Turn on Record mode.
   a. Click on `System` on the System menu bar.
   b. Click on `Record On` ...
   c. Enter a file name using the extension `.DS`, then click on `OK`. HP ITG now records every interaction, saving the data in the file you specified.
2. Perform the procedure.
3. Turn off Record mode.
   a. Click on `System` on the System menu bar.
   b. Click on `Record Off`.
4. Play back the script.
   a. Click on `System` on the System menu bar.
   b. Click on `Playback` ...
   c. Click on a file name, then click on `Open`; or type in a file name, and press (Enter).

Messages

Overview

The HP ITG messages described in this appendix may appear while you use HP ITG, QuickC, or QuickBASIC environments; or during run-time. In HP ITG, the error number appears with the message. In QuickC, QuickBASIC, and run-time, the HP ITG subprogram name is also included to help identify the statement where the problem exists.

For reference information about HP ITG subprograms identified in the messages, see chapter 14, "The HP ITG Library: QuickBASIC" or chapter 15, "The HP ITG Library: C." For reference information about instrument driver components and their allowable values, see the component summary tables for the instrument drivers in HP ITG's online Help system under `Instrument Help` ...

Conventions

The following initials at the beginning of each message explanation help identify where the error can occur:

- *D* means the error can occur in the HP ITG *D*evelopment environment.
- *R* means the error can occur while *R*unning your program in QuickC, QuickBASIC, or run-time.
- *ID* means the problem described by the message exists in the *I*nstrument *D*river file. The error can occur in *D* or *R*.

1000: `Undefined component` *comp_name* `in device` *dev_name*

R—A call to an HP ITG subprogram cannot access the component named in the parameter list. Verify the component name spelling and that the component exists in the instrument driver identified in the message.

1001: `Device descriptor` *number* `is invalid`

R—Your program is using an unrecognized device descriptor in a call to an HP ITG subprogram. Check your program code to ensure `hpt_assign` is called to assign the descriptor and open the device, and that you are using the correct device descriptor.

1002: `Component descriptor` *number* `is invalid`

R—Your program is using an unrecognized component descriptor number in a call to a Version 2 HP ITG subprogram. Either the descriptor has not been assigned by the subprogram, `hpt_assigncomp`, or the device is closed. Check your program code to ensure `hpt_assigncomp` is called and you are using the correct component descriptor. Verify that the device is open.

1003: `State` *number* `invalid for component` *comp_name*

R—The `hpt_setstate2` subprogram uses an incorrect value for the state parameter. Be sure your program uses one of the following constants for the status parameter:

COMP_VALID
COMP_INVALID
COMP_DONTCARE

These constants are defined in the include files, HPITG.H (for C) and HPITG.BI (for QuickBASIC). Be sure the correct file is included in your program.

1004: Improper type for component ***comp_name*** **in device** ***dev_name***

R—Your program calls a subprogram that does not match the type of component specified in the instrument driver. Revise your program to call the correct subprogram for the component type.

1005: Undefined discrete value ***val_name*** **for component** ***comp_name*** **in device** ***dev_name***

R—Your program calls a subprogram to change a component value, but the value does not exist in the driver. Verify the value name's existence and spelling.

1006: Undefined discrete value ***number*** **for component** ***comp_name*** **in device** ***dev_name***

R—Your program calls the HP ITG subprogram, `hpt_set`, using an invalid discrete number for a discrete component. Check your program to ensure that `hpt_assignparm` is called and you are using the correct value name.

1009: Inconsistent array dimensions ***number number*** **for** ***comp_name*****, should be** ***number number***

R—This occurs in QuickBASIC programs only. Double-array dimension in call to an HP ITG subprogram is inconsistent with the component's dimensions defined in the instrument driver. Verify that array dimensions are not transposed in your program and they match the component's defined array size. Use the `hpt_compdims` subprogram to determine the component's array dimensions.

1010: `You need to Install and Setup a printer using the Control program`

D—HP ITG could not find the following line in the Microsoft Windows initialization file, WIN.INI, which configures a printer:

```
DEVICE= ...
```

To configure a printer in HP ITG, click on **`System`** on the system menu bar, then click on **`Configure`**. See chapter 6, "Creating and Using a Soft Test System," for information about printer configuration.

1011: `Out of disk space during the print`

D—HP ITG ran out of hard disk space while writing temporary files during printing. Remove unneeded files to make room on your hard disk.

1012: `Out of memory during the print`

D—HP ITG ran out of conventional memory while printing. To provide adequate memory, quit running other Microsoft Windows applications, close some HP ITG devices, or reconfigure your computer to use less conventional memory during startup.

1013: `Error during print`

D—A general error occurred during printing which was caused by one of several conditions. Check your printer for a paper jam or for enough paper supply. Be sure your printer is properly connected and configured. If the problem is not with the printer, you may have run out of hard disk space or conventional memory.

1014: `You must close the Spooler application before dumping to the HP PaintJet printer`

D—When HP ITG performs a screen dump to an HP PaintJet printer, data is sent directly to the printer, bypassing the printer spooler. To prevent an output conflict, close the spooler application running in Microsoft Windows. You may have to wait for it to finish printing a file.

1015: `A floating point number is required`

D—HP ITG expects a valid floating point number such as 1.2 or 12. Do not use a comma for the radix.

1016: `Could not load device -- too many active windows`

D—HP ITG ran out of conventional memory while creating the windows for an instrument driver. Quit running some other Microsoft Windows applications, close some HP ITG devices, or reconfigure your computer to use less conventional memory on startup.

1017: `File not found:` *file_name*

D—You tried to open a nonexistent file in the development environment's work space or HP ITG Editor. In a dialog box, if you type in a file name for an instrument driver, workfile, or program, verify the file name spelling and directory path.

1018: `A name is required`

D—Each instrument driver loaded into the HP ITG workfile must have a name. Be sure to enter a name in the Instrument Configuration dialog box when you add an instrument driver.

1019: `Could not create file:` *file_name*

D—In the development environment, HP ITG could not create a program file, workfile, or instrument state file. Be sure that the filename meets MS-DOS requirements, and that there is no other file of the same name which is protected or locked by networking software.

1020: `Error writing to demo script file`

D—An error occurred writing to the demonstration script file. Be sure there is enough hard disk space.

1021: `Syntax error reading back demo script file`

D—There is a syntax error in your demonstration script file. Be sure that the file is a valid demonstration script file.

1022: `Error while playing demo file: could not find device` *dev_name*

D—A demonstration file requested a MOVE or PUSH relative to the given device, but no device with that name exists. Load a device of that name and rerun the script.

1023: `Error while playing demo file: menu command could not be executed`

D—A demonstration file with a PUSH ... MENU referenced a nonexistent menu item. Check the menu numbers given in the file.

1024: `Error while playing demo file: DIALOG keyword used without a dialog box`

D—A demonstration file tried to reference a dialog box, but no dialog box was present. Check the demonstration file.

1025: `Error while playing demo file: MINIMIZE or MAXIMIZE error`

D—A demonstration file tried to minimize or maximize a window which has no minimize or maximize arrows. For example, dialog boxes never have minimize or maximize arrows in HP ITG. Check the demonstration file.

1026: `Error while playing demo file: LISTBOX error`

D—A demonstration file tried to pick a string from a list box, but the string is not in the list box. Check the demonstration file.

1027: `Error while playing demo file: COMP error`

D—A demonstration file tried to reference the given component name in a device, but that device has no such component, or that component has no visible panel elements attached to it. Check the demonstration file.

1028: `Error while playing demo file: could not find a button labeled` *button_name*

D—A demonstration script file tried to click on the indicated button, but the window specified in the demonstration file did not contain the button. When writing a demonstration file, you must match spaces exactly; case does not matter. If the button label contains underline characters, precede the underlined characters with an ampersand. For example, to match Open, where the O is underlined, you must type &Open. Correct the demonstration script file.

1029: `This string cannot contain character values above 127:` *string*

D—A string must consist only of characters with decimal values equal to 127 or less.

1030: `Device` *dev_name* `not found`

R—The program calls the subprogram, `hpt_assign`, using a device name in the parameter list that does not match any name in the current workfile. Check the workfile in the development environment to verify the contents. The device name is specified in the Instrument Configuration dialog box when you add a device to the workfile.

1031: `Descriptor` *number* `does not refer to an open device`

R—The program uses an unrecognized device descriptor in a call to an HP ITG subprogram. Check the program to ensure `hpt_assign` is called at the beginning, and that the correct device descriptor is used. The `hpt_assign` subprogram assigns the descriptor and opens the device. This error can also occur if `hpt_close` or `hpt_close_all` are called prematurely.

1032: `Device table full (254 devices maximum)`

D or *R*—You have added more than 254 devices to the HP ITG environment's work space, or the program adds 255 devices using `hpt_add_device`.

1033: *file_name* `not found`

D—You tried to open a nonexistent file in the development environment's work space or HP ITG Editor. In a dialog box, if you type in a file name for an instrument driver, workfile, or program, verify the file name spelling and directory path.

1034: `Error reading` *file_name* `header`

D or *R*—The compiled instrument driver file exists, but contains an error in the file header. Recompile the instrument driver file using HPIDC.EXE.

1035: *file_name* `is not an HP ITG compiled instrument driver file`

D or *R*—The indicated file does not exist as a compiled instrument driver.

1036: Error reading *file_name* object size table.

D or *R*—The compiled instrument driver exists, but does not have a correct size table.

1038: *file_name* `is an old format HP ITG compiled instrument driver file`

D or *R*—A newer version HP ITG cannot read older version compiled instrument drivers. Recompile the old instrument driver source file using the most recent version of HPIDC.EXE.

1039: *file_name* `is incompatible with workfile` *file_name*

D or *R*—The compiled instrument driver named in this message is different from the compiled instrument driver referred to by the workfile.

1040: `Driver unsupported with this system's expanded memory configuration`

D or *R*—The expanded memory used is inadequate to support the large memory requirements of the instrument driver.

1041: `Device` *dev_name* `assigned before hpt_init` was successfully invoked

R—The program calls the `hpt_assign` subprogram before calling the `hpt_init` or `hpt_add_device` subprograms. Be sure `hpt_init` or `hpt_add_device` is called before `hpt_assign`.

1042: `Address` *io_address* `for device` *dev_name* `invalid`

D or *R*—The address for the instrument is outside the range for valid addresses. The range for HP-IB interfaces is 00-29. The range for National Instruments GPIB-PCII/IIA interfaces is 01-30. Check the instrument configuration in HP ITG for the address setting.

1044: `Could not run program` *file_name*

D—HP ITG found, but could not run, the instrument driver compiler or the Control program for printer configuration. If an instrument driver has been revised, HP ITG recompiles the source file using HPIDC.EXE when loading the instrument into the test system. When configuring a printer in HP ITG (click on `System`, then `Configure`), HP ITG runs CONTROL.EXE. Try to run the indicated program from the MS-DOS command line to verify that the program is not corrupted.

1045: `The program` *file_name* `was not found in the current PATH`

D—The indicated program file name could not be found. Be sure the file exists in the current directory. If the file is in a different directory, you should add that directory name to the PATH command in the AUTOEXEC.BAT file. If you change the AUTOEXEC.BAT, reboot the computer so the new command takes effect.

1046: `There is not enough memory to run program` *file_name*

D—HP ITG could not run the indicated program because insufficient memory was available. Reduce memory usage by closing unnecessary instrument panels and other windows. If insufficient memory remains a problem, add more memory.

1047: `Cannot find:` *search_string*

D—The HP ITG Editor cannot find any occurrence (or the next occurrence) of the search text. If you think the text does exist, verify that the spelling and search options are correct.

1048: `Editor out of memory`

D—The test program is larger than the HP ITG Editor's memory buffer can hold. This message can appear when you open an existing file or while you are entering text in the Editor window. Break the program into smaller modules.

1049: `Cannot open file` *file_name*

D—HP ITG cannot open the indicated file. Check the file name spelling, or if the file is write-protected.

1050: `Cannot create file` *file_name*

D—A file for the program you are trying to save cannot be created. Check for a full hard disk, an illegal file name, or a write-protected file.

1051: `Device` *dev_name* `reported error` *number*

D or *R*—The instrument reported the indicated error number during an error checking sequence. Look up the error number in the instrument's manual.

1053: `Floating point error in device` *dev_name*

ID—A floating point error occurred while executing an action list in the given device. This particular message indicates that problems exist in the instrument driver.

1054: `Floating point overflow in device` *dev_name*

ID—A number larger than 1.79769E+308 was generated while executing an action list in the instrument driver. The instrument driver's action list needs revision so numbers larger than this are not generated.

1055: `Division by zero in device` *dev_name*

ID—Division by zero occurred in the indicated instrument driver. The instrument driver needs revision to prevent such operations.

1056: `Floating point underflow in device` *dev_name*

ID—A number smaller than 2.22E-308 was generated while executing an action list in the instrument driver. The instrument driver's action list needs revision so numbers smaller than this are not generated.

1057: `HP ITG stack underflow in device` *dev_name*`, component` *comp_name*

ID—The instrument driver tried to pop a value off of the top of the HP ITG stack, but no item was there. The instrument driver is probably missing a FETCH statement and needs revision. This error is not related to MS-DOS stack allocations.

1058: `HP ITG stack overflow in device` *dev_name*`, component` *comp_name*

ID—The instrument driver tried to push a value onto the stack, but the stack was full. The stack holds only 20 items. Of these items, it can hold 15 floating point numbers and five 256-character strings. This error is not related to MS-DOS stack allocations.

1059: `Array indexing not permitted in device` *dev_name*, `component` *comp_name*

ID—The instrument driver tried to use an array component type where one is not allowed, such as in a non-array ENTER or OUTPUT statement. The instrument driver needs revision.

1060: `Value out of range in an OUTPUT TABLE statement in device` *dev_name*, `component` *comp_name*

ID—The OUTPUT TABLE statement used an index into the table that was too large. The instrument driver needs revision to add more items to the OUTPUT TABLE statement or generate smaller indexes.

1061: `Array component needed in device` *dev_name*, `component` *comp_name*

ID—Commands such as MATSCALE, OUTPUT array, or ENTER array, require an array component type. The instrument driver needs revision.

1062: `No CASE was matched in a SELECT statement in device` *dev_name*, `component` *comp_name*

ID—The SELECT statement in an instrument driver did not have any matching CASE statements. The instrument driver needs revision to add a CASE statement or a CASE ELSE statement.

1063: `Floating point overflow in device` *dev_name*, `component` *comp_name*

ID—A number larger than 1.79769E+308 was generated while executing an action list in the instrument driver. The action list in the instrument driver needs revision to avoid generating such values.

1064: Attempted LOG or LGT of nonpositive number in device ***dev_name*****, component** ***comp_name***

ID—The instrument driver tries to take the LOG or LGT of a nonpositive number. The instrument driver needs revision to prevent such operations.

1065: Wrong type of value on the top of the stack in device ***dev_name*****, component** ***comp_name***

ID—The instrument driver tried to perform some operation on the top of the stack using the wrong type of data. For example, a string cannot be subtracted from another string. The instrument driver needs revision to prevent such operations.

1066: Overflow converting a real value to an integer value in device ***dev_name*****, component** ***comp_name***

ID—A value outside the range between -32768 and 32767 was used as a 16-bit integer. The instrument driver code needs revision so it generates smaller numbers or uses a continuous component type to hold the value.

1067: Array index out of bounds in device ***dev_name*****, component** ***comp_name***

ID—An array component type was indexed with a row or column less than one or greater than the number of rows or columns declared in its TYPE statement. The array component TYPE statement or the array index operation in the instrument driver need revision.

1068: Division by zero in device ***dev_name*****, component** ***comp_name***

ID—Division by zero occurred in the indicated instrument driver's component. The instrument driver needs revision to prevent such operations.

1069: Invalid reference to component in device *dev_name*, **component** *comp_name*

ID—Hit actions or update actions were executed with no default component, yet the instrument driver referred to the current component, probably through SELF or DEFAULT. The instrument driver needs revision to remove the reference to the default component in the action list, or to make the action list a SET ACTIONS list.

1070: Improper type matching in device *dev_name*, **component** *comp_name*

ID—In a component, a string is used where a number is required, or a number is used where a string is required. For example, the skip field in an ENTER array cannot be a string component. The instrument driver needs revision.

1071: String overflow error in device *dev_name*, **component** *comp_name*

ID—In a component, a string with more than 256 characters was attempted. The instrument driver needs revision so all strings in the instrument driver are 256 characters long or less. This includes any temporary strings on the stack.

1072: Attempted SQRT of a negative number in device *dev_name*, **component** *comp_name*

ID—The square root operation was attempted on a negative number in a component. The instrument driver needs revision.

1073: Value outside of valid domain of function in device *dev_name*, **component** *comp_name*

ID—The arcsine or arccosine was attempted on a number less than -1 or greater than 1 in a component. The instrument driver needs revision.

1074: `Numeric field specifier is too large while trying to format` *number* `in device` *dev_name*

ID—A numeric format, such as DDD.DDD, specified more than 250 characters. The instrument driver needs revision so the specifier uses fewer characters.

1075: `Numeric field specifier is too small while trying to format` *number* `in device` *dev_name*

ID—A numeric format, such as DDD.DDD, specified fewer characters than were in the number, as in 7123.444. The instrument drivers needs revision to specify more characters in the format string or use the K format specifier.

1076: `Sign specifier missing from format while trying to format` *number* `in device` *dev_name*

ID—A numeric format such as DDD.DDD specified fewer characters than were in the number, eliminating the minus sign. The instrument driver needs revision to specify more characters or use the K format specifier.

1077: `Insufficient data for ENTER in device` *dev_name*

ID—A linefeed or EOI was received before the entire array was entered. The instrument driver may need revision to ensure that the format and array size in the ENTER array statement matches the data size from the instrument.

1078: `No ENTER terminator found in device` *dev_name*`. See the # specifier in FORMAT.`

ID—The data was entered, but no linefeed or EOI was received in the next 256 characters. The instrument driver needs revision to include the # format specifier. This specifier allows the ENTER statement in the instrument driver to terminate right after the data is entered.

1079: `Numeric data not received in device` *dev_name*

ID—When entering data for a numeric field, a comma, linefeed, or EOI was received before a number was received. The format used on the ENTER statement may need revision in the instrument driver.

1080: `Image specifier greater than dimensioned string length in device` *dev_name*

ID—An image specifier for a string demanded more characters than were allowed in the string component used in the ENTER statement of the instrument driver. The instrument driver needs revision to adjust the length of the string component, the FORMAT used in the ENTER statement, and the data from the instrument.

1081: `Exponent field specifier too small while trying to format` *number* `in device` *dev_name*

ID—Not enough exponent characters are specified to represent the number. The FORMAT in the OUTPUT statement needs revision in the instrument driver.

1082: `Size mismatch in array output or enter in device` *dev_name*

ID—The number of rows in an OUTPUT or ENTER array statement in the instrument driver does not match the declaration of the array component. The instrument driver needs revision so the number of rows in the OUTPUT or ENTER array statement matches either the rows declared in the array component, or the columns declared in the array component.

1083: `Overflow of I/O buffer in device` *dev_name*

ID—OUTPUT statements in the instrument driver caused the output buffer to fill up. The instrument driver needs revision so it includes FLUSH statements that cause the output to be sent to the device more often.

1084: `String value applied to a numeric format in device` *dev_name*

ID—An OUTPUT or ENTER statement in the instrument driver tried to output or enter a string component using a numeric format such as DDD. The OUTPUT and ENTER format strings may need revision in the instrument driver.

1085: `Numeric value applied to a string format in device` *dev_name*

ID—An OUTPUT or ENTER statement in the instrument driver tried to output or enter a numeric component using a string format such as 'AAA'. The OUTPUT and ENTER format strings may need revision in the instrument driver.

1086: Format string calls for more than 1 value to be output or entered in device *dev_name*

ID—A format string in an ENTER or OUTPUT statement in the instrument driver tried to use two or more values, such as K,K, which would output or enter two values. The instrument driver needs revision so the format string uses only one value.

1087: Attempt to enter a number with *number* characters in device *dev_name*, limit is *number*

ID—The instrument sent a very long number that overflowed the HP ITG enter buffer. The instrument driver needs revision so it uses a different format specifier, or commands the instrument to send the data in a different format.

1088: Device descriptor *number* invalid

R—The first parameter in a call to an HP ITG subprogram uses an incorrect device descriptor. Be sure to use the device descriptor assigned by `hpt_assign`.

1089: Descriptor *number* invalid

R—Calls to Version 2 HP ITG subprograms require preassigned component descriptors and state descriptors. Be sure the program includes calls to `hpt_assigncomp` to assign component descriptors, and `hpt_assignstate` to assign state descriptors.

1090: Device *number* not open

R—The program tries to access an instrument and its driver that are not open. Be sure your program calls the `hpt_assign` subprogram before accessing an instrument.

1096: `Device` *number* `still open`

R—The program calls the `hpt_assign` subprogram twice. Check the program for duplicate calls to `hpt_assign` for the same instrument, or call `hpt_close` to close the extra instrument.

1097: `Out of conventional memory`

D or *R*—The program requested allocation of conventional memory (the computer's 640 Kbytes of base memory), but insufficient conventional memory was available. Such a request can be made when loading an instrument driver. Reduce conventional memory usage by not installing drivers such as networking software. Increase expanded memory usage by specifying the expanded memory option in the `hpt_assign` subprogram when loading instrument drivers.

1098: `Out of expanded memory`

D or *R*—The program requested allocation of expanded memory but insufficient expanded memory was available. Such a request can be made when loading an instrument driver. Remember, once expanded memory is allocated, you must deallocate it to make it available. You can do this in HP ITG by closing unnecessary instrument panels. In your program, use the `hpt_close` or `hpt_close_all` subprograms. If the problem persists, you may need to add more expanded memory to your computer.

1099: `No expanded memory detected`

R—The program called the **`hpt_assign`** subprogram using the option to load an instrument driver into expanded memory, but no expanded memory manager was detected. If your computer contains expanded memory, check the hardware and software configurations. If your computer does not have expanded memory, you should add it, or only access conventional memory.

1101: `Out of statically allocated expanded memory for states`

D or *R*—The program requests more expanded memory for instrument states than has been allocated. This can happen if your computer contains the LIM 3.X version of the Expanded Memory Specification instead of LIM 4.0. HP ITG allocates fixed amounts of memory when using the LIM 3.X EMS. HP ITG dynamically allocates memory as needed when using the LIM 4.0 EMS.

1105: `State descriptor` *number* `invalid`

R—The program uses the **`hpt_recall2`** subprogram with an invalid descriptor for an instrument state. Be sure to use the **`hpt_assignstate`** subprogram in the program to assign a state descriptor for **`hpt_recall2`**.

1107: `Error writing state` *state_name* `in device` *dev_name* `to workfile` *file_name*

D—This error happens when trying to save a workfile with its associated states to a hard disk. If the hard disk is full, save the workfile to a floppy disk, then make space on the hard disk.

1108: `Error reading states for device` *dev_name* `from workfile` *file_name*

D or *R*—HP ITG could not read the instrument states in the indicated workfile. The file may be corrupted. Recreate the instrument states and resave the workfile.

1109: `State` *state_name* `not defined for device` *dev_name*

D or *R*—The program tried to recall a nonexistent state for an instrument. Verify the state names created for that instrument in the workfile.

1110: `Undefined component` *comp_name* `in state file`

D—HP ITG tried to use a nonexistent component name in the current state file for the workfile. Be sure to use the same instrument driver for which the state file was created.

1111: `Expected` *name*`, received` *name*

D—HP ITG could not find the state name in the state maintenance file. Verify the state name in the state maintenance file.

1112: `Illegal discrete value` *val_name* `for component` *comp_name*

D—HP ITG could not find the value name for the instrument driver's component in the state maintenance file. The state maintenance file is incompatible with the driver. Be sure to use the state file created for the particular driver.

1113: `Out of memory--could not allocate` *number* `bytes`

D or *R*—Insufficient conventional memory was available to load a workfile or load an instrument driver. Delete other applications or close some HP ITG instrument drivers.

1115: Overflow converting *number* to an integer value

ID—The indicated real number is larger than 32767 or smaller than -32768, so it could not be held as a 16-bit integer. This occurs most often with OUTPUT and ENTER of arrays as integers. The instrument driver needs revision of OUTPUT and ENTER statements for array components to prevent such overflows.

1116: Timeout of device *dev_name* -- Live mode disabled

D or *R*—The instrument, with Live mode enabled, was addressed but did not respond to the command within the timeout period. In the development environment, HP ITG disabled Live mode. Be sure the instrument is plugged in, turned on, connected to the computer, and in the correct mode. (For example, a multimeter requiring an external trigger will timeout if the external trigger never arrives.) Failing all that, change the instrument configuration timeout value in the instrument configuration dialog box.

1117: I/O failure of device *dev_name* -- Live mode disabled

D or *R*—The instrument, with Live mode enabled, was addressed but did not respond to the command. In the development environment, HP ITG disabled Live mode. This is a general I/O failure and can have many causes due to a system or instrument failure.

1118: Device *dev_name* not present -- Live mode disabled

D or *R*—HP ITG could not detect an instrument at the specified address. In the development environment, HP ITG disabled Live mode. The specified address is valid, but no instrument responded. Be sure that the instrument is plugged in, turned on, connected to the computer, set to the expected address, and in the correct mode.

1119: Address *io_address* of device *dev_name* invalid -- Live mode disabled

D or *R*—The instrument's address is the same as the interface address. In the development environment, HP ITG disables Live mode. Address 30 is reserved for HP-IB interfaces. Address 00 is reserved for National Instruments Corporation GPIB-PCII/IIA interfaces. Set the instrument to another address.

1120: Timeout value *number* for device *dev_name* is out of range -- Live mode disabled

D or *R*—The timeout value entered in the instrument configuration dialog box is either less than zero, or greater than 1000. In the development environment, HP ITG disables Live mode. A value of 0 disables timeouts and is valid for any interface, but should be used with caution. Timeout values up to 1000 seconds are acceptable. For Radix MicroSystems VXIbus devices, timeout values entered in the dialog box are ignored.

1122: `Interface card or driver not present for device` *dev_name* `-- Live mode disabled`

D or *R*—The interface address for an instrument uses the wrong select code. For example, the select code of 704 is 7. In the development environment, HP ITG disables Live mode. Be sure the interface select code is configured correctly. The valid select codes are:

HP-IB: 3, 4, 5, 6, or 7

GPIB-PCII/IIA: 8 or 9

If the address given is the name of a VXI instrument, this error indicates that the required interface driver, BIMGR.SYS, was not detected. This driver is available with the EPC-2 system from Radix MicroSystems, Inc. Make sure the following command is in your CONFIG.SYS file:

```
DEVICE=C:\EPCONNEC\BIMGR.SYS
```

If you add the command, be sure to reboot your computer.

1123: `Could not open workfile` *file_name*

R—The program tried to load a workfile during a call to the subprogram, `hpt_init`, but the workfile could not be opened. Be sure the workfile exists in the directory path and the file name spelling is correct.

1124: `Error reading header of workfile` *file_name*

D or *R*—The workfile's file name exists, but the file's header could not be read. The file may not be a valid workfile. You need to recreate the workfile.

1125: *file_name* `is not an HP ITG workfile`

D or *R*—The file name exists, but the header information indicated the file is not a valid workfile. You need to recreate the workfile.

1126: `Error reading device table in workfile` *file_name*

D or *R*—The device table in the workfile could not be read. The device table lists the instruments and their states saved in the workfile. The table may have been corrupted while saving the workfile. You need to recreate the workfile.

1127: `Error creating workfile` *file_name*

D—HP ITG could not resave an existing workfile or create a new workfile. The destination disk may be full. You may need to recreate the workfile if you cannot save the file to a different file name, directory, or disk.

1128: `Error writing header of workfile` *file_name*

D—HP ITG could create or access the file, but could not write data to it. Check to see if the file or disk is write-protected.

1129: `Error writing device table in workfile` *file_name*

D—HP ITG could create or access the file and write header data to it, but could not write the device table data. Check to see if the disk is full.

1130: `You must call hpt_3852_init before using USERSUB in device` *dev_name*`, component` *comp_name*

R—The program tried to run a user-defined subprogram contained in an HP 3852 instrument driver before calling `hpt_3852_init`. HP ITG can generate a call to `hpt_3852_init` as part of the HP ITG initialization code if an HP 3852 driver is saved in a workfile.

1131: `Error mode` *number* `invalid`

R—While running the program in QuickBASIC, the wrong error mode number was used. The number is set by the HP ITG subprogram, **`hptseterrormode`**. See the QuickBASIC documentation to resolve the QuickBASIC error. Check the program to verify that the error mode is set correctly.

1132: `Error in the ID--too many levels of recursion`

ID—A statement in a SET ACTIONS action list calls the component that contains the statement. The instrument driver needs revision to correct such recursion.

Glossary

applications

An application is anything you can control through an HP ITG soft panel that is not a standalone instrument. An application can be written using the HP ITG Instrument Driver Language to perform simulation, modeling, or data analysis. An application can act as a virtual instrument, combining the features of several other instruments to perform a specific measurement task such as collecting frequency response data.

development environment

The HP ITG development environment is displayed when you run HP ITG. The development environment includes the QuickC and QuickBASIC environments when used to continue program development. In the HP ITG environment you can control instruments directly by adjusting instrument panels, and you can generate code based on your interactions with the panels. In the QuickC and QuickBASIC environments, you can run the generated code to debug your program.

dialog box

A dialog box is displayed when you have selected a command that requires you to provide more information about what you want to do. For example, when you select `Config ...` in the panel menu, HP ITG displays the instrument configuration dialog box, which lets you modify the instrument configuration.

When a dialog box appears that has more than one edit field, press (Tab) until the edit cursor appears in the field where you want to enter information, or simply click on the desired field. The cursor, a vertical line, indicates where the text will appear. After completing a dialog box, press (Enter) or click on `OK`.

driver

An instrument driver is an ASCII text file written in the HP ITG Instrument Driver Language. This file determines how the instrument's panel will be displayed by HP ITG. It also contains the HP-IB commands that HP ITG uses to control the instrument.

Editor menu bar

The Editor menu bar is the second bar from the top in the HP ITG Editor window. The commands on the menu bar provide menus that let you control the HP ITG Editor.

Editor window

The HP ITG Editor window is normally displayed across the bottom of the HP ITG development environment. It can be expanded to full-screen, moved to other locations in the work area, and it can be stored as an icon. The code HP ITG generates is written to the Editor whether it is a window or an icon. You can edit this code, as well as insert and delete lines. The Editor window is controlled through the menus available on the Editor menu bar that spans the top of its window.

HP ITG Library

The HP ITG Library is a set of subprograms provided with HP ITG. When an instrument panel is in the Log HP ITG Calls mode, HP ITG generates calls to these subprograms based on your interactions with the panel. When you run your program that is based on this code, you must link the HP ITG Library with your program.

Language-specific libraries are provided for the programming languages HP ITG supports. The libraries are installed when you run the HP ITG Setup program.

HP ITG Quick Library

The HP ITG Quick Library is a version of the HP ITG Library designed for use in the QuickBASIC environment during program development. The library file is installed when you run the HP ITG Setup program.

icon

An icon is a representation of a panel or editor window. The icons are displayed along the right side of the HP ITG development environment. You click on an icon to expand it to full size. You can scroll through the icons by clicking on the up/down arrows at the top of the icon area.

incremental state programming

HP ITG tracks an instrument's current state, then sends the minimum number of commands to put the instrument into the next state. This is called incremental state programming.

instrument state

The instrument drivers contain information that sets panel controls to specific values. Different states can be created and saved individually. A program can call each state instead of changing each control. Use the `Store State ...` command in the instrument panel menu.

list box

A list box provides a list of choices. File names are listed by their names, while drives and directories are indicated with brackets (example: `[-C-]` means drive `C`). The parent directory is indicated with two periods within brackets (`[..]`). Use the vertical scroll bar to see more choices in the list.

There are three ways to make a selection from a list box:

1. Click on your choice, then click on `Open` or press (Enter).
2. Double-click (click twice quickly) on your choice.
3. Click on the `Name` box, type in your choice, then press (Enter).

measurement procedure

A measurement procedure refers to the instrument-control code you can generate using HP ITG. Typically, a measurement procedure consists of recalling stored states, adjusting the controls of the panels, and making a measurement.

overwrite

Overwrite means to replace the information in a file (or a state) with new information.

pointer

The HP ITG pointer is the arrow that is displayed in the development environment. You can move this pointer by moving the mouse.

run-time environment

The run-time environment includes the system used to run a completed program. The system should contain the following files:

- The executable program.
- The HP ITG error file, `HPITG.ERR`.
- The workfile (if used by program).
- The instrument driver files.

soft panel

A soft panel is a representation of an instrument's programming language. HP ITG uses information in an instrument's driver to display a soft panel in the HP ITG development environment. You can control an instrument and generate code that controls the instrument by interacting with this soft panel.

soft test system

A soft test system is the set of instrument panels and their states that you have used in the HP ITG development environment. You can save any number of systems with different combinations of panels. Soft test systems are saved in workfiles.

state library

A state library is a file that contains one or more instrument states. A state library file can be used in any soft test system that can use a state stored in the file. Use the `State Maint ...` command in the instrument panel menu.

subpanel

A subpanel is a panel within a panel. Most instrument panels consist of many subpanels.

System menu bar

The System menu bar is the second bar from the top in the HP ITG development environment. The commands on the System menu bar provide menus that let you control HP ITG and create a soft test system.

timeout

Timeout is the amount of time HP ITG waits for an instrument to respond when controlling it from the HP ITG, QuickC, or QuickBASIC environments; or from an executable program.

work area

This is the area in the HP ITG environment, below the System menu bar, where instrument panels are added.

workfile

A workfile is a file in which HP ITG saves a soft test system. Workfiles should end with the `.WF` file name extension.

Index

HEWLETT PACKARD

Reorder Number
E2020-90000
Printed in U.S.A

E2020-90600
Manufacturing Number