Pascal 2.1 Procedure Library User's Manual

for HP Series 200 Computers

Manual Part No. 09826-90075
Microfiche No. 09826-99075

Hewlett-Packard Desktop Computer Division
3404 East Harmony Road, Fort Collins, Colorado 80525

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

March 1982...Preliminary Printing
May 1982...First Edition
December 1982...Update to include Pascal 2.0.
January 1983...Second Edition (incorporated December 1982 update).
May 1983...Update to include Pascal 2.1.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Table of Contents

Chapter 5: I/O Procedure Library

Chapter 6: Directing Data Flow

The Procedure Library Language Reference

Chapter 7: Outputting Data

Chapter 8: Inputting Data

Chapter 9: Registers

Chapter 10: Errors and Timeouts

Chapter 11: Advanced Transfer Techniques

Chapter 12: The HP-IB Interface

The Procedure Library Language Reference

Chapter 1
Getting Started

Introduction

This manual describes the procedures and functions provided with Pascal 2.0 LIBRARY. The manual is divided into two major sections. The first section (chapters 1 thru 15) is organized by topics. It explains particular programming concepts rather than individual procedures and functions. The second section, the Library Reference, is an alphabetical listing of the individual procedures and functions showing syntax and giving an explanation for each.

The I/O chapters contain detailed programming techniques information. The Graphics chapter is limited in scope. It will be very helpful if you have some backround in graphics programming. The chapter provides graphic device information and some example programs.

This chapter contains three sections. The first is the manual overview which describes the following chapters. The second section describes the LIBRARY as it is shipped to you. The third section explains step-by-step how to add modules to the LIBRARY. It also has a table which shows the module dependencies. For example, "If I must import module C so I can use procedure X, must I also import modules A and/or B?".

Manual Overview

Chapter 2: LIF Procedures. LIF file handling capabilities have been included in the Pascal 2.0 filing system. LIF procedures are no longer included in the Procedure Library.

Chapter 3: Graphics Procedures. The Graphics Library contains the fundamental procedures (primitives) that allow Pascal to communicate with most HP graphic devices. This chapter introduces the viewing transformation and provides example programs that illustrate the use of the graphics library. Sections are included for each device that is supported by Pascal. Summaries of error codes and deviations from HP 1000 conclude the chapter.

Chapter 4: Interfacing Concepts. This chapter presents a brief explanation of relevant interfacing concepts and terminology. This discussion is especially useful for beginners as it covers much of the why and how of interfacing. Experienced programmers may also want to skim this material to better understand the terminology used in this manual.

Chapter 5: The I/O Procedure Library. This chapter presents an introduction to the I/O Procedure Library. This discussion includes the organization of the library, major capabilities, and an introduction into the use of the library. All readers should read the information presented in this chapter.

Chapter 6: Directing Data Flow. This chapter describes how to specify which computer resource is to send data to or receive data from the computer. The use of device specifiers and interface select codes is discussed.

Chapter 7: Outputting Data. This chapter presents methods of outputting data to devices. All details of this process are discussed. Examples of free field and formatted output are given. You may be able to skip sections of this chapter, depending on your application.

Chapter 8: Inputting Data. This chapter presents methods of inputting data to devices. All details of this process are discussed. Examples of free field and formatted input are given. You may be able to skip sections of this chapter, depending on your application.

Chapter 9: Registers. This chapter describes the use and access of interface registers. Both the hardware and firmware registers are described. The individual interface registers are discussed in the corresponding interface chapter.

Chapter 10: Errors and Timeouts. This chapter describes what you need to do in order to handle and recover from error and timeout conditions.

Chapter 11: Advanced Transfer Techniques. This chapter discusses the high-performance transfer techniques provided in the I/O Library. These techniques are called buffered transfers and include interrupt, fast handshake, and direct memory access (DMA) data transfer mechanisms.

Chapter 12: The HP-IB Interface. This chapter describes programming techniques specific to the HP-IB interface. Details of HP-IB communications processes are also included to promote better overall understanding of how this interface may be used. This discussion is valid for the built-in HP-IB interface and for the optional HP 98624A HP-IB interface.

Chapter 13: The Serial Data Communications Interface. This chapter describes programming techniques specific to the HP 98628A serial data communications interface.

Chapter 14: The GPIO Interface. This chapter describes programming techniques specific to the HP 98622 GPIO interface.

Chapter 15: The Serial Interface. This chapter is a programming techniques discussion of the HP 98626 Serial Interface.

An Introduction to the LIBRARY

The LIBRARY is a collection of modules which contain support procedures and functions for your programs. In order to use these procedures and functions, you must IMPORT the modules which contain them. The Pascal Compiler looks for imported modules in the LIBRARY (and files named in the $SEARCH$ compiler directive). The LIBRARY must be on-line when you compile **and** when you run your program. For more information on modules and the IMPORT statement, read the Compiler chapter in the Pascal User's Manual.

The LIBRARY You Received

Beginning with Pascal 2.0, only a few modules exist in the LIBRARY. If you have moved or plan to move the LIBRARY to a new system volume on a hard disc or a Shared Resource Management system, you should add to the LIBRARY all the modules in IO, GRAPHICS and INTERFACE (on the LIB: disc). If you are using a 3.5-inch, 5.25-inch or 8-inch flexible disc as the system volume, then you should add only the modules you need (in order to save space on the system volume).

When Pascal 2.0 is shipped to you, the following modules are contained in LIBRARY.

RND -- The random number generator.

HPM -- The heap management utilities.

UIO -- The UCSD Pascal "unit" utilities.

LOCKMODULE -- The file locking utilities.

RND must be imported when you use the random number generator. RND must be on-line at compile time and at run time. If you won't be using this operation, the module need not be contained in your LIBRARY. The random number generator is described in the Library Reference section of this manual under RAND (the function) and RANDOM (the procedure).

LOCKMODULE must be imported if you use the file locking operations on LOCKABLE files. LOCKMODULE must be on-line at compile time and at run time. File locking operations are described in the Concurrent File Access section of the File System chapter in The Pascal User's Manual.

The HPM and UIO modules need never be imported. The HPM module needs to be on-line if you are using the $HEAP DISPOSE ON$ compiler directive or any of the graphics modules. The UIO modules needs to be on-line if you are using any of the UCSD "UNIT" operations. If you won't be using these operations, the modules need not be contained in your LIBRARY.

The Graphics Procedures

The graphics procedures and functions are contained in the GRAPHICS file on the LIB: disc. If you are using any of these procedures and functions in your programs, your LIBRARY must contain all the modules in the GRAPHICS file except DGL_INQ. DGL_INQ is needed if you use the INQ_WS procedure. Modules GENERAL_1, GENERAL_2 and HPIB_1 from the IO file are also needed at run time. The graphics routines reference them. If they are in the LIBRARY, the linking loader will find and load them.

The modules contained in GRAPHICS are:

DGL_TYPES
DGL_VAR
DGL_ARAS
DGL_RAS
DGL_MAIN
DGL_LIB
DGL_INQ

The I/O Procedures

The I/O procedures and functions are contained in the IO file on the LIB: disc. If you are using I/O procedures and functions in your programs, your LIBRARY must contain the modules which contain those procedures and functions. The Library Reference section of this manual lists the module(s) you must IMPORT for each procedure and function. You must then refer to the Module Dependency Table at the end of Chapter 2 to see what other modules (if any) must be present in your LIBRARY.

The modules contained in IO are:

IODECLARATIONS
GENERAL_0
IOLIBRARY_KERNEL
IOCOMASM
GENERAL_1
HPIB_1
GENERAL_2
GENERAL_3
GENERAL_4
HPIB_0
HPIB_2
HPIB_3
SERIAL_0
SERIAL_3

The INTERFACE File

The INTERFACE file contains modules comprised of interface text only. The interface text is that part of a program under the IMPORT and EXPORT statements and above the IMPLEMENT statement. The interface text is what the compiler needs when it is compiling a list of external references for your program. These modules contain the interface text for much of the operating system software. They are probably of little use to any but system designers possessing the System Internals Documentation for Pascal 2.0.

If you have a hard disc for your system volume (and you are not concerned with disc space), you should copy all of these modules into your LIBRARY. These modules should not be copied **last**. Copy either GRAPHICS, IO or the existing LIBRARY modules into your new LIBRARY last.

If you do not have a hard disc for your system volume (and you are concerned with disc space), you may need one module from the INTERFACE file. SYSGLOBALS is imported by IODECLARATIONS which in turn is imported by each of the I/O modules. SYSGLOBALS should be present in your LIBRARY if you are using any of the I/O modules.

The modules contained in INTERFACE are:

ASM
SYSGLOBALS
MINI BOOTDAMMODULE
LOADER
INITLOAD
ISR
MISC
FS
INITUNITS
LDR
SETUPSYS
KBD
INITKBD
KEYS
KEYSINIT
CRT
INITCRT
BAT
INITBAT
CLOCK
CLOCKINIT
CI
CMD

Building Your LIBRARY

If you know how to use the Librarian, the brief description below will help you get started building a new LIBRARY. If you don't know how to use the Librarian yet, a step-by-step procedure follows the brief description.

Before the Librarian is loaded, your computer configuration must be capable of supporting two mass storage devices simultaneously. The fact that the new file cannot be taken off-line during the process, necessitates the two-volume configuration. The second volume could be another flexible disc drive, a hard disc, a Shared Resource Management system or a memory volume.

Making a Memory Volume

If you don't have two disc volumes, you must create a memory volume. It is usually more convenient to use the memory volume as the destination volume (the one containing the new file). The amount of memory that must be available to make a new library using a single disc and a memory-resident volume depends on how much will be put in the new library. If you were to add both IO and GRAPHICS to the original LIBRARY you would need to allocate about 300 blocks of 512 bytes to the memory volume (153,600 bytes). Memory consumed by the memory volume cannot be recovered without re-booting the computer.

To make a memory volume:

1. At the Command prompt level, press [M] . The computer responds:

   ```
   *** CREATING A MEMORY VOLUME ***

   What unit number ?
   ```

2. You answer:

   #50 [ENTER]

   It asks:

   ```
   How many 512 byte BLOCKS ?
   ```

3. You answer:

   300 [ENTER]

   It asks:

   ```
   How many entries in directory ?
   ```

4. You answer:

   8 [ENTER]

   It finishes:

   ```
   #50:  (RAM) zeroed
   ```

   This has reserved approximately 150K bytes of memory to use as a mass storage device. It is like having a disc drive with a disc named "RAM" inserted in it.

The Brief Description

Creating a library is one of the capabilities of the Librarian. Using the Librarian, modules are copied from the Input library file to the Output library file. When all modules are copied, the Output file is Kept. The operating system must now be informed of the new library. You can rename the original LIBRARY to something else before the Librarian process and then create the new library with the name LIBRARY; you can create the new library and then replace the original using the Filer/Filecopy/Remove commands; or you can use the What/liBrary commands to specify the new library filename as the System Library. The last solution is temporary. When the conputer is re-booted, the operating system is initialized to look for "SYSVOL:LIBRARY" again.

The Step-By-Step Example

In this example, you will build a LIBRARY to support graphics programming. Using the Module Dependency Table at the end of this chapter, you find that this involves copying all of the graphics modules from the GRAPHICS file and three of the modules from the IO file (both files are on the LIB: disc). In this example, the new library is created with the name "TEMP" on a volume named "ANYVOL". In this example, you should substitute the correct name of your destination volume for ANYVOL. When finished, the new library's filename is changed to LIBRARY.

1. Have the LIBRARIAN file on-line and give the Librarian command from the Main Command Level by typing [L].
2. When the Librarian's prompt is displayed, name the the Output file. Press [O] and type:

   `ANYVOL:TEMP` [ENTER]
3. Name the first Input file. Press [I] and type:

   `SYSVOL:LIBRARY.` [ENTER]

   The period in the file name prevents .CODE from being added.
4. Use the [A] command to transfer All the modules.
5. Put the LIB: disc on-line, press [I] for Input and type:

   `LIB:GRAPHICS.`

   Notice the period.
6. Use the [A] command to transfer All the graphics modules to the new library.
7. Press [I] again to specify a new Input file and type:

   `LIB:IO.` [ENTER]

   Notice the period.
8. Press the space bar four times until the module name GENERAL_1 appears.
9. Use the Transfer command three times to transfer the next three modules. These are GENERAL_1, HPIB_1 and GENERAL_2.

When the Transfer is complete, give the Keep command by pressing [K].

Before quitting the Librarian, you can examine the new library to see that all the modules are there. Specify ANYVOL:TEMP as the Input file. Press the spacebar repeatedly. The names of the modules contained in the library will be displayed one after another.

The last step is to Filecopy the new library called TEMP to the system volume with the name LIBRARY. Give the Filer's Filecopy command and type:

```
ANYVOL:TEMP,CODE,SYSVOL:LIBRARY
```

If the original LIBRARY is still on SYSVOL, you will be informed of this and asked:

```
SYSVOL:LIBRARY
exists...Remove, Overwrite, Neither ? (R/O/N)
```

Respond with the Remove option by pressing R.

Notes and Possible Problems

If you add enough modules to the Output file, the Librarian may eventually report the "file header full" error. If this happens to you, start over and use the H command to specify a larger library Header before specifying the Output file. A header specification of 58 is usually big enough for most situations.

The Module Dependency Table

The Module Dependency Table shows which modules are needed in the LIBRARY to support the modules you must IMPORT.

Module Imported	Must Also Be in LIBRARY
RND	SYSGLOBALS
LOCKMODULE	SYSGLOBALS
DGL_LIB	DGL_TYPES, DGL_VARS, DGL_AUTL, DGL_TOOLS, DGL_GEN, DGL_RASTER, DGL_HPGL, DGL_CONFG_OUT, DGL_KNOB, DGL_HPGLI, DGL_CONFG_IN, GLE_AUTL, GLE_UTLS, GLE_TYPES, GLE_STROKE, GLE_STEXT, GLE_SMARK, GLE_SCLIP, GLE_FILE_IO, GLE_HPIB_IO, GLE_HPGL_OUT, GLE_HPGL_IN, GLE_RAS_OUT, GLE_KNOB_IN, GLE_GEN, GLE_GENI
DGL_POLY	DGL_TYPES, DGL_VARS, DGL_AUTL, DGL_TOOLS, DGL_GEN, DGL_RASTER, DGL_HPGL, DGL_CONFG_OUT, DGL_KNOB, DGL_HPGLI, DGL_CONFG_IN, DGL_LIB, GLE_AUTL, GLE_UTLS, GLE_TYPES, GLE_STROKE, GLE_STEXT, GLE_SMARK, GLE_SCLIP, GLE_FILE_IO, GLE_HPIB_IO, GLE_HPGL_OUT, GLE_HPGL_IN, GLE_RAS_OUT, GLE_KNOB_IN, GLE_GEN, GLE_GENI
DGL_INQ	DGL_TYPES, DGL_VARS, DGL_GEN, GLE_TYPES, GLE_GEN
DGL_TYPES	—
GENERAL_0	IODECLARATIONS, SYSGLOBALS
GENERAL_1	IODECLARATIONS, SYSGLOBALS
GENERAL_2	IODECLARATIONS, SYSGLOBALS, GENERAL_1, HPIB_1
GENERAL_3	IODECLARATIONS, SYSGLOBALS
GENERAL_4	IODECLARATIONS, SYSGLOBALS, HPIB_1
HPIB_0	IODECLARATIONS, SYSGLOBALS
HPIB_1	IODECLARATIONS, SYSGLOBALS
HPIB_2	IODECLARATIONS, SYSGLOBALS, HPIB_0, HPIB_1
HPIB_3	IODECLARATIONS, SYSGLOBALS, GENERAL_1, HPIB_0, HPIB_1
SERIAL_0	IODECLARATIONS, SYSGLOBALS
SERIAL_3	IODECLARATIONS, SYSGLOBALS
IOCOMASM	IODECLARATIONS, SYSGLOBALS
IODECLARATIONS	SYSGLOBALS

The Module Dependency Table

DGL_LIB needs these modules at load time.

ASM, FS, HPM, IODECLARATIONS, ISR, KBD, KEYS, MINI, MISC, SYSGLOBALS, GENERAL_1, GENERAL_2, HPIB_1.

DGL_INQ needs these modules at load time.

ASM, SYSGLOBALS

Chapter 2
The LIF Procedures

The LIF file handling capabilities were included in the Pascal filing system in Pascal 2.0. The LIF procedures have been dropped from the Procedure Library since they are now included in HP Standard Pascal on your Series 200 computer.

Sample Programs

The LIF library procedures are exercised for your examination in the following programs.

```
PROGRAM SAMPLE1(INPUT,OUTPUT,LISTING);
{
 THIS PROGRAM SHOWS HOW TO ACCESS LIF ASCII FILES
 AND
 SHOWS THE USE OF SOME OF THE NON READ/WRITE
 FUNCTIONS IN THE LIF LIBRARY.
}
IMPORT LIFLIB;

VAR
    DFILE  :LIFFILE;    {lif file info. pointer}
    BUFFER :LIFBUFFER;
    STRBUF :STRING[80];
    I,J,UNIT,ASIZE :INTEGER;
    NAME   :LIFNAME;

BEGIN
{set file name and pascal unit #}

    NAME:='TESTF';
    UNIT:=3;

{purge the file to make sure it is gone before trying to create it }

    I:=LIFPURGE(UNIT,NAME);
    WRITELN('PURGE RESULT',I);

{ create an ASCII (type 1) file to occupy 5 sectors }

    I:=LIFCREATE(UNIT,NAME,1,5);
    WRITELN('CREATE RESULT',I);

{
  open the file ASCII (type 1) file on system unit UNIT named NAME
  for writing (LIFW). LIFGETFIB indicates that the file info. record
  is to be allocated now and DFILE made to point to it
}

    I:=LIFOPEN(DFILE,LIFGETFIB,UNIT,NAME,1,LIFW);
    WRITELN('OPEN RESULT',I);

{ check the file size. this is the created size (in sectors) }

    I:=LIFGETFLD(DFILE,LIFFSIZE,J);
    WRITELN('FILE SIZE',J);

{ put data into the file from a string

  note: the size is passed in a separate parameter from the data.

  note: if the string is length 0 then a range error will result
        from the use of  STRBUF[1]
}

    STRBUF:='ASCII TEST RECORD 1';
    I:=LIFASCIIPUT(DFILE,STRLEN(STRBUF),STRBUF[1]);
    WRITELN('PUT RESULT',I);

{put data into the file from a packed array of 0..255 (TYPE LIFBUFFER)}

    FOR J:=0 TO 29 DO BUFFER[J]:=ORD('A');

    I:=LIFASCIIPUT(DFILE,30,BUFFER);
    WRITELN('PUT RESULT',I);
```

```
{close the file, save it and reduce the allocated size to the
 minimum number of sectors needed to hold the data now in the file }

    I:=LIFCLOSE(DFILE,LIFMINSIZE);
    WRITELN('CLOSE RESULT',I);

{open the file for reading (LIFR) }
    I:=LIFOPEN(DFILE,LIFGETFIB,UNIT,NAME,1,LIFR);

{ check the file size }
    I:=LIFGETFLD(DFILE,LIFFSIZE,J);
    WRITELN('FILE SIZE',J);

{ read each record into a packed array of 0..255 and print it }

    WHILE NOT LIFEOF(DFILE) DO
    BEGIN
       I:=LIFASCIIGET(DFILE,80,ASIZE,BUFFER);
    {-- note that ASIZE is the number of data bytes --}
       FOR J:=0 TO ASIZE-1 DO WRITE(CHR(BUFFER[J]));
       WRITELN;
    END;

{ close the file and keep it }
    I:=LIFCLOSE(DFILE,LIFKEEP);

    WRITELN('SECOND READING');

{open the file and read each record into a string }
    I:=LIFOPEN(DFILE,LIFGETFIB,UNIT,NAME,1,LIFR);

    WHILE NOT LIFEOF(DFILE) DO
    BEGIN

{ check the record size before reading the data}
        I:=LIFGETFLD(DFILE,LIFRSIZE,ASIZE);
        IF ASIZE>20 THEN WRITELN('BIG RECORD',ASIZE);

{force string size to avoid a range check error on call to LIFASCIIGET }
        SETSTRLEN(STRBUF,80);

{ 80 is the max number of bytes to read }
{ ASIZE will contain the actual number read}
        I:=LIFASCIIGET(DFILE,80,ASIZE,STRBUF[1]);

{set the string length before using the string }
        SETSTRLEN(STRBUF,ASIZE);

        WRITELN(STRBUF);
    END;

{ close the file and remove(purge) it }
    I:=LIFCLOSE(DFILE,LIFREMOVE);
END.
```

```
PROGRAM SAMPLE2(INPUT,OUTPUT);
{program to show how to access LIF non ASCII files }

IMPORT LIFLIB;
TYPE
   SECTYP = ARRAY[1..256] OF CHAR;

VAR
   FREC   :LIFFILE;     {lif file info pointer }
   NAME   :LIFNAME;
   SECTOR :SECTYP;
   C1     :CHAR;
   ASIZE,DVALUE,SBYTE,S,I,K,L,VOL: INTEGER;

CONST
   NUMSECTS = 4;

BEGIN

{ set PASCAL unit # and lif file name }
    VOL := 3;
    NAME := 'RDATA';

{ remove the file before trying to create it}
    S := LIFPURGE(VOL,NAME);

{create a type 9 file to occupy 4 sectors
 note: there is no formal definiton for a type 9 file,
       it is used here only as an example }

    S := LIFCREATE(VOL,NAME,9,4);

{open the file for writing}
    S := LIFOPEN(FREC,LIFGETFIB,VOL,NAME,9,LIFW);

 { if no error on open then proceed }
    IF S = 0 THEN
    BEGIN

    {set character to fill record}
        C1 := 'A';

    {write all records}
        FOR I := 1 TO NUMSECTS DO
        BEGIN

        {fill the output record}
            FOR L := 1 TO 256 DO SECTOR[L] := C1;

        {calculate start byte for each record}
            SBYTE := (NUMSECTS-I)*256;

            writeln('sector',numsects-i+1:2,sbyte:8,'  ',c1);

        {get character to fill next record}
            C1 := SUCC(C1);

        {output the record}
            S := LIFPUT(FREC,SBYTE,256,SECTOR);
```

```
            { if put ok then continue }
                IF S<>0 THEN
                BEGIN
                    writeln('status is',s:2,' sector is',i:2);

                    {check actual bytes writen}
                        S := LIFGETFLD(FREC,LIFRSIZE,DVALUE);
                        writeln('lifrsize is',dvalue:4);
                END;
            END;

        {close the file and keep it}
            S := LIFCLOSE(FREC,LIFKEEP);

        END;
        writeln('-----------------------');

    { now read the file and see what got put out there }
        S := LIFOPEN(FREC,LIFGETFIB,VOL,NAME,9,LIFR);

    { if open worked then continue }
        IF S=0 THEN
        BEGIN

{ set start byte }
        SBYTE := 0;

    { read all records }
        FOR K := 1 TO NUMSECTS DO
        BEGIN
            writeln('sector',K:2,sbyte:8,' ');

        { read and use SBYTE as auto incremented by LIFGET }
            S := LIFGET(FREC,SBYTE,256,ASIZE,SECTOR);
            IF S<>0 THEN writeln('get failed',s:3);

        {print part of the record}
            FOR I := 1 TO 20 DO WRITE(SECTOR[I]);
            WRITELN;

        END;

   { close and remove(purge) the file}
        S := LIFCLOSE(FREC,LIFREMOVE);

    END;
END.
```

Chapter 3
The Graphics Procedures

Introduction

The Device-Independent Graphics Library (DGL) is a collection of predefined procedures that allow Pascal to communicate with most HP graphic peripherals. These procedures are listed in the Procedure Library Summary at the front of the Library Reference section and defined within the Library Reference.

DGL is a low level two-dimensional graphics system designed to provide the elementary input and output functions necessary for controlling graphic devices.

The capabilities provided by DGL can be divided into five major areas: graphics output primitives, primitives attributes, viewing transformation, input, and control.

Graphics output primitives are the building blocks of a graphics picture. The graphics system uses three types of graphics output primitives: lines, text, and moves. Primitive attributes affect the appearance of individual output primitives. For example, a line primitive's attributes are color and linestyle. Character size is a primitive attribute which only applies to text.

A viewing transformation is the method whereby an object, a collection of graphics primitives defined in an abstract coordinate system, is converted to an image displayed on a physical display device.

An input capability provides a means of entering information from a graphic device; thus allowing graphic programs to be truly interactive.

Control functions are used to manage various aspects of a graphics application such as initialization and modification of the graphics environment.

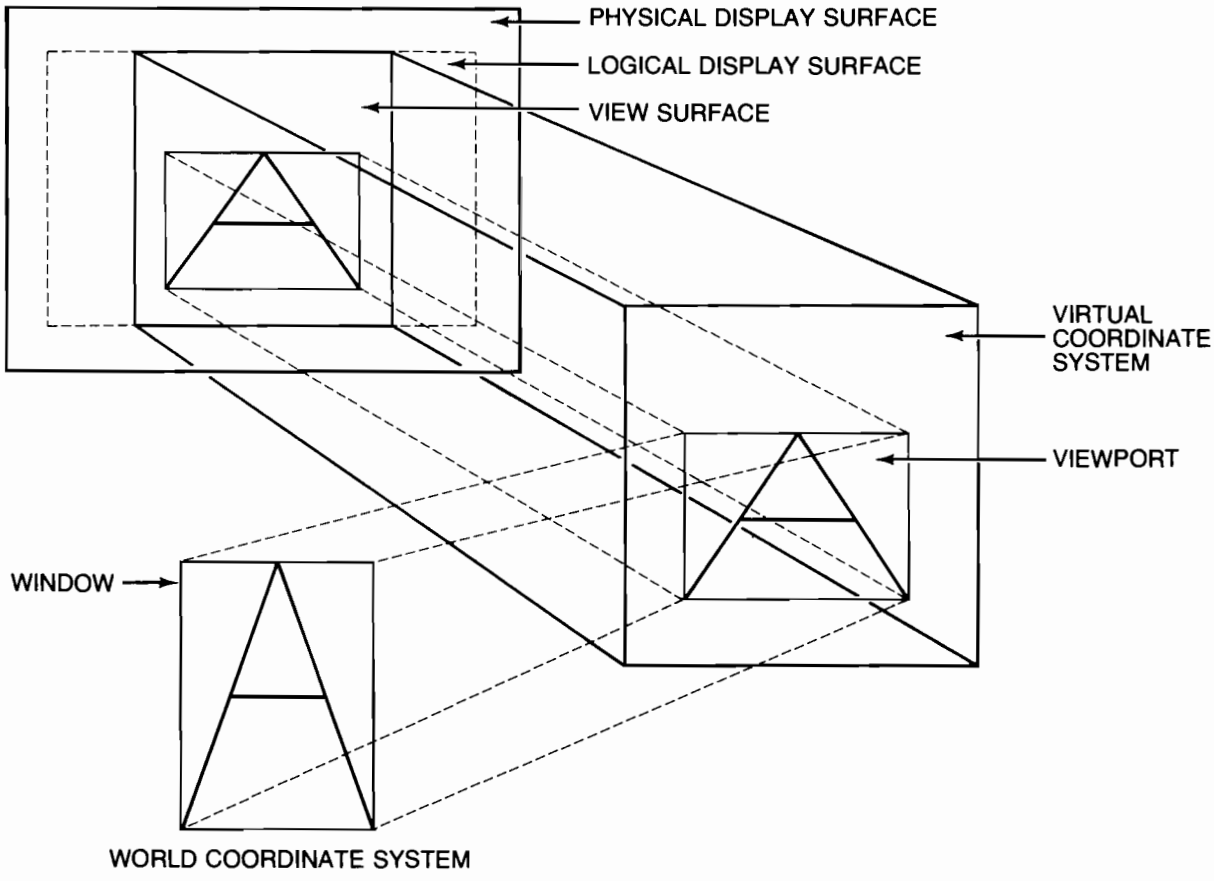
The viewing transformation is explained first in this chapter, followed by device handler information. Helpful programming examples with explanations included in the source code are provided following the device handler information. The chapter is concluded with a list of deviations from DGL 1000 and an error code summary.

The Viewing Transformation

Of these five functional capabilities, the viewing transformation is the most critical to grasp. The viewing transformation is a mapping from one two-dimensional coordinate system to another.

One of the coordinate systems is the world coordinate system. This is an abstract coordinate system in which the user specifies the location of all graphics primitives. The second is the virtual coordinate system. This coordinate system describes the view surface. The view surface is the portion of the physical display device which will be used to display the graphics image.

The viewing transformation maps from the world coordinate system to the virtual coordinate system. The virtual coordinate system is then mapped onto the display device which produces the picture that the user sees.



Device Handlers

Pascal supports the following devices for input (locators) and/or output (display) operations.

- HP Models 16, 26 and 36 Computers
- HP 9872A/B/C/S/T Plotters
- HP 7580A and 7585A Plotters
- HP 7470A Plotter
- HP 98627A Color Interface (display only)
- HP 9111A Graphics Tablet (locator only)
- General HPGL Devices

Unrecognized HPGL devices are handled on the assumption that they have the same capabilities as the HP 9872B plotter. See the end of this chapter for details concerning the general display handler.

HP Models 16, 26 and 36 Displays

Description

The dimensions of the graphic displays are as follows:

Model 16
Screen size: 168 mm wide by 126 mm high
Screen capacity: 400 points wide by 300 points high
Aspect ratio of maximum area: 0.75

Model 26
Screen size: 120 mm wide by 90 mm high
Screen capacity: 400 points wide by 300 points high
Aspect ratio of maximum area: 0.75

Model 36
Screen size: 210 mm wide by 160 mm high
Screen capacity: 512 points wide by 390 points high
Aspect ratio of maximum area: 0.7617

The default logical display surface of the graphics display device is the maximum physical limits of the screen. The physical origin is the lower left corner of the display.

The view surface is always centered within the current logical display surface.

Initialization

When the graphics display is initialized (DISPLAY_INIT) the following device dependent actions are performed:

- The starting position is in the lower left corner of the display.
- Graphics memory is cleared.
- The graphics display is turned on.
- The view surface is centered within the logical display limits.
- The drawing mode (see OUTPUT_ESC) is set to dominate.
- The DISPLAY_INIT CONTROL parameter is ignored for the graphics display.

Primitive Attributes

SET_COLOR

The supported values of color are:

COLOR = 0 — Color set to background color (erase)
= 1 — Color set to white.

The color attribute interacts with the set special drawing mode output escape function (1052) as follows:

DOMINATE (INTEGERarray[1] = 0) (Default mode)

COLOR = 0 — Background (erase, set bits to 0)
= 1 — White (set bits)

NON-DOMINATE (INTEGERarray[1] = 1)

COLOR = 0 — Background (erase, set bits to 0)
= 1 — White (set bits to 1)

ERASE (INTEGERarray[1] = 2)

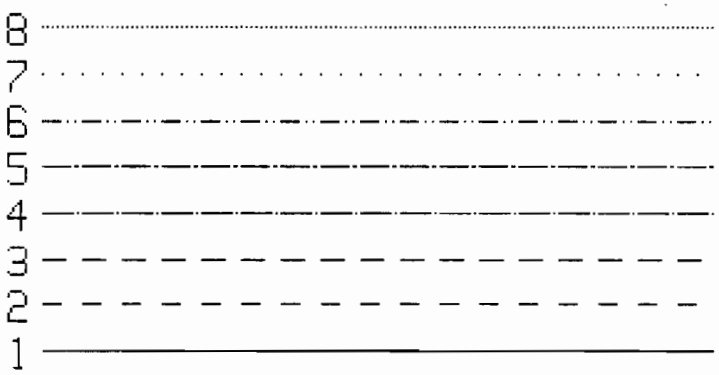
COLOR = 0 — Background (erase, set bits to 0)
= 1 — Erase white (set bits to 0)

COMPLEMENT (INTEGERarray[1] = 3)

COLOR = 0 — Background (erase, set bits to 0)
= 1 — Complement (invert bits)

SET_LINE_STYLE

Eight pre-defined linestyles are supported on the graphics display. All of the linestyles may be classified as being "continuous".



CLEAR_DISPLAY

A call to CLEAR_DISPLAY erases all of the graphics display.

Inquiry Escape Functions

None.

Output Escape Functions

OPCODE	FUNCTION
52	Dump graphics to the graphics printer (PRINTER:).
1050	Turn on or off the graphics display. INTEGERarray [1] = 0 — turn display off. INTEGERarray [1] <> 0 — turn display on.
1051	Turn on or off the alpha display. INTEGERarray [1] = 0 — turn display off. INTEGERarray [1] <> 0 — turn display on.
1052	Set special drawing modes. Using this escape function will redefine the meaning of the set color attribute. For details on how a given drawing mode affects a color set SET–COLOR above. Out of range values default to dominate drawing mode. INTEGERarray[1] = 0 — Dominate drawing mode. = 1 — Non-dominate drawing mode. = 2 — Erase drawing mode. = 3 — Complement drawing mode.

Locator Echoes on the Graphics Display

All locator echoes are supported by the graphics display. The starting position is unaffected by echoes on the graphics display.

HP Models 16, 26 and 36 Locators

The default locator limits are set equal to the maximum physical limits of the screen.

The physical origin of the locator device is the lower left corner of the display.

Initialization

LOCATOR_INIT

When the locator device is initialized, the graphics display is left unaltered.

Await Locator Input

AWAIT_LOCATOR

When using the HP Model 16, 26 or 36, as await locator the keyboard keys have the following meanings:

Arrow keys — Move the cursor in the direction indicated.
Knob — Move the cursor right and left.
Knob with SHIFTα — Move the cursor up and down.
Number keys (1 to 9) — Change the amount the cursor is moved per arrow key press or knob rotation. [1] provides the least movement and [9] provides the most.

All other keys act as the locator buttons. The ordinal value of the locator button (key) struck is returned in BUTTON.

Invoking AWAIT_LOCATOR with ECHO = 0 or ECHO = 1 turns on the graphics display.

Echo Supported

Locator input can be echoed on either a graphics display device or a locator device. For the echoes supported on a graphics display device, see the chapter which describes the graphics display in question.

The supported echoes on the locator device are as follows:

ECHO#	Echo Performed
0	No echo performed.
1	The position of the locator is indicated by a small crosshair cursor on the internal graphics display. The initial position of the cursor is located at the current starting position of the internal graphics display. For back to back AWAIT_LOCATOR calls this would mean the second AWAIT_LOCATOR would begin were the first AWAIT_LOCATOR left the cursor.
9 or greater	Same as ECHO #1

Sample Locator Input

The SAMPLE–LOCATOR function returns the last AWAIT–LOCATOR result of 0.0 if AWAIT–LOCATOR has not been invoked since LOCATOR–INIT.

Echoes Supported:

ECHO#	Echo Performed
0	No echo
1	The desktop beeper is sounded when the locator is sampled.

HP 9872A/B/C/S/T Plotters

Description

The dimensions of the HP 9872 plotters are as follows:

Platen surface:	420mm wide by 297mm high
Plotting area:	400mm wide by 285mm high
Plotting capacity:	16000 points wide by 11400 points high
Aspect ratio of maximum area:	0.7125
Resolution:	40.0 points/mm in X and Y directions

The default logical display surface is set equal to the area defined by P1 and P2 at the time DISPLAY_INIT is invoked.

The physical origin of the graphics display is 12mm to the right of the left edge of the platen and 6mm above the lower edge of the platen. This is the lower left boundary for pen movement.

The viewsurface is always justified in the lower left corner of the current logical display surface.

Initialization

DISPLAY_INIT

When the HP 9872 plotters are initialized the following device dependent actions are performed:

- The starting position is undefined.
- Pen velocity is set to 36 cm/sec.
- Paper cutter is enabled (HP 9872S/T).
- Advance page option is enabled (HP 9872S/T).
- Paper is advanced one full page (HP 9872S/T).
- The DISPLAY_INIT CONTROL parameter is ignored for the HP 9872 displays.

Primitive Attributes

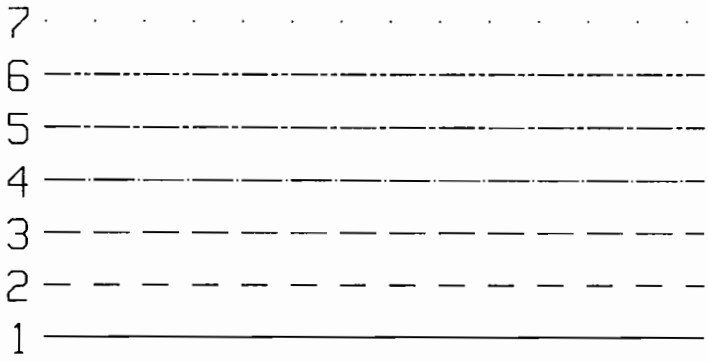
SET_COLOR

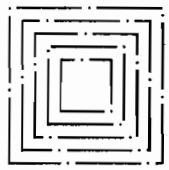
The supported values of color are:

0 Current pen is put away.
1 Pen 1 is selected.
2 Pen 2 is selected.
3 Pen 3 is selected.
4 Pen 4 is selected.
5 Pen 5 is selected (HP 9872C/T).
6 Pen 6 is selected (HP 9872C/T).
7 Pen 7 is selected (HP 9872C/T).
8 Pen 8 is selected (HP 9872C/T).

SET_LINE_STYLE

Seven predefined linestyles are supported on the HP 9872 plotters. All linestyles supported on the HP 9872 plotters may be classified as being "continuous".





CONTINUOUS

CLEAR_DISPLAY

A call to CLEAR_DISPLAY is ignored for the HP 9872A/B/C plotters and advances the paper on the HP 9872S/T plotters.

Inquiry Escape Functions

No inquiry escape functions are supported.

Output Escape Functions

Opcode	Function
1052	Enable cutter. Provides means to control the HP 9872 S/T paper cutters. Paper is cut after it is advanced. IntegerArray[1] = 0 Cutter is disabled IntegerArray[1] <> 0 Cutter is enabled
1053	Advance the paper either one half or a full page. (HP 9872S or HP 9872T only). IntegerArray[1] = 0 Advance page half IntegerArray[1] <> 0 Advance page full
2050	Select pen speed. This instruction allows the user to modify the plotter's pen speed. Pen speed may be set from 1 to 36 cm / sec. IntegerArray[1] = Pen speed (integer from 1 to 36). IntegerArray[2] = Pen number (integer 1 to 4 for HP 9872A/B/S). (integer 1 to 8 for HP 9872C/T). Pen numbers outside of these ranges will change speed for all pens.

Locator Echoes on the Graphics Display

AWAIT_LOCATOR

The type of echoes available on the graphics display depend on whether or not the graphics display and locator are the same physical device. For echoes supported on the locator device, see the section which discusses the locator device in question.

Same Physical Device

If the locator and display are the same device (e.g., HP 9872 display and HP 9872 locator at the same device address) then the following echoes are supported on the graphics display:

Echo Number	Echo Performed
2	Small cursor
	Initially the 9872's pen will be moved to the current locator echo position. The pen will continue to reflect the current locator position (i.e., tracked) until the locator operation is terminated (i.e. ENTER pressed).
3	Full cross hair cursor
	Simulated by ECHO #2.
4	Rubber band line
	Simulated by ECHO #2.
5	Horizontal rubber band line
	Simulated by ECHO #2 except the current locator X coordinate and the locator echo position Y coordinate are returned.
6	Vertical rubber band line
	Simulated by ECHO #2 except the locator echo position X coordinate and the current locator Y coordinate are returned.
7	Snap horizontal / vertical rubber band line
	If the locator's X displacement from the locator echo position is greater than or equal to its Y displacement, ECHO #5 is simulated. Otherwise ECHO #6 is simulated.
8	Rubber band box
	Simulated by ECHO #2.

Different Physical Devices

If the locator and graphics display are physically different (e.g., HP 9872 display and 9111A locator), then the following echoes are supported on the display:

Echo Number	Echo Performed
2	Small cursor Initially the 9872's pen will be moved to the current locator echo position. The pen will continue to reflect the current locator position (i.e., tracked) until the locator operation is terminated.
3	Full cross hair cursor Simulated by ECHO #2.
4	Rubber band line Simulated by ECHO #2.
5	Horizontal rubber band line Initially the plotter's pen will be moved to the current locator echo position. The pen will then continue to reflect the X coordinate of the current locator position and the Y coordinate of the current locator echo positon.
6	Vertical rubber band line Initially the plotter's pen position will be moved to the current locator echo position. The pen will then continue to reflect the X coordinate of the current locator echo position and the Y coordinate of the current locator position.
7	Snap horizontal / vertical rubber band line If the locators X displacement from the locator echo position is greater than or equal to its Y displacement, ECHO #5 is simulated. Otherwise ECHO #6 is simulated.
8	Rubber band box Simulated by ECHO #2.

HP 9872A/B/C/S/T Locator

The default logical locator limits are set equal to the area defined by P1 and P2.

The physical origin of the locator device is 12 mm to the right of the left edge of the platen and 6 mm above the lower edge of the platen. This is the lower left corner of pen movement.

No locator points are returned while the pen control buttons are depressed.

Initialization

LOCATOR_INIT

When the locator device is initialized, the plotter's graphics display is left unaltered.

Wait Locator Input

AWAIT_LOCATOR

The wait locator function enables a digitizing mode in the HP 9872 plotter which causes the enter light to be turned on. The operator then positions the pen to the desired position with the cursor buttons and then strikes the enter key. A one is returned as the button value if the pen is down and a zero is returned if the pen is up.

Echo Supported

Locator input can be echoed on either a graphics display device or a locator device. For the echoes supported on a graphics display device, see the chapter which describes the graphics display in question.

When following locator input on the locator device with ECHO = 0 or ECHO = 1, the pen position will remain at the last position it was moved to by the operator. This means that the starting position for the next graphics primitive will be wherever the pen was left.

The supported echoes on the locator device are as follows:

Echo Number	Echo Performed
0	Same as ECHO #1
1	The HP 9872's pen tracks the locator position.
9 or greater	Same as ECHO #1

Sample Locator Input

SAMPLE_LOCATOR

The sample locator function returns the current plotter pen position without waiting for an operator response.

Echoes Supported

No locator echoes are supported with the HP 9872 graphics plotters when using the sample locator functions.

HP 7470A Plotter

Description

The dimensions of the HP 7470 plotter are as follows:

Plotting area:	257.5 mm wide by 190 mm high
Plotting capacity:	10300 points wide by 7600 points high
Aspect ratio of maximum area:	0.737864
Resolution:	40 points / mm in X and Y directions

The default logical display surface is set equal to the area defined by P1 and P2.

The viewsurface is always justified in the lower left corner of the current logical display surface (edge closest to power plug).

Initialization

When the HP 7470 plotter is initialized (DISPLAY_INIT) the following device dependent actions are performed:

- The starting position is undefined.
- Pen velocity is set to 38 cm/sec.
- The DISPLAY_INIT CONTROL parameter is ignored for the HP 7470 display.

Primitive Attributes

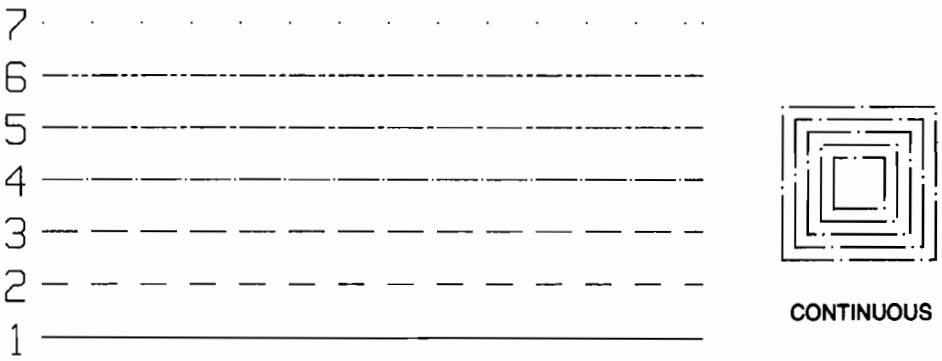
SET_COLOR

The supported values of color are:

1 Pen 1 is selected.
2 Pen 2 is selected.

SET_LINE_STYLE

Seven pre-defined linestyles are supported on the HP 7470 plotter. All linestyles supported on the HP 7470 plotter may be classified as being "continuous".



CLEAR_DISPLAY

A call to CLEAR_DISPLAY is ignored for the HP 7470 plotter.

Inquiry Escape Functions

No inquiry escape functions are supported.

Output Escape Functions

Opcode	Function
2050	Select pen speed. This instruction allows the user to modify the plotter's pen speed. Pen speed may be set from 1 to 38 cm / sec. IntegerArray[1] = Pen speed (integer from 1 to 38). IntegerArray[2] = Pen number (integer 1 or 2; others select all pens).

Locator Echoes on the Graphics Display

AWAIT_LOCATOR

The type of echoes available on the graphics display depends on whether or not the graphics display and locator are the same physical device. For echoes supported on the locator device, see the section which discusses the locator device in question.

Same Physical Device

If the locator and display are the same device (e.g., HP 7470 display and HP 7470 locator at the same device address) then the following echoes are supported on the graphics display:

Echo Number	Echo Performed
2	Small cursor Initially the 7470's pen will be moved to the current locator echo position. The pen will continue to reflect the current locator position (i.e., tracked) until the locator operation is terminated (i.e., ENTER pressed).
3	Full cross hair cursor Simulated by ECHO #2.
4	Rubber band line Simulated by ECHO #2.
5	Horizontal rubber band line Simulated by ECHO #2 except the current locator X coordinate and the locator echo position Y coordinate are returned.
6	Vertical rubber band line Simulated by ECHO #2 except the locator echo position X coordinate and the current locator Y coordinate are returned.
7	Snap horizontal / vertical rubber band line If the locators X displacement from the locator echo position is greater than or equal to its Y displacement, ECHO #5 is simulated. Otherwise ECHO #6 is simulated.
8	Rubber band box Simulated by ECHO #2.

Different Physical Devices

If the locator and graphics display are physically different (e.g., HP 7470 display and 9111A locator), then the following echoes are supported on the display:

Echo Number	Echo Performed
2	Small cursor Initially the 7470's pen will be moved to the current locator echo position. The pen will continue to reflect the current locator position (i.e., tracked) until the locator operation is terminated.
3	Full cross hair cursor Simulated by ECHO #2.
4	Rubber band line Simulated by ECHO #2.
5	Horizontal rubber band line Initially the plotter's pen will be moved to the current locator echo position. The pen will then continue to reflect the X coordinate of the current locator position and the Y coordinate of the current locator echo positon.
6	Vertical rubber band line Initially the plotter's pen position will be moved to the current locator echo position. The pen will then continue to reflect the X coordinate of the current locator echo position and the Y coordinate of the current locator position.
7	Snap horizontal / vertical rubber band line If the locators X displacement from the locator echo position is greater than or equal to its Y displacement, ECHO #5 is simulated. Otherwise ECHO #6 is simulated.
8	Rubber band box Simulated by ECHO #2.

HP 7470 Locator

The default logical locator limits are set equal to the area defined by P1 and P2.
No locator points are returned while the pen control buttons are depressed.

Initialization

LOCATOR_INIT

When the locator device is initialized, the plotter's graphics display is left unaltered.

Wait Locator Input

AWAIT_LOCATOR

The wait locator function enables a digitizing mode in the HP 7470 plotter which causes the enter light to be turned on. The operator then positions the pen to the desired position with the cursor buttons and then strikes the enter key. Button returns a one if the pen is down and a zero if the pen is up.

Echo Supported

Locator input can be echoed on either a graphics display device or a locator device. For the echoes supported on a graphics display device, see the chapter which describes the graphics display in question.

Following locator input, the pen position will remain at the last position it was moved to by the operator. This means that the starting position for the next graphics primitive will be wherever the pen was left.

The supported echoes on the locator device are as follows:

Echo Number	Echo Performed
0	Same as ECHO #1
1	The HP 7470's pen tracks the locator position.
9 and greater	Same as ECHO #1

Sample Locator Input

SAMPLE_LOCATOR

The sample locator function returns the current plotter pen position without waiting for an operator response.

Echoes Supported

No locator echoes are supported with the HP 7470 graphics plotters when using the sample locator functions.

HP 7580 and 7585 Plotters

Description

The dimensions of the HP 7580 plotter are as follows:

Plotting area: 809.5mm wide by 524.25mm high
Plotting capacity: 32 380 points wide by 20 970 points high
Aspect ratio of maximum area: 0.6476
Resolution: 40.0 points/mm in X and Y directions

The dimensions of the HP-7585 plotter are as follows:

Plotting area: 1100mm wide by 890mm high
Plotting capacity: 44 000 points wide by 35 670 points high
Aspect ratio of maximum area: 0.8090
Resolution: 40.0 points/mm in X and Y directions

The default logical display surface is set equal to the area defined by P1 and P2 at the time DISPLAY_INIT is invoked. The maximum logical display surface, that is the largest size that the logical display surface may be set with the SET_DISPLAY_LIM procedure, is determined by the size of paper loaded in the plotter at the time the DISPLAY_INIT procedure is invoked.

If the paper is changed while the graphics display is initialized, it should be the same size of paper that was in the plotter when DISPLAY_INIT was called. If a different size of paper is required, the device should be terminated (DISPLAY_TERM) and re-initialized after the new paper has been placed in the plotter.

The view surface is always justified in the lower left corner (corner nearest the turret) of the current logical display surface.

The physical origin is at the lower left boundary of pen movement.

Initialization

When the plotter is initialized (DISPLAY_INIT) the following device dependent actions are performed:

- The starting position is undefined.
- Pen velocity, force, and acceleration are set to the default values for the turret loaded.
- ASCII character set is set to 'ANSI ASCII'.
- The automatic pen options are set.
- The DISPLAY_INIT CONTROL parameter is ignored for the graphics display.

Primitive Attributes

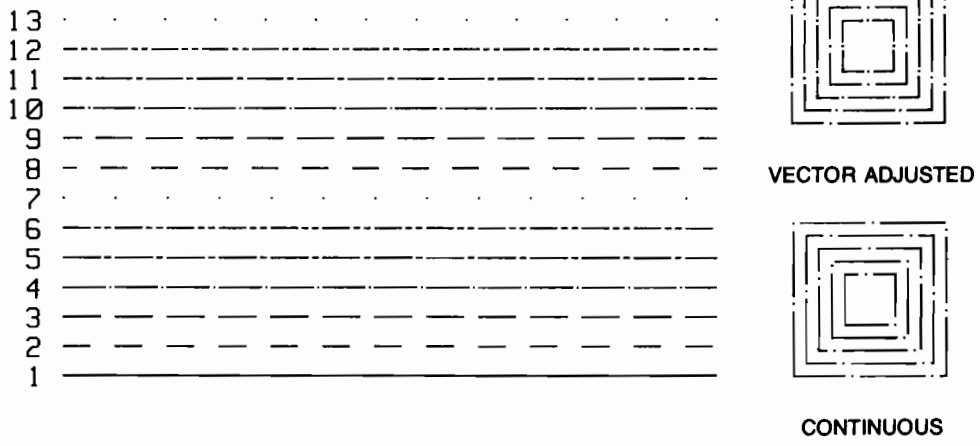
SET_COLOR

The supported values of color are:

COLOR = 0 — Current pen is put away.
= 1 — Pen 1 is selected.
= 2 — Pen 2 is selected.
= 3 — Pen 3 is selected.
= 4 — Pen 4 is selected.
= 5 — Pen 5 is selected.
= 6 — Pen 6 is selected.
= 7 — Pen 7 is selected.
= 8 — Pen 8 is selected.

SET_LINE_STYLE

Thirteen pre-defined linestyles are supported on the plotters. Linestyles one through seven may be classified as being "continuous". Linestyles eight through thirteen are the same patterns as styles two through eight drawn in the "vector adjusted" format.



CLEAR_DISPLAY

A call to CLEAR_DISPLAY is ignored for the plotter.

Inquiry Escape Functions

OPCODE	FUNCTION
2050	Inquire about current turret. INTEGERarray [1] = −1 — Turret mounted, type UNKNOWN INTEGERarray [1] = 0 — No turret mounted INTEGERarray [1] = 1 — Fiber tip pens INTEGERarray [1] = 2 — Roller ball pens INTEGERarray [1] = 3 — Capillary pens INTEGERarray [2] = 0 — No turret mounted or turret has no pens INTEGERarray [2] = n — Sum of these values: 1: Pen in stall #1 2: Pen in stall #2 4: Pen in stall #3 8: Pen in stall #4 16: Pen in stall #5 32: Pen in stall #6 64: Pen in stall #7 128: Pen in stall #8

For example, if INTEGERarray[2] = 3, pens would only be contained in stalls 1 and 2.

Output Escape Functions

OPCODE	FUNCTION
1052	Set automatic pen. This instruction provides a means for utilizing the smart pen options of the plotter. Initially, all automatic pen options are enabled. INTEGERarray [1] = n — Sum of these values: 1 : Lift pen if it has been down for 60 seconds. 2 : Put pen away if it has been motionless for 20 seconds. 4 : Do not select a pen until a command which makes a mark. This causes the pen to remain in the turret for the longest possible time.
2050	Select pen velocity. This instruction allows the user to modify the plotter's pen speed. Pen speed may be set from 1 to 60 cm/sec. INTEGERarray [1] = Pen speed (integer from 1 to 60). INTEGERarray [2] = Pen number (integer from 1 to 8; other integers select all pens)
2051	Select pen force. The force may be set from 10 to 66 gram-weights. INTEGERarray [1] = Pen force (integer from 1 to 8). 1: 10 gram-weights 2: 18 gram-weights 3: 26 gram-weights 4: 34 gram-weights 5: 42 gram-weights 6: 50 gram-weights 7: 58 gram-weights 8: 66 gram-weights INTEGERarray [2] = Pen number (integer 1 to 8; other integers select all pens)
2052	Select pen acceleration. The acceleration may be set from 1 to 4 G's. INTEGERarray [1] = Pen acceleration (integer from 1 to 4). INTEGERarray [2] = Pen number (integer 1 to 8; other integers select all pens)

Locator Echoes on the Graphics Display

AWAIT_LOCATOR

The type of echoes available on the graphics display depend on whether or not the graphics display and locator are the same physical device. For echoes supported on the locator device, see the section which discusses the locator device in question.

If the locator and display are the same device (e.g. HP 7580 display and HP 7580 locator at the same device address) then the following echoes are supported on the graphics display:

ECHO#	ECHO PERFORMED
2	Small cursor Initially the plotter's pen will be moved to the current locator echo position. The pen will continue to reflect the current locator position (i.e., tracked) until the locator operation is terminated (i.e., ENTER pressed).
3	Full cross hair cursor Simulated by ECHO #2.
4	Rubber band line Simulated by ECHO #2.
5	Horizontal rubber band line Simulated by ECHO #2 except the current locator X coordinate and the locator echo position Y coordinate are returned.
6	Vertical rubber band line Simulated by ECHO #2 except the locator echo position X coordinate and the current locator Y coordinate are returned.
7	Snap horizontal / vertical rubber band line If the locators X displacement from the locator echo position is greater than or equal to its Y displacement, ECHO #5 is simulated. Otherwise ECHO #6 is simulated.
8	Rubber band box Simulated by ECHO #2.

If the locator and graphics display are physically different (e.g. HP 7580 display and 9111A locator), then the following echoes are supported on the display:

ECHO#	ECHO PERFORMED
2	Small cursor Initially the plotter's pen will be moved to the current locator echo position. The pen will continue to reflect the current locator position (i.e. tracked) until the locator operation is terminated.
3	Full cross hair cursor Simulated by ECHO #2.
4	Rubber band line Simulated by ECHO #2.
5	Horizontal rubber band line Initially the plotter's pen will be moved to the current locator echo position. The pen will then continue to reflect the X coordinate of the current locator position and the Y coordinate of the current locator echo positon.
6	Vertical rubber band line Initially the plotter's pen position will be moved to the current locator echo position. The pen will then continue to reflect the X coordinate of the current locator echo position and the Y coordinate of the current locator position.
7	Snap horizontal / vertical rubber band line If the locators X displacement from the locator echo position is greater than or equal to its Y displacement, ECHO #5 is simulated. Otherwise ECHO #6 is simulated.
8	Rubber band box Simulated by ECHO #2.

HP 7580 and 7585 Locators

The default logical display surface is set equal to the area defined by P1 and P2 at the time LOCATOR_INIT is invoked. The maximum logical display surface, that is the largest size that the logical locator surface may be set with the SET_LOCATOR_LIM procedure, is determined by the size of paper loaded in the plotter at the time the LOCATOR_INIT procedure is invoked.

If the paper is changed while the locator is initialized, it should be the same size of paper that was in the plotter when LOCATOR_INIT was called. If a different size of paper is required, the device should be terminated (LOCATOR_TERM) and re-initialized after the new paper has been placed in the plotter.

The physical origin of the locator device is at the lower left corner of the pen movement.

Initialization

LOCATOR_INIT

When the locator device is initialized, the plotter's graphics display is left unaltered.

Await Locator Input

AWAIT_LOCATOR

The AWAIT_LOCATOR function enables a digitizing mode in the HP 7580A/7585A plotter which causes the enter light to be turned on. The operator then positions the pen to the desired position with the joystick and strikes the ENTER key. The pen state, 0 for 'up', and 1 for 'down' is returned in the button parameter.

Locator input can be echoed on either a graphics display device or a locator device. For the echoes supported on a graphics display device, see the chapter which describes the graphics display in question.

Following locator input (echo on locator), the pen position will remain at the last position it was moved to by the operator. This means that the starting position for the next graphics primitive will be wherever the pen was left.

The supported echoes on the locator device are as follows:

ECHO#	ECHO PERFORMED
0	Same as ECHO #1
1	The HP 7580/7585's pen tracks the locator position.
9 and above	Same as ECHO #1

Sample Locator Input

SAMPLE_LOCATOR

The sample locator function returns the current plotter pen position without waiting for an operator response.

No locator echoes are supported with the HP 7580/7585 graphics plotters when using the sample locator functions.

HP 98627 Display

Description

HP 98627 is a low cost I/O card for the HP 9826/9836 that adds a color CRT display via an external monitor. The external monitor is user supplied and may be of any size. The user indicates in the CONTROL variable of the DISPLAY_INIT procedure information regarding the type of the display.

CONTROL =	256	US STD	(512 x 390, 60 Hz refresh)
	512	EURO STD	(512 x 390, 50 Hz refresh)
	768	US TV	(512 x 474, 15.75 Khz horizontal refresh, interlaced)
	1024	EURO TV	(512 x 512, 50 Hz vertical refresh, interlaced)
	1280	HI RES	(512 x 512, 60 Hz)
	1536	Internal	(HP use only)

The physical size of the display (needed by the SET_DISPLAY_LIM procedure) may be given to the graphics system by an escape function. The physical limits have an assumed value based on CONTROL until the escape function is given. These limits are:

CONTROL =	256	153.3mm wide and 116.7mm high.
	512	153.3mm wide and 116.7mm high.
	768	153.3mm wide and 142.2mm high.
	1024	153.3mm wide and 153.3mm high.
	1280	153.3mm wide and 153.3mm high.

The default logical display surface of the graphics display device is the maximum physical limits of the screen. The physical origin is the lower left corner of the display.

The view surface is always centered within the current logical display surface.

Initialization

DISPLAY_INIT

When the HP 98627 graphics display is initialized the following device dependent actions are performed:

- The starting position is in the lower left corner of the display.
- Graphics memory is cleared.
- The graphics display is turned on.
- The view surface is centered within the logical display limits.
- The DISPLAY_INIT CONTROL parameter is used as specified above.

Primitive Attributes

SET_COLOR

The supported values of color are:

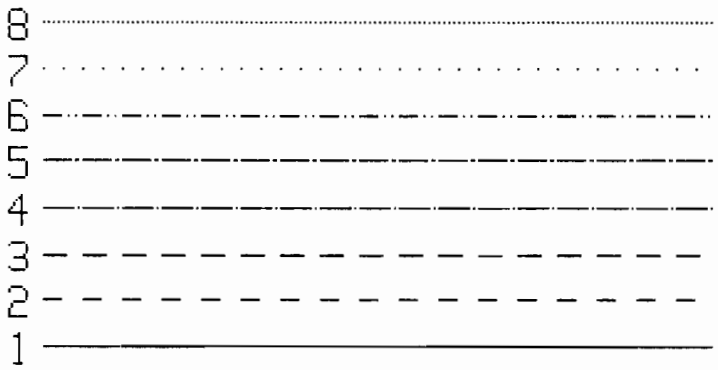
0	Background	2	Red	4	Green	6	Blue
1	White	3	Yellow	5	Cyan	7	Magenta

The color attribute interacts with the set special drawing mode escape function (1052) as follows:

COLOR #	ACTION	RED PLANE	GREEN PLANE	BLUE PLANE
DOMINATE (IntegerArray[1] = 0) (Default mode)				
0	Background	0	0	0
1	Dominate White	1	1	1
2	Dominate Red	1	0	0
3	Dominate Yellow	1	1	0
4	Dominate Green	0	1	0
5	Dominate Cyan	0	1	1
6	Dominate Blue	0	0	1
7	Dominate Magenta	1	0	1
NON-DOMINATE (IntegerArray[1] = 1)				
0	Background	0	0	0
1	Non-dominate White	1	1	1
2	Non-dominate Red	1	no change	no change
3	Non-dominate Yellow	1	1	no change
4	Non-dominate Green	no change	1	no change
5	Non-dominate Cyan	no change	1	1
6	Non-dominate Blue	no change	no change	1
7	Non-dominate Magenta	1	no change	1
ERASE (IntegerArray[1] = 2)				
0	Background	0	0	0
1	Erase White	0	0	0
2	Erase Red	0	no change	no change
3	Erase Yellow	0	0	no change
4	Erase Green	no change	0	no change
5	Erase Cyan	no change	0	0
6	Erase Blue	no change	no change	0
7	Erase Magenta	0	no change	0
COMPLEMENT (IntegerArray[1] = 3)				
0	Background	0	0	0
1	Complement White	invert	invert	invert
2	Complement Red	invert	no change	no change
3	Complement Yellow	invert	invert	no change
4	Complement Green	no change	invert	no change
5	Complement Cyan	no change	invert	invert
6	Complement Blue	no change	no change	invert
7	Complement Magenta	invert	no change	invert

SET_LINE_STYLE

Eight pre-defined linestyles are supported on HP 98627. All of the linestyles may be classified as being "continuous".



CLEAR_DISPLAY

A call to CLEAR_DISPLAY erases all of the graphics display.

Inquiry Escape Functions

No inquiry escape functions are supported.

Output Escape Functions

Opcode	Function
250	Specify device limits. RealArray[1] = Points (dots) per mm in X direction RealArray[2] = Points (dots) per mm in Y direction
1052	Set special drawing modes. Using this escape function will redefine the meaning of the set color attribute. See SET_COLOR for details. Out of range values default to dominate drawing mode. IntegerArray[1] = 0 Dominate drawing mode. IntegerArray[1] = 1 Non-Dominate drawing mode. IntegerArray[1] = 2 Erase drawing mode. IntegerArray[1] = 3 Complement drawing mode.

Locator Echoes on the Graphics Display

All locator echoes are supported by the HP 98627. The starting position is unaffected by echoes on the HP 98627.

HP 9111 Tablet

Description

The dimensions of the HP 9111 graphics tablets are as follows:

Active platen size:	300.8 mm wide by 217.6 mm high The entire area within the outline on the platen.
Platen addressability:	12032 points wide by 8704 points high. The entire area within the outline on the platen.
Resolution:	40.0 points/mm in X and Y directions

The default locator limits are the active platen limits.

The physical origin of the locator device is the lower left corner of the outlined area on the platen.

Initialization

LOCATOR_INIT

When the locator device is initialized, any existing errors in the data tablet are cleared.

Wait Locator Input

AWAIT_LOCATOR

When the wait locator function is invoked, the DIGITIZE light on the data tablet is turned on and the position of the stylus is constantly monitored until the stylus (locator's button) is depressed. To digitize a point, the operator positions the stylus to the desired position and depresses it. When the stylus is depressed, the DIGITIZE light is turned off, the digitized point is returned, and the value of the locator's button (always 1) is returned to the application program.

Echoes Supported

Locator input can be echoed on either a graphics display device or a locator device. For the echoes supported on a graphics display device, see the section which describes the graphics display in question.

The supported echoes on the locator device are as follows:

Echo Number	Echo Performed
0	No echo is performed.
1	The HP 9111's beeper is sounded when the stylus is depressed.
9 thru 255	Same as ECHO #1

Sample Locator Input

SAMPLE_LOCATOR

The sample locator function returns the current position of the stylus on the platen without waiting for an operator response.

Echoes Supported

The following locator echoes are supported with the HP 9111 Data Tablets when using the sample locator function:

Echo Number	Echo Performed
0	No echo is performed.
1	The HP 9111's beeper is sounded when the locator is sampled.

General HPGL Display Handler

Description

At device initialization an inquiry is made of the device as to its type. If the device responds that it is an HPGL device, but the graphics package does not recognize the particular device name, the general HPGL display handler will be used.

Since many characteristics of the device cannot be inquired in HPGL, the graphics package will make the assumption that the device has the same capabilities as the HP 9872B plotter.

The maximum physical limits of the graphics display are determined by the default settings of P1 and P2. The default settings of P1 and P2 are the values they have after an HPGL "IN" command.

The default logical display surface is set equal to the area defined by P1 and P2 at the time DISPLAY_INIT is invoked.

The viewsurface is always justified in the lower left corner of the current logical display surface.

Initialization

DISPLAY_INIT

When the HPGL display is initialized the following device dependent actions are performed:

- The starting position is undefined.
- The DISPLAY_INIT *control* parameter is ignored by this device handler.

Primitive Attributes

SET_COLOR

The general handler assumes the HPGL device can support up to 32767 pens. Color values larger than the number that a given device supports may produce device dependent errors. For additional information on a given HPGL device, refer to the 'SP' command in that device's programming manual.

SET_LINE_STYLE

Seven pre-defined linestyles are supported on the HP 9872 plotters. All linestyles supported on the HP 9872 plotters may be classified as being "continuous". The HPGL display drivers assume the same is true.

CLEAR_DISPLAY
A call to CLEAR_DISPLAY sends an advance full page command to the device. Any error generated by the device is cleared.

Inquiry Escape Functions

No inquiry escape functions are supported.

Output Escape Functions

No output escape functions are supported.

Locator Echoes on the Graphics Display

Same as for the HP 9872; see that section.

General HPGL Locator Handler

Description

At device initialization an inquiry is made of the device as to its type. If the device responds that it is an HPGL device, but the graphics package does not recognize the particular device name, the general HPGL drivers will be used.

Since many characteristics of the device cannot be inquired in HPGL, the graphics package will make the assumption that the device has the same capabilities as the HP 9872B plotter.

The maximum physical limits of the graphics locator are determined by the default settings of P1 and P2. The default settings of P1 and P2 are the values they have after an HPGL 'IN' command.

The default logical locator limits are set equal to the area defined by P1 and P2.

Initialization

LOCATOR_INIT
When the locator device is initialized, the graphics display is left unaltered.

Wait Locator Input

AWAIT_LOCATOR
See HP 9872 device handler.

Sample Locator Input

SAMPLE_LOCATOR
See HP 9872 device handler.

Sample Programs

The following programs demonstrate the use of procedures available in the graphics system.

The first example is complete and ready to run. The second example includes a general "set-up" procedure which will be needed by later examples. In fact, several procedures introduced in early examples will be used by later examples. Examples that contain the "alias" compiler directive will need to access some of the procedures defined in earlier examples.

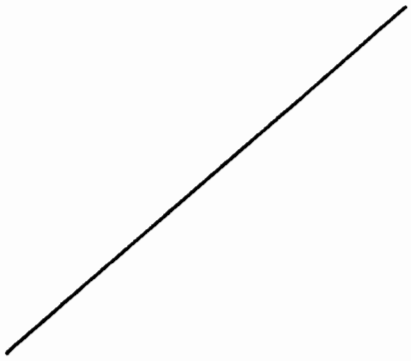
Example Programs

The following programs demonstrate the use of procedures available in the graphics system.

Each example can be compiled and executed by itself. However, some procedures in the early examples reappear in later examples. If you have already typed in a procedure appearing in a later example, the Editor's Copy command can be used to "clone" these procedures.

This program shows the initialization and and termination procedure for using the graphics library.

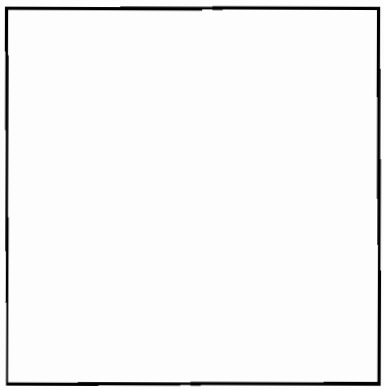
```
program example0;

import dgl_lib;                 {access graphics routines}

const
  crt_adr = 3;                  { device address of graphics crt }
  control = 0;                  { device control word; ignored for the crt }

var
  error : integer;              { display initialization error return; 0 if ok }

begin   { main program }

  graphics_init;                            { initialize the graphics libary }
  display_init (crt_adr,control,error);     { initialize display device      }

  if error = 0 then
    begin                       { draw a line using the default window }
      move (-0.5,-0.5);         { specify first end point }
      line ( 0.5, 0.5);         { draw line to second end point }
    end;

  graphics_term;                {terminate graphics library}

end.
```



Results of Program

This example shows the concept of device independence, the ability to create the same picture on any graphics device with the same set of graphics routines.

```
program example1 (input,output);

import dgl_lib;                 {access graphics routines}

var
  init_ok : boolean;            {true if initialized}

procedure setup_example ( var init_ok : boolean );

{ This procedure is used to initialize the graphics library and a user        }
{ specified output device.                                                    }

var
  display_address : integer;
  error_return    : integer;

  { Control is used only by the HP 98627A color graphics interface card.  See }
  { the graphics device appendix for details.                                 }
  control         : integer;

begin
  { Initialize the graphics library.  Graphics_init must be the first         }
  { graphics procedure called in the graphics library.                        }
  graphics_init;

  { Request the graphic display address from the user. (#12 clears the crt,   }
  { #10 generates a line feed).                                               }
  writeln (#12,'Enter graphics display address',#10);
  writeln ('Internal graphics CRT = 3');
  writeln ('External plotters     = HPIB address',#10);
  prompt  ('> ');
  readln  (display_address);
  writeln (#12);

  { initialize the graphics display device }
  display_init ( display_address, control, error_return );

  if error_return <> 0 then
    begin
      init_ok := false;
      writeln('Display initialization error #',error_return:1)
    end
  else
    init_ok := true;
end; { setup_example }

begin { main program }
  setup_example (init_ok);
  if init_ok then
    begin
      { draw a box using the default window }
      move (-0.5,-0.5);
      line (-0.5, 0.5);
      line ( 0.5, 0.5);
      line ( 0.5,-0.5);
      line (-0.5,-0.5);
    end;
  graphics_term;                {terminate the graphics library}
end.
```



Results of Program

This program shows the concept of graphics output primitives. The output primitives of move, line, and text will be shown.

Graphics output primitives are the building blocks of a graphics picture. Just as an algorithm is broken into the simplest possible instructions to create a computer program, a picture may be broken down into graphics output primitives. The graphics library uses three types of graphics output primitives: moves, lines, and text. Each output primitive starts where the previous output primitive ended. This means that the starting point of an output primitive is not explicitly specified. The move primitive can be used to redefine the starting position of the next primitive.

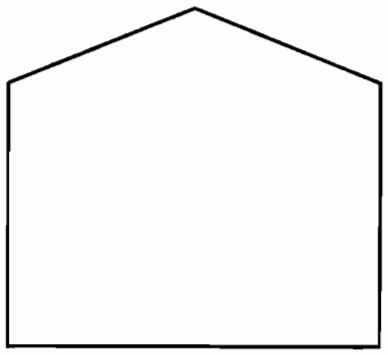
The output primitives are accessed using five procedures provided by the graphics library. They are; move, int_move, line, int_line, and gtext. See the language reference section for a description of each procedure.

The location of each output primitive is specified in the world coordinate system. The world coordinate system is a two-dimensional space which is expressed in units that are selected by the user using the set_window procedure covered in a later example.

The default value for the world coordinate system (-1.0 to 1.0 in the X and Y axes) is used in this example.

```
program example2 (input,output);

import dgl_lib;                 { access graphics routines }

var
  init_ok : boolean;            { true if initialized }

procedure setup_example ( var init_ok : boolean );

var
  display_address, control, error_return : integer;

begin
  display_address := 3;   { internal CRT }
  control         := 0;
  error_return    := 0;

  graphics_init;
  display_init ( display_address, control, error_return );

  if error_return <> 0 then
    begin
      init_ok := false;
      writeln('Display initialization error #',error_return:1)
    end
  else
    init_ok := true;
end; { setup_example }

Procedure draw_house;
```

```
{ This procedure draws a house using move and line graphics output primitives }

begin
  move (-0.5,-0.5);
  line ( 0.5,-0.5);
  line ( 0.5, 0.2);
  line ( 0.0, 0.4);
  line (-0.5, 0.2);
  line (-0.5,-0.5);
end;

begin
  setup_example (init_ok);
  if init_ok then
    begin
      draw_house;                 { Plot a house }
      move (-0.5, 0.6);           { Add some text }
      gtext ('HOME SWEET HOME');
    end;
  graphics_term;                  { terminate the graphics library }
end.
```

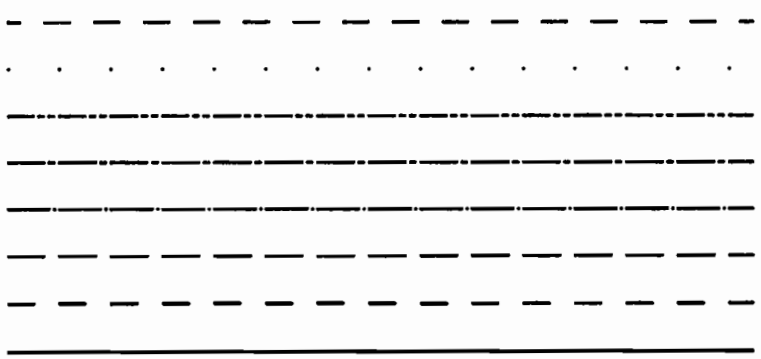


Results of Program

The following examples (3,4 and 5) show the concept of primitive attributes.

Primitive attributes are the general characteristics of output primitives. They affect the appearance of individual output primitives. This example shows how the set_line_style and set_color procedures affect the line primitive.

```
program example3 (input,output);

import dgl_lib;                 { access graphics routines }

var
  init_ok : boolean;            { true if initialized }
  index   : integer;            { do loop index }

procedure setup_example ( var init_ok : boolean );

var
  display_address,control, error_return : integer;

begin
  display_address := 3;   { internal CRT }
  control         := 0;
  error_return    := 0;

  graphics_init;
  display_init ( display_address, control, error_return );

  if error_return <> 0 then
    begin
      init_ok := false;
      writeln('Display initialization error #',error_return:1)
    end
  else
    init_ok := true;
end; { setup_example }

begin  { main program }

  setup_example (init_ok);
  if init_ok then
    { draw 8 lines using colors 1 through 8 and linestyles 1 through 8 }
    for index := 1 to 8 do
      begin
        set_color(index);
        set_line_style(index);
        move(-1.0,-0.5 + index / 8.0);
        line( 1.0,-0.5 + index / 8.0);
      end;
  graphics_term;                { terminate the graphics library }
end.
```



Results of Program

This example shows how the set_char_size procedure is used to change the character size of text.

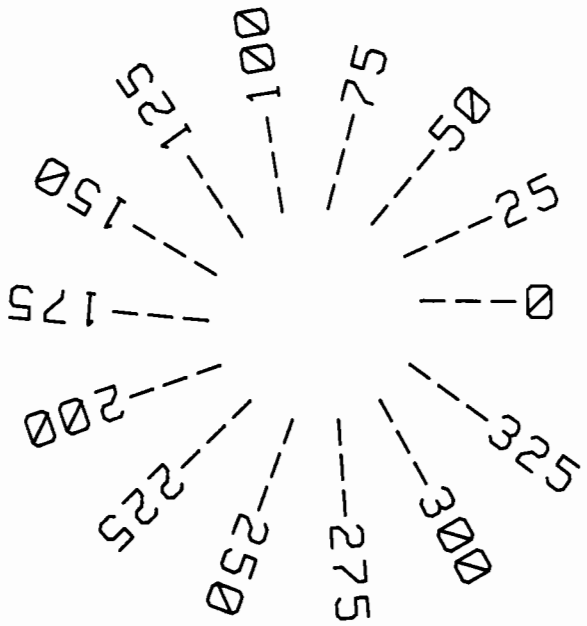
```
program example4 (input,output);

import dgl_lib;                    { access graphics routines }

var
  init_ok : boolean;               { true if initialized }

procedure setup_example ( var init_ok : boolean );

var
  display_address,control, error_return : integer;

begin
  display_address := 3;   { internal CRT }
  control         := 0;
  error_return    := 0;

  graphics_init;
  display_init ( display_address, control, error_return );

  if error_return <> 0 then
    begin
      init_ok := false;
      writeln('Display initialization error #',error_return:1)
    end
  else
    init_ok := true;
end; { setup_example }

begin { main program }

  setup_example (init_ok);
  if init_ok then
    begin
      { plot strings using different character sizes }
      move (-1.0, 0.8);
      set_char_size (0.045,0.075);
      gtext ('( 0.045, 0.075 )');

      move (-1.0, 0.4);
      set_char_size (0.09,0.15);
      gtext ('( 0.09, 0.15 )');

      move (-1.0, 0.0);
      set_char_size (0.045,0.15);
      gtext ('( 0.045,0.15 )');

      move (-1.0,-0.4);
      set_char_size (0.11,0.075);
      gtext ('( 0.11, 0.075 )');

    end;
  graphics_term;                   { terminate the graphics library }
end.
```

```
( 0.045, 0.075 )

( 0.09, 0.15 )

( 0.045,0.15 )

( 0.11, 0.075 )
```

Results of Program

This example shows how the procedure set_text_rot is used to set the angle at which text is rotated.

```
program example5 (input,output);

import dgl_lib;                { access graphics routines }

var
  deg      : integer;
  cnt      : integer;
  s        : string[40];
  init_ok  : boolean;          { true if initialized }

procedure setup_example ( var init_ok : boolean );

var
  display_address,control, error_return : integer;

begin
  display_address := 3;   { internal CRT }
  control         := 0;
  error_return    := 0;

  graphics_init;
  display_init ( display_address, control, error_return );

  if error_return <> 0 then
    begin
      init_ok := false;
      writeln('Display initialization error #',error_return:1)
    end
  else
    init_ok := true;
end; { setup_example }

begin { main program }
  setup_example (init_ok);
  if init_ok then
    begin
      set_char_size (0.09,0.15);

      { plot text using many different text directions }
      deg := 0;
      repeat
        move(0.0,0.0);                          { plot from center of display }

        set_text_rot (cos(deg*3.14/180),sin(deg*3.14/180));   { set rotation }

        s := '   --- ';                         { create string to be plotted }
        strwrite(s,7,cnt,deg:1);

        gtext(s);                               { plot the string }

        deg := deg + 25;
      until deg > 340;

    end;
  graphics_term;    { terminate the graphics library }
end.
```



Results of Program

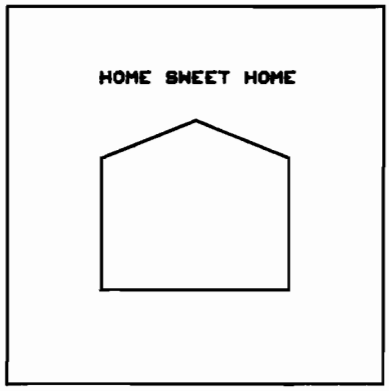
The next 4 examples (6 thru 9) show aspects of the viewing transformation. Refer to figure 1.0 for an overall diagram of the viewing transformation.

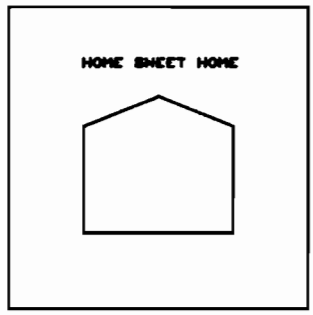
This program shows the set_display_lim procedure. Set_display_lim specifies a subset of the physical display surface to be used for graphics output. This area is called the logical display limits. The limits of this area are expressed in terms of millimeters, offset from the physical orgin of the device. The location of the physical origin, and the default limits of a display device are device dependent, and are specified in the device dependent appendix of this document.

The clipping limits are set to be equal to the logical display limits in the graphics library. Clipping is the elimination from view of all visible primitives or parts of primitives which lie outside the clipping limits.

```
program example6 (input,output);

import dgl_lib;                 { access graphics routines }

var
  init_ok : boolean;            { true if initialized }
  error_return : integer;

procedure setup_example ( var init_ok : boolean );

var
  display_address,control, error_return : integer;

begin
  display_address := 3;   { internal CRT }
  control         := 0;
  error_return    := 0;

  graphics_init;
  display_init ( display_address, control, error_return );

  if error_return <> 0 then
    begin
      init_ok := false;
      writeln('Display initialization error #',error_return:1)
    end
  else
    init_ok := true;
end; { setup_example }

procedure draw_house;

begin
  move (-0.5,-0.5);
  line ( 0.5,-0.5);
  line ( 0.5, 0.2);
  line ( 0.0, 0.4);
  line (-0.5, 0.2);
  line (-0.5,-0.5);
end;

procedure draw_box (xmin,xmax,ymin,ymax : real);

begin
  move (xmin,ymin);
  line (xmax,ymin);
  line (xmax,ymax);
  line (xmin,ymax);
  line (xmin,ymin);
end;
```

```
begin
  setup_example (init_ok);
  if init_ok then
    begin
      { Define the logical display limits to be located between 0 mm and     }
      { 50 mm in the X and Y directions                                      }
      set_display_lim(0.0,50.0,0.0,50.0,error_return);

      { Draw a box around the current window limits }
      draw_box (-1.0,1.0,-1.0,1.0);

      draw_house;                { Draw a picture }
      move (-0.5, 0.6);
      gtext ('HOME SWEET HOME');

      { Define the logical display limits to be located between 60 mm and    }
      { 100 mm in the X direction, and between 30 mm and 70 mm in the Y      }
      { direction.                                                           }
      set_display_lim(60.0,100.0,30.0,70.0,error_return);

      { Draw a box around the current window limits }
      draw_box (-1.0,1.0,-1.0,1.0);


      draw_house;                { Draw the same picture as above }
      move (-0.5, 0.6);
      gtext ('HOME SWEET HOME');

      { Note that the same picture is drawn, only its location and size on   }
      { the plotter have changed.                                            }

      { Also note that the size of the box around the pictures measures to   }
      { the size specified in the set_display_lim procedure.                 }

      { The next example will show how the set_aspect ratio procedure        }
      { interacts with set_display_lim.                                      }
    end;
  graphics_term;          { terminate the graphics library }
end.
```





Results of Program

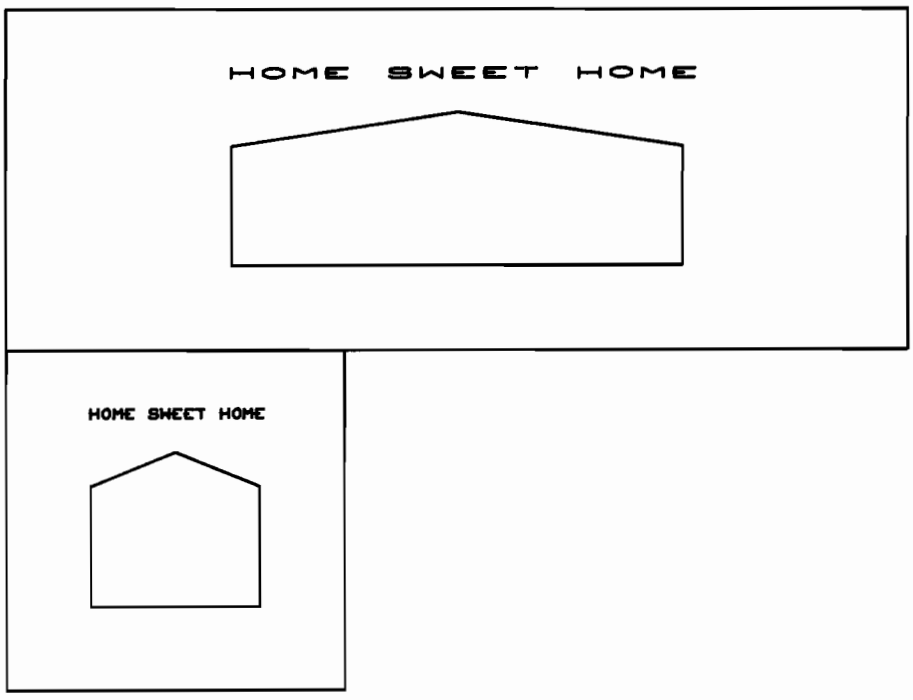
This program shows the set_aspect procedure. Set_aspect sets the aspect ratio of the virtual coordinate system, and hence the aspect ratio of the view surface (the area in which all plotting occurs) to be height divided by width. A ratio of 1.0 defines a square virtual coordinate system; a ratio greater than 1.0 specifies it to be higher than it is wide; and a ratio less than 1.0 specifies it to be wider than it is high.

The initial aspect ratio of the virtual coordinate system is 1.0, meaning the virtual coordinate system is a unit square. This produces a view surface that is the largest inscribed square within the logical display limits (set by set_display_lim). By changing the aspect ratio, the view surface defines the largest inscribed rectangle within the logical display limits.

The placement of the view surface is dependent upon the device being used. It is generally centered on CRT displays and is usually placed in the lower left-hand corner of plotters.

```
program example7 (input,output);

import dgl_lib;                 { access graphics routines }

var
 init_ok : boolean;             { true if initialized }
 error_return : integer;


procedure setup_example ( var init_ok : boolean );

var
  display_address,control, error_return : integer;

begin
  display_address := 3;    { internal CRT }
  control         := 0;
  error_return    := 0;

  graphics_init;
  display_init ( display_address, control, error_return );

  if error_return <> 0 then
    begin
      init_ok := false;
      writeln('Display initialization error #',error_return:1)
    end
  else
    init_ok := true;
end; { setup_example }

procedure draw_house;

begin
  move (-0.5,-0.5);
  line ( 0.5,-0.5);
  line ( 0.5, 0.2);
  line ( 0.0, 0.4);
  line (-0.5, 0.2);
  line (-0.5,-0.5);
end;

procedure draw_box (xmin,xmax,ymin,ymax : real);

begin
  move (xmin,ymin);
  line (xmax,ymin);
  line (xmax,ymax);
  line (xmin,ymax);
  line (xmin,ymin);
end;
```

```
begin
 setup_example (init_ok);
 if init_ok then
  begin
   { Use set_display_lim to define the logical display limits to be a        }
   { rectangle on the lower portion of the display.                          }
   set_display_lim(0.0,120.0,0.0,45.0,error_return);

   { Set the view surface to have a 1:1 aspect ratio. }
   set_aspect(1,1);

   { draw a picture with a box around the current window }
   draw_box (-1.0,1.0,-1.0,1.0);
   draw_house;
   move (-0.5, 0.6);
   stext ('HOME SWEET HOME');

   { Note that only a portion of the area defined with set_display_lim    }
   { is used.                                                             }

   { Use set_display_lim to define the logical display limits to be a     }
   { rectangle the same size and shape as above, but located above the    }
   { last rectangle.                                                      }
   set_display_lim(0.0,120.0,45.0,90.0,error_return);

   { Set the view surface to have the same aspect ratio as the rectangle. }
   set_aspect(120.0,45.0);

   draw_box (-1.0,1.0,-1.0,1.0);          { draw the same picture as above }

   draw_house;
   move (-0.5, 0.6);
   stext ('HOME SWEET HOME');

   { Note that the picture fills the area defined with set_display_lim.   }
   { Also note that distortion has occured in the picture.  This will be  }
   { discused in a later example.                                         }

  end;
 graphics_term;                          { terminate the graphics library }
end.
```



Results of Program

This program shows the set_viewport procedure. Set_viewport sets the limits of the viewport, the area onto which the window map, in units of the virtual coordinate system. The graphics library calculates the range of the virtual coordinate system based on the value of the aspect ratio. The coordinates of the longer axis are always set to range from 0.0 to 1.0, and those of the shorter axis from 0.0 to a value that achieves the specified aspect ratio.

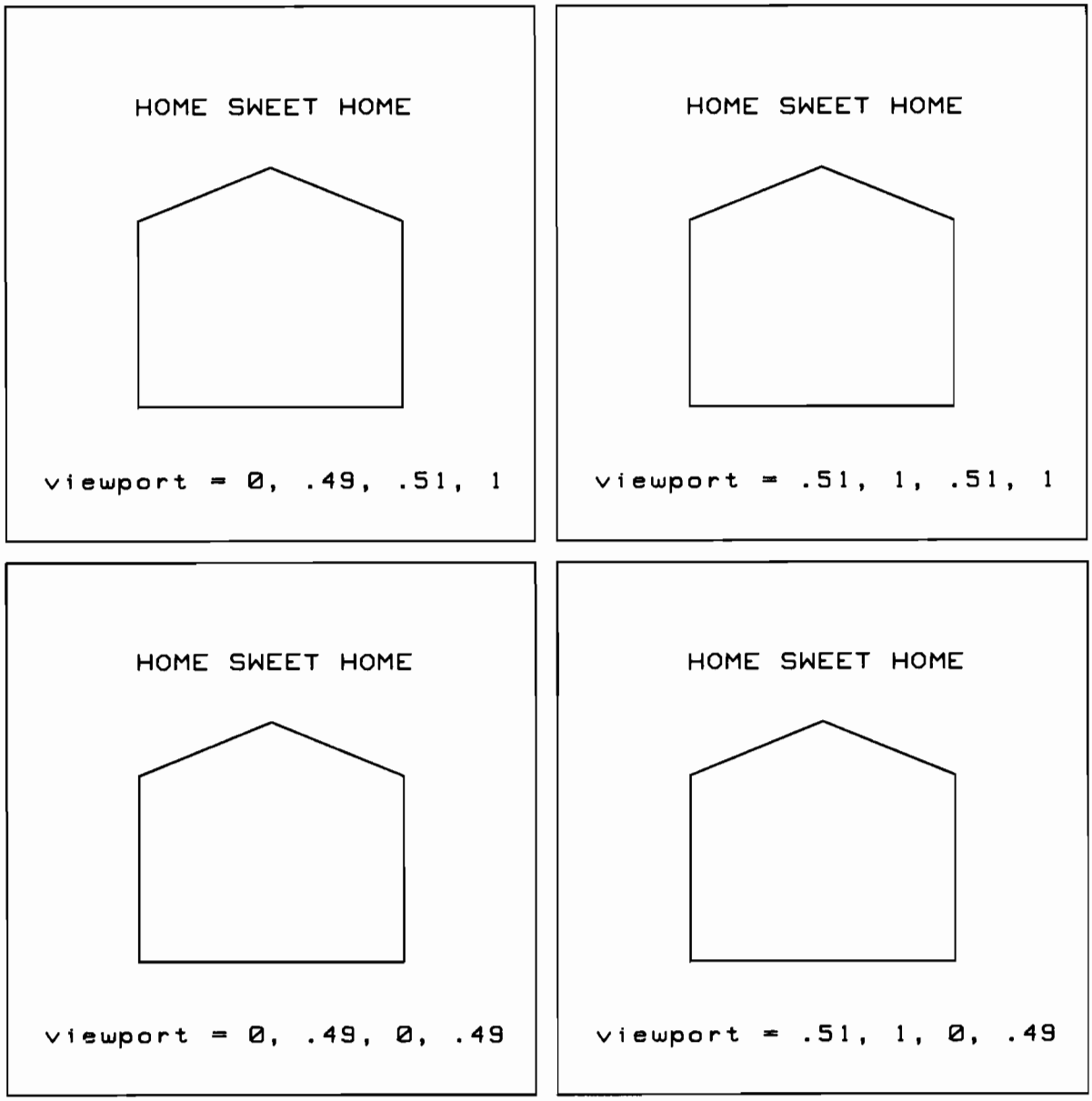
ASPECT RATIO	X LIMITS	Y LIMITS
AR < 1	0,1	0,1/AR
AR = 1	0,1	0,1
AR > 1	0,1/AR	0,1

Since the inital aspect ratio is 1:1, the initial viewport is mapped onto the maximum visible square within the logical display limits.

By changing the limits of the viewport, an application program can display an image in several different positions on the same display surface. This program shows an image sequentially displayed in the four corners of a display surface.

```
program example8 (input,output);

import dgl_lib;                 { access graphics routines }

var
  init_ok : boolean;            { true if initialized }
  error_return : integer;

procedure setup_example ( var init_ok : boolean );

var
  display_address,control, error_return :integer;

begin
  display_address := 3;   { internal CRT }
  control         := 0;
  error_return    := 0;

  graphics_init;
  display_init ( display_address, control, error_return );

  if error_return <> 0 then
    begin
      init_ok := false;
      writeln('Display initialization error #',error_return:1)
    end
  else
    init_ok := true;
end; { setup_example }

procedure draw_house;

begin
  move (-0.5,-0.5);
  line ( 0.5,-0.5);
  line ( 0.5, 0.2);
  line ( 0.0, 0.4);
  line (-0.5, 0.2);
  line (-0.5,-0.5);
end;

procedure draw_box (xmin,xmax,ymin,ymax : real);
```

```
begin
  move (xmin,ymin);
  line (xmax,ymin);
  line (xmax,ymax);
  line (xmin,ymax);
  line (xmin,ymin);
end;

begin
  setup_example (init_ok);
  if init_ok then
    begin
      { Using four different viewports, display the same picture. }

      set_viewport(0.0,0.49,0.0,0.49);
      draw_box (-1.0,1.0,-1.0,1.0);
      draw_house;
      move (-0.5, 0.6);
      gtext ('HOME SWEET HOME');
      move (-0.85,-0.8);
      gtext ('viewport = 0, .49, 0, .49');

      set_viewport(0.51,1.0,0.0,0.49);
      draw_box (-1.0,1.0,-1.0,1.0);
      draw_house;
      move (-0.5, 0.6);
      gtext ('HOME SWEET HOME');
      move (-0.85,-0.8);
      gtext ('viewport = .51, 1, 0, .49');

      set_viewport(0.0,0.49,0.51,1.0);
      draw_box (-1.0,1.0,-1.0,1.0);
      draw_house;
      move (-0.5, 0.6);
      gtext ('HOME SWEET HOME');
      move (-0.85,-0.8);
      gtext ('viewport = 0, .49, .51, 1');

      set_viewport(0.51,1.0,0.51,1.0);
      draw_box (-1.0,1.0,-1.0,1.0);
      draw_house;
      move (-0.5, 0.6);
      gtext ('HOME SWEET HOME');
      move (-0.85,-0.8);
      gtext ('viewport = .51, 1, .51, 1');
    end;
  graphics_term;    { terminate the graphics library }
end.
```



Results of Program

This example shows the set_window procedure.

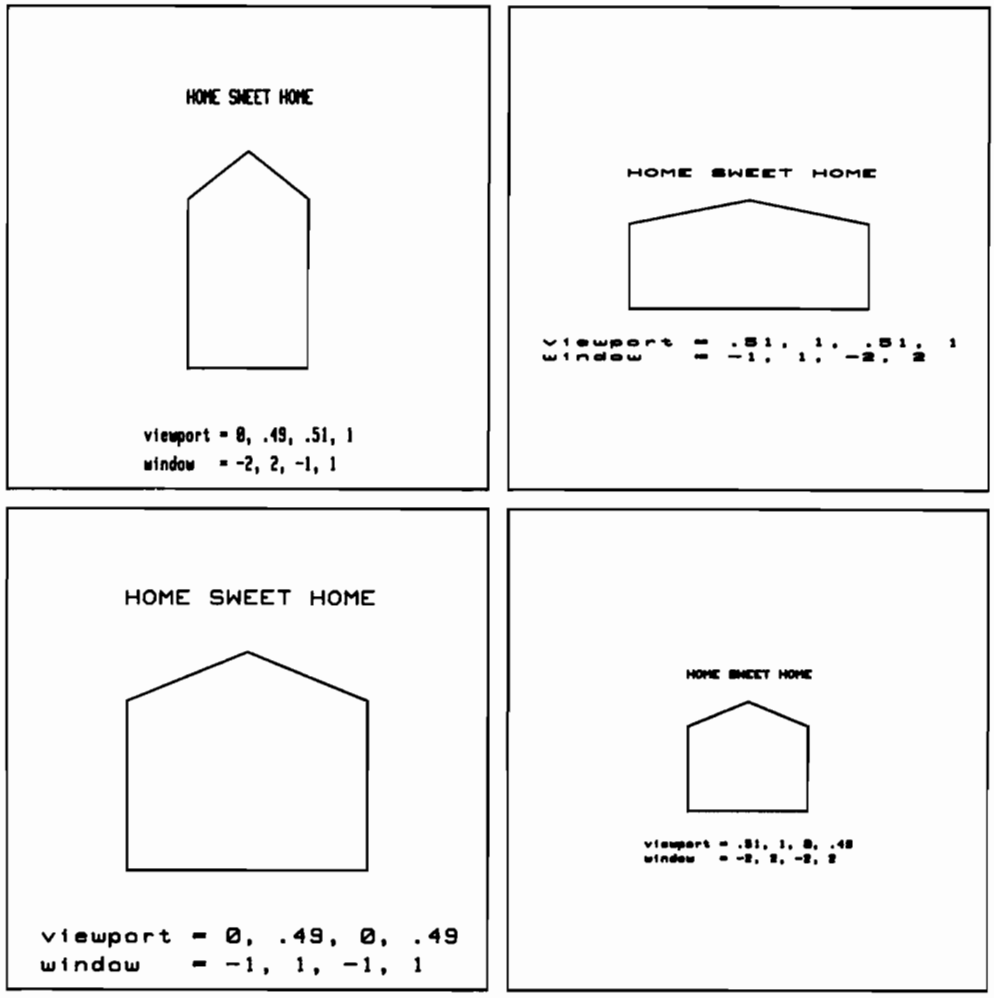
Set_window specifies the portion of the world coordinate system which maps onto the viewport. Setting the window allows the application program to define which portion of the world coordinate space is to be viewed. This provides the application program with the flexibility of working in units that are relevent to the application. Since the window is in the same coordinate space in which objects are defined, the bounds of the window can affect the size of the image displayed. The larger the limits of the window, the smaller the object's image. If, however, the window is specifed as having smaller limits than the object, portions of the object will be plotted outside the viewport bounds.

The window is defined in units of the world coordinate system. The window's aspect ratio should be the same as the aspect ratio of the viewport if distortion is not desired. In general, setting the window by calculating its dimensions as a function of the viewport dimensions is a good way to prevent distortion.

This example shows the same object drawn with different window dimensions.

```
program example9 (input,output);

import dgl_lib;                { access graphics routines }

var
  init_ok : boolean;           { true if initialized }
  error_return : integer;

procedure setup_example ( var init_ok : boolean );

var
  display_address,control, error_return : integer;

begin
  display_address := 3;   { internal CRT }
  control         := 0;
  error_return    := 0;

  graphics_init;
  display_init ( display_address, control, error_return );

  if error_return <> 0 then
    begin
      init_ok := false;
      writeln('Display initialization error #',error_return:1)
    end
  else
    init_ok := true;
end; { setup_example }

procedure draw_house;

begin
  move (-0.5,-0.5);
  line ( 0.5,-0.5);
  line ( 0.5, 0.2);
  line ( 0.0, 0.4);
  line (-0.5, 0.2);
  line (-0.5,-0.5);
end;

procedure draw_box (xmin,xmax,ymin,ymax : real);
```

```
begin
  move (xmin,ymin);
  line (xmax,ymin);
  line (xmax,ymax);
  line (xmin,ymax);
  line (xmin,ymin);
end;

begin
  setup_example (init_ok);
  if init_ok then
    begin
      { draw the object with the default window of -1,1,-1,1 }
      set_viewport(0.0,0.49,0.0,0.49);
      set_window  (-1.0,1.0,-1.0,1.0);
      draw_box (-1.0,1.0,-1.0,1.0);
      draw_house;
      move (-0.5, 0.6);
      gtext ('HOME SWEET HOME');
      move (-0.85,-0.8);
      gtext ('viewport = 0, .49, 0, .49');
      move (-0.85,-0.92);
      gtext ('window   = -1, 1, -1, 1');

      { draw the object with a window 2 times larger. }
      set_viewport(0.51,1.0,0.0,0.49);
      set_window  (-2.0,2.0,-2.0,2.0);
      draw_box (-2.0,2.0,-2.0,2.0);
      draw_house;
      move (-0.5, 0.6);
      gtext ('HOME SWEET HOME');
      move (-0.85,-0.8);
      gtext ('viewport = .51, 1, 0, .49');
      move (-0.85,-0.92);
      gtext ('window   = -2, 2, -2, 2');

      { draw the object with distortion in the X axes }
      set_viewport(0.0,0.49,0.51,1.0);
      set_window  (-2.0,2.0,-1.0,1.0);
      draw_box (-2.0,2.0,-1.0,1.0);
      draw_house;
      move (-0.5, 0.6);
      gtext ('HOME SWEET HOME');
      move (-0.85,-0.8);
      gtext ('viewport = 0, .49, .51, 1');
      move (-0.85,-0.92);
      gtext ('window   = -2, 2, -1, 1');

      { draw the object with distortion in the Y axes }
      set_viewport(0.51,1.0,0.51,1.0);
      set_window  (-1.0,1.0,-2.0,2.0);
      draw_box (-1.0,1.0,-2.0,2.0);
      draw_house;
      move (-0.5, 0.6);
      gtext ('HOME SWEET HOME');
      move (-0.85,-0.8);
      gtext ('viewport = .51, 1, .51, 1');
      move (-0.85,-0.92);
      gtext ('window   = -1, 1, -2, 2');

      { note that character size is affected by the window. }
    end;
  graphics_term;     { terminate the graphics library }
end.
```



Results of Program

This program shows how locator input is used. The locator device returns an (X,Y) point in the world coordinate system. Typical locator devices are digitizers and graphics cursors on CRTs.

A call to await_locator causes the application program to wait until a locator button is pressed. The value of the selected button and a world coordinate point are then returned to the calling program.

Several different types of echoing can be performed. Some echoes are performed only on the locator device, these echoes may include blinking a light or sounding a bell each time a point is entered. Other echoes are performed on the graphics display device. All locator echoes on the graphics display begin at a world coordinate point called the locator echo positon. Some echoes may also use the locator echo position as a reference point. For example, many devices support a rubberband line echo. The fixed end of the rubber band line will be at the locator echo position.

This example uses set_echo_pos, and await_locator to enter 6 points, which will beused as endpoints of 5 lines.

For details on the operation of the KNOB or other graphics device, refer to the device dependent sections of this document.

```
program example10 (input,output);

import dgl_lib;          { access graphics library }

var
 display_address : integer;
 locator_address : integer;
 control_word    : integer;
 error_return    : integer;
 x               : real;
 y               : real;
 button          : integer;
 i               : integer;


begin
 graphics_init; { initialize graphics library }

 { set up the viewing transformation to use a large portion of the logical }
 { display surface                                                         }
 set_aspect(4,3);
 set_window(0,3,0,2);

 { Get display address }
 writeln (#12,'Enter graphics display address',#10);
 writeln ('Internal graphics CRT = 3');
 writeln ('External plotters     = HPIB address',#10);
 prompt  ('> ');
 readln  (display_address);
 write   (#12);

 { initialize graphics display }
 display_init ( display_address, control_word, error_return );

 { check if an error occured }
 if error_return <> 0 then
  writeln('Display initialization error #',error_return:1)
 else
  begin
```

```
    { set locator address }
    writeln (#12,'Enter graphics locator address',#10);
    writeln ('Knob                = 2');
    writeln ('External locators = HPIB address',#10);
    prompt  ('> ');
    readln  (locator_address);
    write   (#12);

    { initialize graphics locator }
    locator_init ( locator_address, error_return);

    { check if an error occured }
    if error_return <> 0 then
     writeln('Locator initialization error #',error_return:1)
   end;


  { check if an error occured }
  if error_return = 0 then
   begin
    { set first point using small cross hair cursor }
    await_locator (2,button,x,y);
    move (x,y);

    { set the locator echo position to that point }
    set_echo_pos(x,y);

    { get a point, set the locator echo position to that point, and then }
    { draw a line to that point. }
    for i:= 1 to 5 do
     begin
      await_locator (4,button,x,y);
      set_echo_pos(x,y);
      line (x,y);
     end;
   end;
  graphics_term;                { terminate the graphics library }
end.
```

```
{ This example shows several uses of the OUTPUT_ESC procedure.                }
{                                                                             }
{ In addition to providing a variety of device independent functions, the     }
{ graphics library provides a mechanism to access some capabilities of       }
{ devices that are not supported in a device independent manner.  For        }
{ example HPGL plotters have the ability to change the speed at which the    }
{ plotter pen is moved.  The graphics library does not have a device         }
{ independent procedure to change the pen speed, but it is still a useful    }
{ feature.  The graphics library, therefore, provides a standard way to      }
{ access device dependent features.  The mechanism by which these features   }
{ are accessed is called graphics escape functions.  There are two types of  }
{ graphics escape functions.  Output graphics escape functions, implemented  }
{ by OUTPUT_ESC, provide access to special features of the graphic display   }
{ device.  INPUT_ESC allows a program to inquire about the capabilities of   }
{ the device.  The feature to be accessed is specified by the value of an    }
{ opcode passed to the procedure.  The value of opcode may differ for each   }
{ device, and can be found in the Device Handlers section of this manual.    }
{ If the opcode specified is not supported by a particular device or if the  }
{ parameters are specified incorrectly, an error will be returned and the    }
{ procedure will be ignored.                                                 }
{                                                                             }

{ The following 4 example procedures show how OUTPUT_ESC can be used to      }
{ perform device dependent actions on the graphics display.  Each example    }
{ has a section of code which should be added to the main program type       }
{ declaration area.                                                          }

{ ********************************    1    ********************************* }

type
  rmode = (dominate,erase,complement);

procedure set_raster_drawing_mode ( mode : rmode);

{ This procedure will change the current drawing mode.  }

const
  change_drawing_mode = 1052;

var
  ilist : array [1..1] of integer;
  rlist : array [1..1] of real;
  error : integer;

begin
  case mode of
    dominate :    ilist[1] := 0;  { set bits in display memory }
    erase    :    ilist[1] := 2;  { clear bits in display memory }
    complement :  ilist[1] := 3;  { invert bits in display memory }
  end; { of case }

  output_esc (change_drawing_mode,1,0,ilist,rlist,error);
  if error <> 0 then writeln('OUTPUT_ESC opcode not supported');
end;

{ ********************************    2    ********************************* }

procedure set_plotter_speed ( speed : integer );

{ This procedure will change the speed that a 'HPGL' plotter pen draws.  }
{ The speed passed in must be in the range 1 to 36 (cm/sec)              }

const
  change_speed = 2050;

var
  ilist : array [1..2] of integer;
  rlist : array [1..1] of real;
  error : integer;
```

```
begin
  ilist [1] := speed;
  ilist [2] := 0;     { change speed for all pens }

  output_esc (change_speed,2,0,ilist,rlist,error);
  if error <> 0 then writeln('OUTPUT_ESC opcode not supported');
end;

{ ********************************    3    ******************************** }

type
  switch = (on,off);

procedure turn_graphics ( mode : switch );

{ This procedure will turn the graphics display on or off. }

const
  gon_goff = 1050;

var
  ilist : array [1..1] of integer;
  rlist : array [1..1] of real;
  error : integer;

begin
  if mode = on then
    ilist [1] := 1
  else
    ilist [1] := 0;

  output_esc (gon_goff,1,0,ilist,rlist,error);
  if error <> 0 then writeln('OUTPUT_ESC opcode not supported');
end;

{ ********************************    4    ******************************** }

type
  switch = (on,off);

procedure turn_alpha ( mode : switch );

{ This procedure will turn the alpha display on or off. }

const
  aon_goff = 1051;

var
  ilist : array [1..1] of integer;
  rlist : array [1..1] of real;
  error : integer;

begin
  if mode = on then
    ilist [1] := 1
  else
    ilist [1] := 0;

  output_esc (aon_goff,1,0,ilist,rlist,error);
  if error <> 0 then writeln('OUTPUT_ESC opcode not supported');
end;
```

Deviations from HP 1000 Graphics

In general, the HP 9826/9836 Pascal Graphics Procedure Library is a strict subset of the DGL Graphics Procedure Library used on the HP 1000. Graphics programs written using DGL 1000 will usually transport with little reprogramming effort. Differences between 1000 DGL and DGL libraries are listed below.

Procedure Syntax

DGL 1000 uses five letter procedure names. The procedure names for 9826/9836 DGL have been lengthened to make them more descriptive. The table below lists the DGL procedure names and the equivalent 9826/9836 names. When transporting programs, the existing DGL 1000 INCLUDE file can be used along with the ALIAS compiler directive to convert to DGL 9826/9836 names.

Text Procedures

The ZTEXT procedure in DGL 1000 requires a parameter of type PACKED ARRAY OF CHAR. The equivalent GTEXT procedure in 9826/9836 DGL expects a parameter of type STRING. When transporting programs, the parameter type should be changed either manually or automatically by a specially written procedure.

A new call, SET_TEXT_ROT, has been added to 9826/9836 DGL. Text rotation is supported by DGL 1000 for certain devices using escape functions. When transporting programs, refer to the specific device driver reference for further details.

Integer Move and Draw Procedures

The calls, INT_MOVE and INT_LINE supplied by 9826/9836 DGL are not supported by DGL 1000. When transporting to the HP 1000, the ALIAS directive can be used to convert these routines to the ZMOVE and ZDRAW routines of DGL 1000.

Multiple Displays

The HP 1000 uses a segmentation scheme to link multiple device drivers to a program. On the 9826/9836 all graphics device drivers are simultaneously resident in memory. Transported DGL 1000 programs should be revised so that the entire program is resident in memory. Transported 9826/9836 programs using multiple device drivers will have to be segmented to be run on the HP 1000.

Procedure Name Cross Reference

DGL 1000	DGL 9826/9836
ZBEGN	GRAPHICS_INIT
ZEND	GRAPHICS_TERM
ZDINT	DISPLAY_INIT
ZDEND	DISPLAY_TERM
ZLINT	LOCATOR_INIT
ZLEND	LOCATOR_TERM
ZASPK	SET_ASPECT
ZDLIM	SET_DISPLAY_LIM
ZLLIM	SET_LOCATOR_LIM
ZVIEW	SET_VIEWPORT
ZWIND	SET_WINDOW
ZMOVE	MOVE
ZDRAW	LINE
ZTEXT	GTEXT
ZCOLR	SET_COLOR
ZLSTL	SET_LINE_STYLE
ZCSIZ	SET_CHAR_SIZE
ZLOCP	SET_ECHO_POS
ZSLOC	SAMPLE_LOCATOR
ZWLOC	AWAIT_LOCATOR
ZNEWF	CLEAR_DISPLAY
ZOESC	OUTPUT_ESCAPE
ZIESC	INPUT_ESCAPE

Error Code Summary

The graphics procedures return some errors via an error code return parameter, and others are returned with the pascal work station "escape". The errors returned by the error code are generally errors that may occur in a working program while the errors returned with the escape function are generally program development errors.

When an error occurs that uses the escape function, escapecode −27 is used. Additional information may be found by invoking the function GRAPHICSERROR, which will return one of the following errors:

0 No errors since the last call to GRAPHICSERROR or since the last call to init_graphics.

1 The graphics system is not initialized.
ACTION: Call ignored.

2 The graphics display is not enabled.
ACTION: Call ignored.

3 The locator device is not enabled.
ACTION: Call ignored.

4 Echo value requires a graphics display to be enabled.
ACTION: Call completes with echo value = 1.

5 The graphics system is already initialized.
ACTION: Call ignored.

6 Illegal aspect ratio specified. X_SIZE and Y_SIZE must be greater than zero.
ACTION: Call ignored.

7 Illegal parameters specified.
ACTION: Call ignored.

8 The parameters specified are outside the physical display limits.
ACTION: Call ignored.

9 The parameters specified are outside the limits of the window.
ACTION: Call ignored.

10 The logical locator and the logical display use the same physical device. The logical locator limits cannot be redefined explicitly, they must correspond to the logical view surface limits.
ACTION: Call ignored.

11 The parameters specified are outside the current virtual coordinate system boundary.
ACTION: Call ignored.

12 The escape function requested is not supported by the graphics display device.
ACTION: Call ignored.

13 The parameters specified are outside of the physical locator limits.
ACTION: Call ignored.

The function GRAPHICSERROR returns the value of the last error generated and then clears the value of the return error. A user who is trapping errors and wishes to keep the value of the error must save it in some variable.

Chapter 4
Interfacing Concepts

Introduction

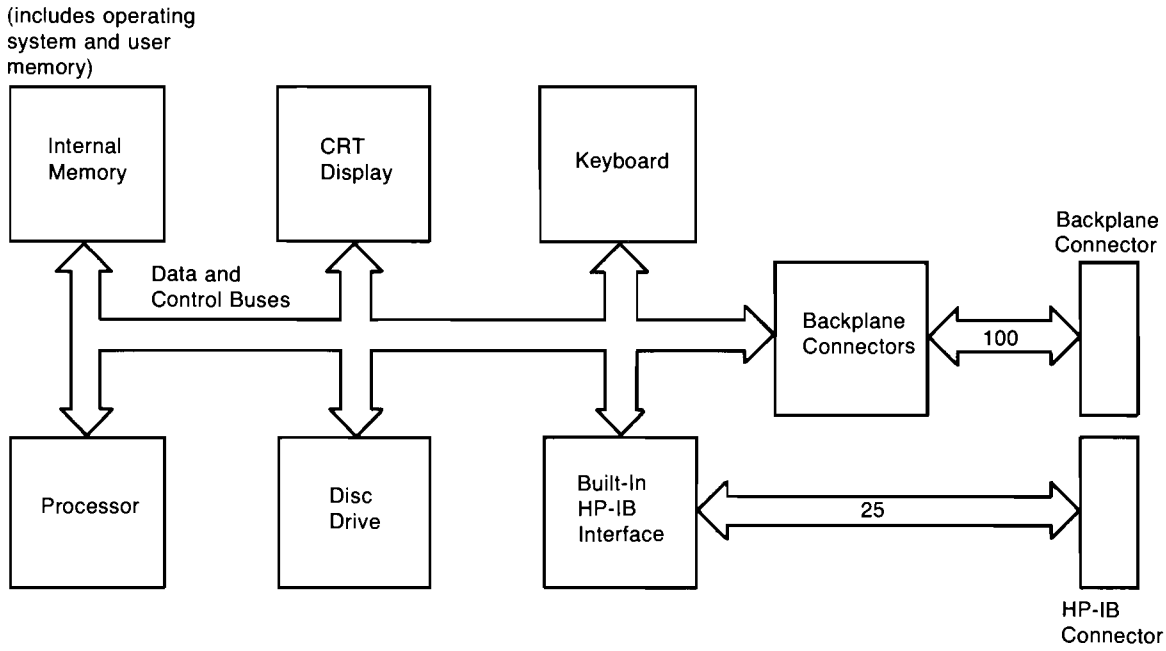
This chapter describes the functions and requirements of interfaces between the computer and its resources. Most of the concepts in this chapter are presented in an informal manner. Hopefully, **all** levels of programmers can gain useful background information that will increase their understanding of the **why** and **how** of interfacing.

Terminology

These terms are important to your understanding of the text of this manual. They are not highly technical, so don't worry about not having a PhD. in computer science to be able to understand all of them. The purpose of this section is to make sure that our terms have the same meanings.

The term **computer** is herein defined to be the processor, its support hardware, and the Pascal-language operating system; together these system elements **manage** all computer resources. The term **computer resource** is herein used to describe all of the "data-handling" elements of the system. Computer resources include: internal memory, CRT display, keyboard, and disc drive, and any external devices that are under computer control.

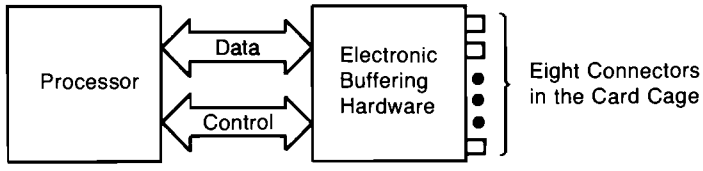
The term **hardware** describes both the electrical connections and electronic devices that make up the circuits within the computer; any piece of hardware is an actual physical device. The term **software** describes the user-written, Pascal-language programs. **Firmware** refers to the pre-programmed, machine-language programs that are invoked by Pascal-language procedures and functions. As the term implies, firmware is not modified by the user. The machine-language routines of the operating system are firmware programs.



Block Diagram of the Computer

The term **I/O** is an acronym that comes from "Input and Output"; it refers to the process of copying data to or from computer memory. Moving data from computer memory to another resource is called **output**. During output, the **source** of data is computer memory and the **destination** is any resource, including memory. Moving data from a resource to computer memory is **input**; the source is any resource and the destination is a variable in computer memory.

The term **bus** refers to a common group of hardware lines that are used to transmit information between computer resources. The computer communicates directly with the internal resources through the data and control buses. The **computer backplane** is an extension of these internal data and control buses. The computer communicates indirectly with the external resources through interfaces connected to the backplane hardware.



Backplane Hardware

Why Do You Need an Interface?

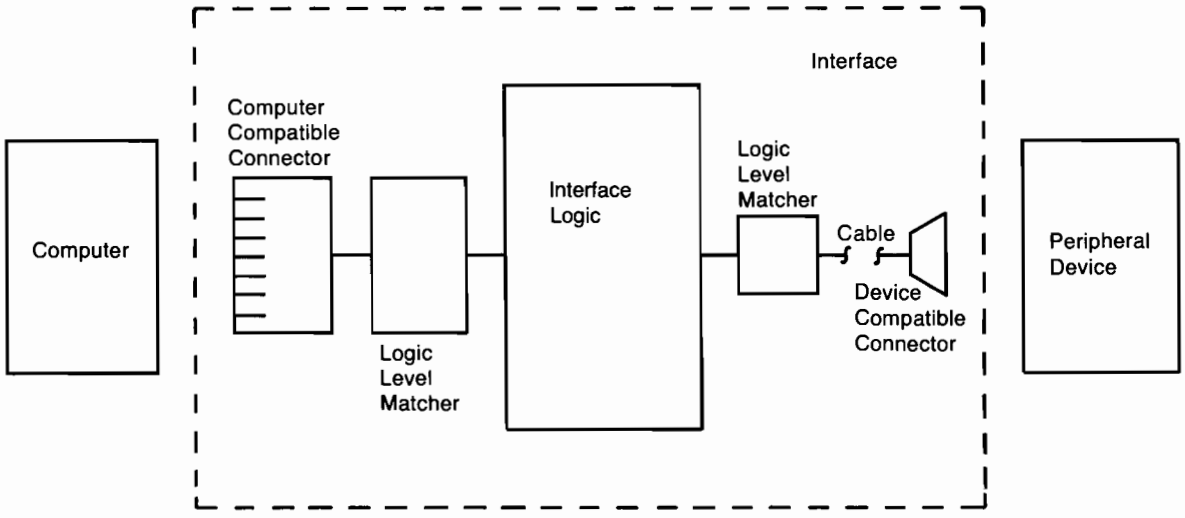
The primary function of an interface is, obviously, to provide a communication path for data and commands between the computer and its resources. Interfaces act as intermediaries between resources by handling part of the "bookkeeping" work, ensuring that this communication process flows smoothly. The following paragraphs explain the need for interfaces.

First, even though the computer backplane is driven by electronic hardware that generates and receives electrical signals, this hardware was not designed to be connected directly to external devices. The electronic backplane hardware has been designed with specific electrical logic levels and drive capability in mind. Exceeding its ratings will damage this electronic hardware.

Second, you cannot be assured that the connectors of the computer and peripheral are compatible. In fact, there is a good probability that the connectors may not even mate properly, let alone that there is a one-to-one correspondence between each signal wire's function.

Third, assuming that the connectors and signals are compatible, you have no guarantee that the data sent will be interpreted properly by the receiving device. Some peripherals expect single-bit serial data while others expect data to be in 8-bit parallel form.

Fourth, there is no reason to believe that the computer and peripheral will be in agreement as to when the data transfer will occur; and when the transfer does begin the transfer rates will probably not match. As you can see, interfaces have a great responsibility to oversee the communication between computer and its resources. The functions of an interface are shown in the following block diagram.



Functional Diagram of an Interface

Electrical and Mechanical Compatibility

Electrical compatibility must be ensured before any thought of connecting two devices occurs. Often the two devices have input and output signals that do not match; if so, the interface serves to match the electrical levels of these signals before the physical connections are made.

Mechanical compatibility simply means that the connector plugs must fit together properly. All of the 9826 interfaces have 100-pin connectors that mate with the computer backplane. The peripheral end of the interfaces may have unique configurations due to the fact that several types of peripherals are available. Most of the interfaces have cables available that can be connected directly to the device so you don't have to wire the connector yourself.

Data Compatibility

Just as two people must speak a common language, the computer and peripheral must agree upon the form and meaning of data before communicating it. As a programmer, one of the most difficult compatibility requirements to fulfill before exchanging data is that the format and meaning of the data being sent is identical to that anticipated by the receiving device. Even though some interfaces format data, most interfaces have little responsibility for matching data formats; most interfaces merely move agreed-upon quantities of data to or from computer memory. The computer must generally make the necessary changes, if any, so that the receiving device gets meaningful information.

Timing Compatibility

Since all devices do not have standard data-transfer rates, nor do they always agree as to when the transfer will take place, a consensus between sending and receiving device must be made. If the sender and receiver can agree on both the transfer rate and beginning point (in time), the process can be made readily.

If the data transfer is not begun at an agreed-upon point in time and at a known rate, the transfer must proceed one data item at a time with acknowledgement from the receiving device that it has the data and that the sender can transfer the next data item; this process is known as a "handshake". Both types of transfers are utilized with different interfaces and both will be fully described as necessary.

Additional Interface Functions

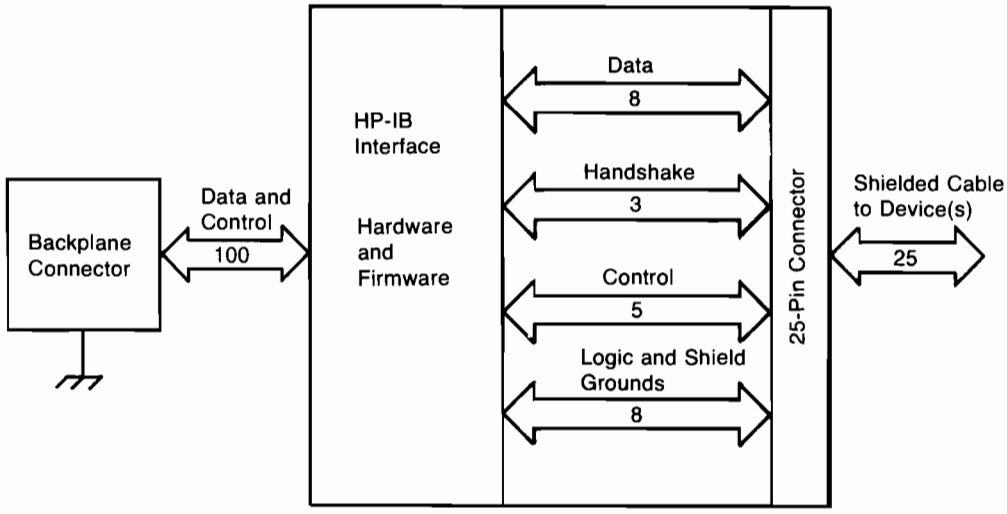
Another powerful feature of some interface cards is to relieve the computer of low-level tasks, such as performing data-transfer handshakes. This distribution of tasks eases some of the computer's burden and also decreases the otherwise-stringent response-time requirements of external devices. The actual tasks performed by each type of interface card vary widely and are described in the next section of this chapter.

Interface Overview

Now that you see the need for interfaces, you should see what kinds of interfaces are available for the 9826. Each of these interfaces is specifically designed for specific methods of data transfer; each interface's hardware configuration reflects its function.

The HP-IB Interface

This interface is Hewlett-Packard's implementation of the IEEE-488 1975 Standard Digital Interface for Programmable Instrumentation. The acronym "HP-IB" comes from Hewlett-Packard Interface Bus, often called the "bus".



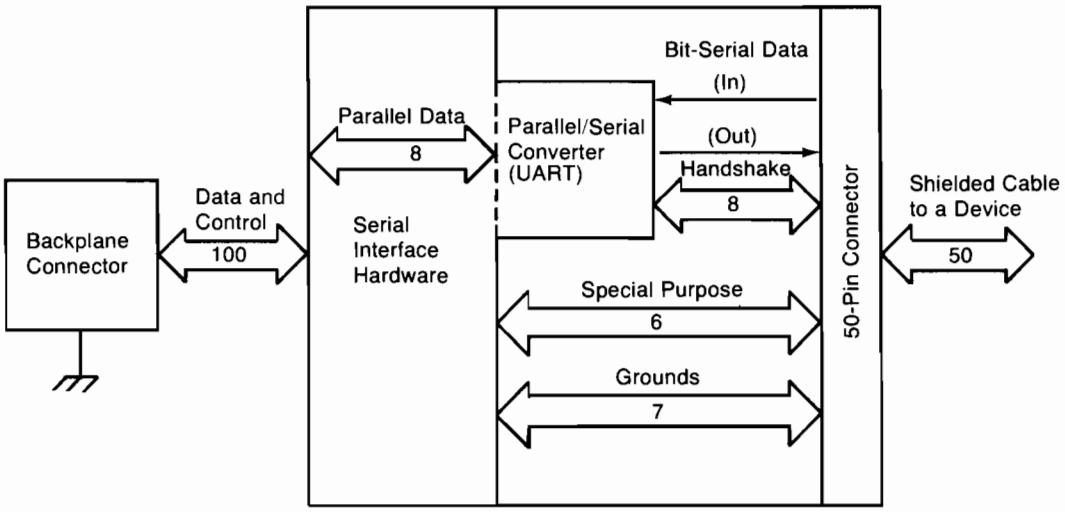
Block Diagram of the HP-IB Interface

The HP-IB interface fulfills all four compatibility requirements (hardware, electrical, data, and timing) with no additional modification. Just about all you need to do is connect the interface cable to the desired HP-IB device and begin programming. All resources connected to the computer through the HP-IB interface must adhere to this IEEE standard.

The "bus" is somewhat of an independent entity; it is a communication arbitrator that provides an organized protocol for communications between several devices. The bus can be configured in several ways. The devices on the bus can be configured to act as senders or receivers of data and control messages, depending on their capabilities.

The Serial Interface

The serial interface changes 8-bit parallel data into bit-serial information and transmits the data through a two-wire (usually shielded) cable; data is received in this serial format and is converted back to parallel data. This use of two wires makes it more economical to transmit data over long distances than to use 8 individual lines.

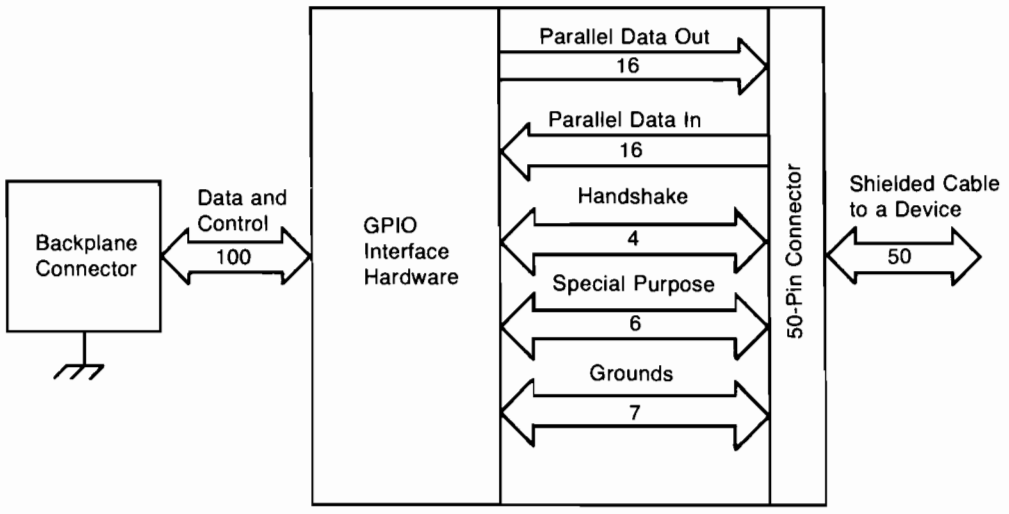


Block Diagram of the Serial Interface

Data is transmitted at several programmable rates using either a simple data handshake or no handshake at all.

The GPIO Interface

This interface provides the most flexibility of the three interfaces. It consists of 16 output-data lines, 16 input-data lines, two handshake lines, and other assorted control lines. Data is transmitted using several types of programmable handshake conventions and logic sense.



Block Diagram of the GPIO Interface

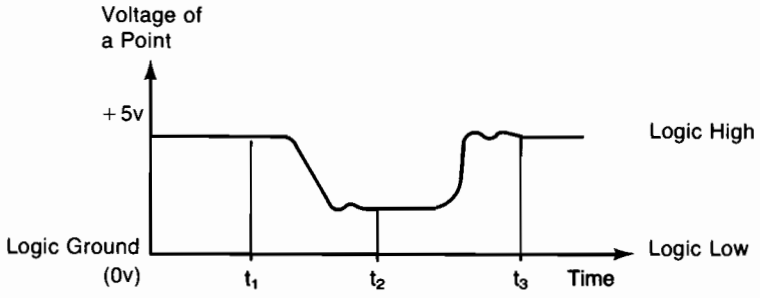
Much of the flexibility of this interface lies in the fact that you have almost direct access to the internal data bus for outputting and entering data.

Data Representations

As long as data is only being used internally, it really makes little difference how it is represented; the computer always understands its own representations. However, when data is to be moved to or from an external resource, the data representation is of paramount importance.

Bits and Bytes

Computer memory is no more than a large collection of individual bits (**bi**nary digi**ts**), each of which can take on one of two logic levels (high or low). Depending on how the computer interprets these bits, they may mean on or not on (off), true or not true (false), one or zero, busy or not busy, or any other bi-state condition. These logic levels are actually voltage levels of hardware locations within the computer. The following diagram shows the voltage of a point versus time and relates the logic levels to voltage levels.



Voltage and Positive-True Logic

In some cases, you want to determine the state of an individual bit (of a variable in computer memory, for instance). The logical binary functions (BIT_SET, BINCMP, BINIOR, BINEOR, and BINAND) provide access to the individual bits of data.

In most cases, these individual bits are not very useful by themselves, so the computer groups them into multiple-bit entities for the purpose of representing more complex data. Thus, all data in computer memory are somehow represented with binary numbers.

The computer's hardware can access groups of 16 bits at one time through the internal data bus; this size group is known as a **word**. With this size of bit group, 65536 (= 2 ↑ 16) different bit patterns can be produced. The computer can also use groups of eight bits at a time; this size group is known as a **byte**. With this smaller size of bit group, 256 (= 2 ↑ 8) different patterns can be produced. How the computer and its resources interpret these combinations of ones and zeros is very important and gives the computer all of its utility.

The computer is also capable of logically handling 32 bits; this size group is known as a long word and is the Pascal INTEGER type.

Representing Numbers

The following binary weighting scheme is often used to represent numbers with a single data byte. Only the non-negative integers 0 through 255 can be represented with this particular scheme.

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	0	1	0	1	1	0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Notice that the value of a 1 in each bit position is equal to the power of two of that position. For example, a 1 in the 0th bit position has a value of 1 (=2 ↑ 0), a 1 in the 1st position has a value of 2 (=2 ↑ 1), and so forth. The number that the byte represents is then the total of all the individual bit's values.

Determining the Number Represented

```
0 * 2↑0 =   0
1 * 2↑1 =   2
1 * 2↑2 =   4      Number represented =
0 * 2↑3 =   0
1 * 2↑4 =  16      2 + 4 + 16 + 128 = 150
0 * 2↑5 =   0
0 * 2↑6 =   0
1 * 2↑7 = 128
```

The preceding representation is used by the "ORD" function when it interprets a byte of data. The next section explains why the character "A" can be represented by a single byte.

```
PROGRAM example(input,output);
VAR number : INTEGER;
BEGIN
  number := ORD('A');
  WRITELN(' Number = ',number);
END.
```

Printed Result

```
Number = 65
```

Representing Characters

Data stored for humans is often alphanumeric-type data. Since less than 256 characters are commonly used for general communication, a single data byte can be used to represent a character. The most widely used character set is defined by the ASCII standard[1]. This standard defines the correspondence between characters and bit patterns of individual bytes. Since this standard only defines 128 patterns (bit 7 = 0), 128 additional characters are defined by the 9826 (bit 7 = 1). The entire set of the 256 characters on the 9826 is hereafter called the "extended ASCII" character set.

When the CHR function is used to interpret a byte of data, its argument must be specified by its binary-weighted value. The single (extended ASCII) character returned corresponds to the bit pattern of the function's argument.

```
PROGRAM example(input,output);
VAR number : INTEGER;
BEGIN
  number := 65;
  WRITELN(' Character is ',chr(number));
END.
```

Printed Result

```
Character is A
```

Representing Signed Integers

There are two ways that the computer represents signed integers. The first uses a binary weighting scheme similar to that used by the ORD function. The second uses ASCII characters to represent the integer in its decimal form.

Internal Representation of Integers

Bits of computer memory are also used to represent signed (positive and negative) integers. Since the range allowed by eight bits is only 256 integers, a double word (four bytes) is used to represent integers. With this size of bit group, 4 294 967 296 (= 2 ↑ 32) unique integers can be represented.

The range of integers that can be represented by 32 bits can arbitrarily begin at any point on the number line. In the 9826, this range of integers has been chosen for maximum utility; it has been divided as symmetrically as possible about zero, with one of the bits used to indicate the sign of the integer.

1 ASCII stands for "American Standard Code for Information Interchange". See the Appendix for the complete table.

With this "2's complement" notation, the most significant bit (bit 31) is used as a sign bit. A sign bit of 0 indicates positive numbers and a sign bit of 1 indicates negatives. You still have the full range of numbers to work with, but the range of absolute magnitudes is divided in half (−2 147 483 648 through 2 147 483 647). The following 32-bit integers are represented using this 2's-complement format.

Binary representation	Decimal equivalent
1111 1111 1111 1111 1111 1111 1111 1111	−1
0000 0000 0000 0000 0000 0000 0000 0001	1
1111 1111 1111 1111 1111 1111 0000 0001	−255
0000 0000 0000 0000 0000 0000 1111 1111	255

sign bit — 2 ↑ 30
2 ↑ 8 — 2 ↑ 7
2 ↑ 0

The representation of a positive integer is generated according to place value, just as when bytes are interpreted as numbers. To generate a negative number's representation, first derive the positive number's representation. Complement (change the ones to zeros and the zeros to ones) all bits, and then to this result add 1. The final result is the two's-complement representation of the negative integer. This notation is very convenient to use when performing math operations. Let's look at a simple addition of 2 two's-complement integers.

Example: 3 + (−3) = ?

First, +3 is represented as:	0000 0000 0000 0000 0000 0000 0000 0011
Now generate −3's representation:	
first complement +3,	1111 1111 1111 1111 1111 1111 1111 1100
then add 1	+ 0000 0000 0000 0000 0000 0000 0000 0001
−3's representation:	1111 1111 1111 1111 1111 1111 1111 1101
Now add the two numbers:	1111 1111 1111 1111 1111 1111 1111 1101
	+ 0000 0000 0000 0000 0000 0000 0000 0011
	1← 1← carry on
final carry not used	0000 0000 0000 0000 0000 0000 0000 0000 all places

ASCII Representation of Integers

ASCII digits are often used to represent integers. In this representation scheme, the decimal (rather than binary) value of the integer is formed by using the ASCII digits 0 through 9 {CHR(48) through CHR(57), respectively}. An example is shown below.

Example

The decimal representation of the binary value "1000 0000" is 128. The ASCII-decimal representation consists of the following three characters.

Character	1	2	8
Decimal value of character	49	50	56
Binary value of character	00110001	00110010	00111000

Representing Real Numbers

Real numbers, like signed integers, can be represented in one of two ways with the computers. They are represented in a special binary mantissa-exponent notation within the computers for numerical calculations. During output and enter operations, they can also be represented with ASCII-decimal digits.

Internal Representation of Real Numbers

Real numbers are represented internally by using a special binary notation[1]. With this method, all numbers of the REAL data type are represented by eight bytes: 52 bits of mantissa magnitude, 1 bit for mantissa sign, and 11 bits of exponent. The following equation and diagram illustrate the notation; the number represented is 1/3.

Byte	1	2	3	4	...	8
Decimal value of character	63	213	85	85	...	85
Binary value of characters	00111111	11010101	01010101	01010101	...	01010101

mantissa sign exponent mantissa

1 The internal representation used for real numbers is the IEEE standard 64-bit floating-point notation.

ASCII Representation of Real Numbers

The ASCII representation of real numbers is very similar to the ASCII representation of integers. Sign, radix, and exponent information are included with ASCII-decimal digits to form these number representations. The following example shows the ASCII representation of 1/3. Even though, in this case, 18 characters are required to get the same accuracy as the eight-byte internal representation shown above, not all real numbers represented with this method require this many characters.

ASCII characters	0		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Decimal value of characters	48	46	51	51	51	51	51	51	51	51	51	51	51	51	51	51	51	51

Chapter 5
The I/O Procedure Library

Introduction

This chapter presents an introduction to the I/O Procedure Library. This discussion includes the organization of the library, major capabilities, and an introduction into the use of the library. The last sections of this chapter contain a list of module capabilities. It is recommended that you scan these sections to familiarize yourself with what features are available in the I/O Library.

Pascal I/O

The Pascal language has been well known for some time as a good high-level langauge with modularity and transportability features. It has not had good I/O capabilities, particularly device I/O. The Pascal language on the HP 9826 and 9836 computers still does not have I/O as a fundamental part of the language.

Rather than adding specific built-in language features to support I/O, graphics, and other useful extensions, HP Standard Pascal has a general extension mechanism called modules. A module is very similar to a Pascal PROGRAM in that it can contain VARiables, CONSTants, PROCEDUREs, and FUNCTIONs.

Various portions of a module can be EXPORTed for anyone to use. The Pascal I/O Procedure Library is a collection of several modules placed in the LIBRARY file. When you want to use the capabilities of the I/O library you must tell the compiler which modules you want from the I/O library. This is done with the IMPORT feature.

An example of using the I/O library follows. You want to write a program that reads a string from a device and then writes a string to the same device. The read and write string procedures are both in the I/O module called GENERAL_2. So the program might look like:

```
PROGRAM test ( INPUT , OUTPUT );
IMPORT GENERAL_2;                { tell the compiler which module }
VAR    str : STRING[255];
BEGIN
  READSTRING(724,str);           { read  str with CR/LF termination }
  WRITESTRINGLN(724,str);        { write str with CR/LF termination }
END.
```

I/O Library Organization

Each of the I/O Library modules contains related features and capabilities. I/O consists of general capabilities that are valid for all interfaces and devices and of specific capabilities that are valid only for a specific interface or type of interface. Reading a character is an example of a general capability. Checking for ACTIVE CONTROL is an HP-IB specific operation.

The I/O Library is divided into groups: general and interface specific. The interfaces currently supported in the I/O Library consist of HP-IB, Serial, and Parallel (GPIO) interfaces. In the implementation of the I/O Library, all the necessary Parallel capabilities are handled in the general capabilities group. So, the I/O Library consists of three groups:

- GENERAL
- HPIB
- SERIAL

Each of these groups consists of several modules. The last section in this chapter contains a list of the procedures and functions in each of the modules in the I/O Library.

GENERAL

The GENERAL group contains the common operations used by all interfaces. This group consists of the following modules:

Module	Capability	Example
GENERAL_0	machine and hardware dependent status and control	hardware register access
GENERAL_1	character I/O	input a character
GENERAL_2	string and numeric I/O	input a real number
GENERAL_3	error messages	
GENERAL_4	transfers and buffers	output data via DMA
IODECLARATIONS	common constants, types, variables	what type of card is the interface at interface select code 7
IOCOMASM	binary operations	binary AND of two integers

HPIB

The HPIB group contains routines that are useful for the built-in and optional HP-IB interfaces.

Module	Capability	Example
HPIB_0	access to HP-IB interface bus lines	clear the ATN line
HPIB_1	low level bus control	send an ATN bus command
HPIB_2	HP-IB messages	send selective device clear
HPIB_3	high level bus status and control	request bus service

SERIAL

The SERIAL group contains the capabilities specific to serial interfaces. Currently, the HP 98626 and 98628 are supported.

Module	Capability	Example
SERIAL_0	access to serial interface lines	set Clear To Send
SERIAL_3	high level serial control	set baud rate to 2400

Each module is a separate entity in the Pascal system. Being separate, only those modules imported from the system library are used in the running of an application program. This partitioning of the library minimizes the size of the program. The Pascal system, in normal programming, will load and link all the modules that you have imported. You only need to explicitly import the appropriate modules and use their procedures and functions.

I/O Library Initialization

The I/O Library provides a setup procedure, IOINITIALIZE, and a clean up procedure, IOUNINITIALIZE. Both procedures operate in a very similar manner. They perform the following operations:

- Reset all interfaces.
- Stop all transfers.
- Release all I/O resources (such as DMA channels).

A well written Pascal program that uses the I/O Library will include these procedures. These procedures are in the GENERAL_1 module. The example program from the previous section rewritten would look like:

```
PROGRAM test ( INPUT , OUTPUT );
IMPORT GENERAL_1,
       GENERAL_2;                { tell the compiler which modules  }
VAR    str : STRING[255];
BEGIN
  IOINTIALIZE;                   { set up the I/O system            }
  READSTRING(724,str);           { read  str with CR/LF termination }
  WRITESTRINGLN(724,str);        { write str with CR/LF termination }
  IOUNINITIALIZE;                { clean up the I/O system          }
END.
```

The I/O system is used by the rest of the Pascal system for I/O operations. Because of this use, IOINITIALIZE is called by the system when power is first applied to the computer. Also, because I/O errors can occur during normal operation, the STOP and CLR I/O keys call IOUNINITIALIZE to clean up the I/O system state. This information leads to the fact that it is, in many instances, unnecessary to call IOINITIALIZE and IOUNINITIALIZE. It is, however, strongly recommended that you use these procedures. The use of the set-up and clean-up procedures will make your programs more resistant to hardware and firmware problems and to programming errors in software.

GENERAL Modules

GENERAL modules contain the capabilities that are useful for all interfaces. For syntax and semantics information refer to the reference section in the back of this manual.

MODULE iocomasm	
FUNCTION bit_set	Is a bit set in a 32-bit integer?
FUNCTION binand	Logical AND of two 32-bit integers.
FUNCTION binior	Logical OR of two 32-bit integers.
FUNCTION bineor	Exclusive OR of two 32-bit integers.
FUNCTION bincmp	Logical complement of a 32-bit integer.
MODULE general_0	
FUNCTION ioread_word	Read a 16-bit interface register.
PROCEDURE iowrite_word	Write a 16-bit interface register.
FUNCTION ioread_byte	Read an 8-bit interface register.
PROCEDURE iowrite_byte	Write an 8-bit interface register.
FUNCTION iostatus	Read the firmware interface register.
PROCEDURE iocontrol	Write the firmware interface register.
MODULE general_1	
PROCEDURE ioinitialize	Reset the entire I/O system.
PROCEDURE iouninitialize	Reset the entire I/O system.
PROCEDURE ioreset	Reset a single interface card.
PROCEDURE readchar	Read a character from an interface.
PROCEDURE writechar	Write a character to an interface.
PROCEDURE readword	Read a 16-bit word from an interface.
PROCEDURE writeword	Write a 16-bit word to an interface.
PROCEDURE set_timeout	Set up an interface timeout value.
MODULE general_2	
PROCEDURE readnumber	Read a real number.
PROCEDURE writenumber	Write a real number.
PROCEDURE readstring	Read a string.
PROCEDURE readstring_until	Read a string until a character match.
PROCEDURE writestring	Write a string.
PROCEDURE readnumberln	Read a real number until a LF occurs.
PROCEDURE writenumberln	Write a real number with a CR/LF.
PROCEDURE writestringln	Write a string with a CR/LF.
PROCEDURE readuntil	Read until a character match.
PROCEDURE skipfor	Skip over a number of characters.
MODULE general_3	
FUNCTION ioerror_message	What is the error message for a specific I/O error?
MODULE general_4	
PROCEDURE abort_transfer	Stop a transfer.
PROCEDURE transfer	Transfer a block of data as bytes.
PROCEDURE transfer_word	Transfer a block of data as words.
PROCEDURE transfer_until	Transfer in until a match character.
PROCEDURE transfer_end	Transfer using a card condition.
PROCEDURE iobuffer	Create a transfer buffer.
PROCEDURE buffer_reset	Reset the buffer space.
FUNCTION buffer_space	How much space is left in the buffer.
FUNCTION buffer_data	How much data is left in the buffer.
PROCEDURE readbuffer	Read a character from a buffer.
PROCEDURE writebuffer	Write a character to a buffer.
PROCEDURE readbuffer_string	Read a string from a buffer.
PROCEDURE writebuffer_string	Write a string to a buffer.
FUNCTION buffer_active	Is there a transfer active on the buffer?
FUNCTION isc_active	Is there a transfer active on the interface?

HPIB Modules

HPIB modules contain routines that are useful for the built-in and optional HP-IB interfaces. For syntax and semantics information refer to the reference section in the back of this manual.

MODULE hpib_0	
PROCEDURE set_hpib	Set an HP-IB hardware line.
PROCEDURE clear_hpib	Clear an HP-IB hardware line.
FUNCTION hpib_line	Is an HP-IB hardware line set?
MODULE hpib_1	
PROCEDURE send_command	Send an ATN command.
FUNCTION my_address	What is my bus address?
FUNCTION active_controller	Am I active controller?
FUNCTION system_controller	Am I system controller?
FUNCTION end_set	Was EOI received with the last byte?
MODULE hpib_2	
PROCEDURE abort_hpib	Stop all bus activity.
PROCEDURE clear	Send clear command to a device.
PROCEDURE listen	Send listen command to a device.
PROCEDURE local	Send local command to a device.
PROCEDURE local_lockout	Send lockout command to all devices.
PROCEDURE pass_control	Pass active control to a device.
PROCEDURE ppoll_configure	Configure PPOLL response of a device.
PROCEDURE ppoll_unconfigure	Remove PPOLL response of a device.
PROCEDURE remote	Send remote command to a device.
PROCEDURE secondary	Send a secondary command.
PROCEDURE talk	Send talk command to a device.
PROCEDURE trigger	Send trigger command to a device.
PROCEDURE unlisten	Send unlisten command to all devices.
PROCEDURE untalk	Send untalk command to all devices.
MODULE hpib_3	
FUNCTION requested	Is SRQ asserted?
FUNCTION ppoll	What is the bus parallel poll byte?
FUNCTION spoll	What is the device serial poll byte?
PROCEDURE request_service	Request bus service (via SRQ).
FUNCTION listener	Am I a listener?
FUNCTION talker	Am I a talker?
FUNCTION remoted	Is REN being asserted?
FUNCTION locked_out	Am I in the local lockout state?

SERIAL Modules

SERIAL modules contain the capabilities specific to serial interfaces. Currently, the HP 98626 Serial 98628 Datacomm cards are supported. For syntax and semantics information refer to the reference section in the back of this manual.

MODULE serial_0

PROCEDURE set_serial	Set a serial line.
PROCEDURE clear_serial	Clear a serial line.
FUNCTION serial_line	Is a serial line set?

MODULE serial_3

PROCEDURE set_baud_rate	Set the interface baud rate.
PROCEDURE set_stop_bits	Set the interface number of stop bits.
PROCEDURE set_char_length	Set the interface character length.
PROCEDURE set_parity	Set the interface parity.
PROCEDURE send_break	Send a serial BREAK.
PROCEDURE abort_serial	Stop all serial activity.

IODECLARATIONS Module

The rest of the I/O Library consists of modules that contain procedures and functions. The IODECLARATIONS module is a module of constants, types, and variables. This module is used by the rest of the I/O Library for range checking, common variables, and I/O system tables. IODECLARATIONS is also of use to you, the programmer, for various reasons. This section will not fully discuss the IODECLARATIONS module. It will only discuss the points of general interest.

The useful information in IODECLARATIONS relates to interface information. Typical questions about interfaces include:

- What is the range of interfaces?
- Is there an interface on interface select code 12?
- Is the interface on interface select code 15 a serial interface?
- Is the interface on interface select code 15 a 98626 serial interface or a 98628 serial interface?

The descriptions that follow will show the actual Pascal code used to define the various constants, types and variables.

Range of Interface Select Codes and Devices

This range is supported by several constants and types. The I/O Library supports various select codes, as described in the next chapter. The interface select code range is from 0 through 31. There are two constants that define this range:

```
CONST   IOMINISC    = 0 ;
        IOMAXISC    = 31;
```

In addition to defining the upper and lower limits of select codes there are type definitions that support interface select code and device variables. These type definitions are:

```
TYPE    TYPE_ISC        = IOMINISC..IOMAXISC ;
        TYPE_DEVICE     = IOMINISC..IOMAXISC*100+99;
```

These type definitions are used in the I/O Library for interface select code and device parameters. With the compiler option $RANGE ON$, which is the default, the compiler will emit a range check for your parameters. So, if you tried to use an interface select code of 45, the program would generate an error. You can use the type definitions for interface select code and device variables, if you desire. It is also possible to use integer variables and other integer subranges for interface select code and device variables.

Information about the Interface

There is a table defined in the IODECLARATIONS module that contains common information about all interface cards in the computer. This table is called ISC_TABLE and is an array of structured elements, a compound data type. The definition of this table is:

```
VAR     ISC_TABLE       : PACKED ARRAY [TYPE_ISC]
                          OF isc_table_type;
```

The compound data type ISC_TABLE_TYPE contains several pieces of information. The definition of this type is:

```
TYPE    isc_table_type  = RECORD
                  io_drv_ptr: ^driver;        { ptr to drivers }
                  io_tmp_ptr: ^memory;        { ptr to R/W     }
                  CARD_TYPE : -32768..32767;
                  user_time : INTEGER;        { for timeout    }
                  CARD_ID   : -32768..32767;
                  card_ptr  : ^card;          { card addr      }
                END;
```

The table contains pointers to the actual drivers, driver read/write memory space, user specified timeout value and a pointer to the physical address of the interface card in the computer's memory. The table also contains the type of card and card id information. You should only need to examine the card type and card id.

Note

All of this information is for system use. Do not modify any table entries.

The following program lists the type of card and card id for all interfaces.

```
PROGRAM list_cards ( INPUT , OUTPUT );
IMPORT IODECLARATIONS;
VAR  isc : TYPE_ISC;
BEGIN
  FOR isc := IOMINISC TO IOMAXISC DO
    WRITELN('card ',          isc:2,
            ' is of type ',   ISC_TABLE[isc].CARD_TYPE:4,
            ' with an id of ',ISC_TABLE[isc].CARD_ID:4);
END.
```

This program is not useful because the values for card type and id are integers and you do not know what each value means. The IODECLARATIONS module has a series of pre-defined constants for the card type and id.

The card type field contains information about the generic card type—whether the card is Serial, HP-IB, etc. The constants are as follows:

```
CONST NO_CARD          = 0 ;
      OTHER_CARD       = 1 ;
      SYSTEM_CARD      = 2 ;
      HPIB_CARD        = 3 ;
      GPIO_CARD        = 4 ;
      SERIAL_CARD      = 5 ;
      GRAPHICS_CARD    = 6 ;
      SRM_CARD         = 7 ;
```

The card id contains hardware specific information. For example, the id will inform you whether an HPIB_CARD is the internal interface or an optional 98624 plug in. This should only be necessary if you are doing low level operations to the interfaces.

Note

The appearance of a card id in the following list **does not** imply Pascal support for the specified interface. The cards are mentioned because they may be supported by other languages which run on this machine.

The constants are defined as follows:

```
CONST HP98626_DSNDL   = -7 ;
      HP98629         = -6 ;
      HP_DATACOMM     = -5;
      HP98620         = -4;
      INTERNAL_KBD    = -3;
      INTERNAL_CRT    = -2;
      INTERNAL_HPIB   = -1;
      NO_ID           =  0;          { no card       }
      HP98624         =  1;          { HP-IB         }
      HP98626         =  2;          { serial        }
      HP98622         =  3;          { gpio          }
      HP98623         =  4;          { bcd           }
      HP98625         =  8;          { disk          }
      HP98628_ASYNC   = 20;          { serial        }
      HP98627         = 28;          { graphics      }
```

A program to determine card type and id is shown below.

```
PROGRAM list_cards ( INPUT , OUTPUT );
IMPORT IODECLARATIONS;
VAR  isc : TYPE_ISC;
BEGIN
  FOR isc := IOMINISC TO IOMAXISC DO BEGIN

    IF ISC_TABLE[isc].CARD_TYPE > SYSTEM_CARD
      THEN BEGIN
        WRITE('card ',isc:2,' is of type : ');
        CASE ISC_TABLE[isc].CARD_TYPE OF

          HPIB_CARD:      WRITE(' HPIB      ');
          GPIO_CARD:      WRITE(' GPIO      ');
          SERIAL_CARD:    WRITE(' SERIAL    ');
          GRAPHICS_CARD:  WRITE(' GRAPHICS  ');

          OTHERWISE       WRITE(' other     ');

        END; { of CASE         }
      END;   { of IF  NO_CARD }

    IF ISC_TABLE[isc].CARD_TYPE > SYSTEM_CARD
      THEN BEGIN
        WRITE(' and of id : ');
        CASE ISC_TABLE[isc].CARD_ID OF

          HP_DATACOMM:    WRITE(' HP 98628 - NON ASYNC ');
          INTERNAL_HPIB:  WRITE(' built in             ');
          HP98624:        WRITE(' HP 98624             ');
          HP98626:        WRITE(' HP 98626             ');
          HP98622:        WRITE(' HP 98622             ');
          HP98623:        WRITE(' HP 98623             ');
          HP98625:        WRITE(' HP 98625             ');
          HP98628_ASYNC:  WRITE(' HP 98628 - ASYNC     ');
          HP98627:        WRITE(' HP 98627             ');

          OTHERWISE       WRITE(' other     ');

        END; { of CASE          }
        WRITELN('');
      END;   { of IF  NO_CARD  }

  END;       { of FOR DO BEGIN }
END.
```

Other Types

In addition to the previously specified information there are some pre-defined types used throughout the I/O Library. These type definitions are:

```
IO_BIT          = 0..15 ;
IO_BYTE         = 0..255 ;
IO_WORD         = -32768..32767 ;
IO_STRING       = STRING[255];
```

Notes

Chapter 6
Directing Data Flow

Introduction

This chapter describes how to specify which computer resource is to send data to the computer or receive data from the computer. There are three main resources for the source and destination of data:

- Internal devices
- External devices
- Mass storage files

The I/O Library is used for accessing internal and external devices and is discussed here. The Pascal system has other methods for accessing mass storage files and these commands are covered in the Pascal System User's Manual.

Specifying a Resource

The procedures and functions that perform I/O have a device selector parameter as a part of the parameter list. This parameter has two forms: a simple device selector and an addressed device selector.

Simple Device Selectors

Devices include the built-in CRT and keyboard, external printers and instruments, and all other physical entities that can be connected to the computer through an interface. Thus, each device connected to the computer can be accessed through its interface. Each interface has a unique number by which it is identified, known as its interface select code. The internal devices are accessed with the following, permanently assigned interface select codes.

Device	Select Code
CRT Display	1
Keyboard	2
Built-in HP-IB	7

Optional interfaces all have switch-settable select codes. These interfaces cannot use select codes 0 through 7; the valid range is 8 through 31. The following settings on optional interfaces have been made at the factory but can be changed to any other unique select code. See the interface's installation manual for further instructions.

Device	Select Code
98624A HP-IB	8
98622A GPIO	12
98628A Datacomm	20
98625A Disc	14

An example program using interface select codes is shown below:

```
PROGRAM selectcode ( INPUT , OUTPUT );
IMPORT  GENERAL_2;
VAR     str : STRING[255];
BEGIN
  WRITESTRING(1,'type something - terminated by the ENTER key');
  READSTRING_UNTIL(CHR(13),2,str);
  WRITESTRING(12,'message from keyboard - ');
  WRITESTRINGLN(12,str);
END.
```

Addressed Device Selectors

Each device on an HP-IB interface has an address by which it is uniquely identified. The addressed device selector is a combination of the interface select code and the device's bus address. This combination is:

interface select code * 100 + device bus address = addressed device selector

A printer with a bus address of 1 on the internal HP-IB interface (which is an interface select code of 7) would be accessed with a device selector of 701.

An example program using an addressed device selector is shown below:

```
PROGRAM device ( INPUT , OUTPUT );
IMPORT  GENERAL_2;
VAR     num : REAL;
BEGIN
  READNUMBERLN(724,num);
  WRITESTRING(701,'reading from voltmeter - ');
  WRITENUMBERLN(701,num);
END.
```

Chapter 7
Outputting Data

Introduction

The preceding chapter described how to identify a specific device as the destination of data in a WRITESTRING procedure. Even though a few examples were shown, the details of how the data is sent was not discussed. This chapter describes the topic of outputting data to devices.

There are two general classes of output operations. The first type, known as "free field" output, uses the computer's default data representation. The second class provides precise control over each character to be sent and is called "formatted" output.

The I/O Library is a separate set of procedures and functions. As such, it does not have variable length or variable type parameter lists. In Pascal there is a normal "print" facility called WRITE and WRITELN (for write line) that can have a variable list. Some examples are:

```
WRITELN('hello there');
WRITELN('the value received was ',i);
WRITE(i,' times ',j,' is equal to ',i*j);
WRITE(client.name,' has ',client.eyecolor,' eyes ');
```

Note that there are no requirements for what types of constants, variables, or expressions are allowed in a list, nor are there any requirements for their order in a list.

Because of this restriction on the variability of lists, the I/O Library only normally supports a small set of types. These types are:

- Real expressions
- Strings (up to 255 characters)
- Characters (8 bits)
- Words (16 bits)

The procedures that handle these types will only handle one of the type. These operations can be used in a series to get the effect of a list.

Free Field Output

As mentioned in the previous section, there are four main types supported directly by the I/O Library output facility. These are:

- Real Expressions
- String Expressions
- Characters
- Words

Real Expressions

There are two output procedures for real expressions: WRITENUMBER and WRITENUMBERLN. Both operate in an identical fashion except that WRITENUMBERLN appends a carriage return and line feed to the characters sent to the device. The form of these procedures is:

```
WRITENUMBER    (  device_specifier , numeric_expression ) ;
WRITENUMBERLN  (  device_specifier , numeric_expression ) ;
```

Both procedures are in the I/O Library module GENERAL_2. The device specifier can be a simple interface select code or a device specifier. The numeric expression can be any valid expression including simple real, integer, or integer subrange variables, numeric constants, and numeric expressions. An example program follows:

```
PROGRAM realexpression (INPUT,OUTPUT);
IMPORT        IODECLARATIONS,
              GENERAL_2;
VAR a       : REAL;
    i       : INTEGER;
    device  : TYPE_DEVICE;
BEGIN
  device:=701;
  i:=12;
  a:=12.34;
  WRITENUMBERLN(device,i);
  WRITENUMBERLN(device,a);
  WRITENUMBERLN(device,1234);
  WRITENUMBERLN(device,a+1234);
  WRITENUMBERLN(device,i+12);
END.
```

This program will produce the following output:

```
1.20000E+001
1.23400E+001
1.23400E+003
1.24634E+003
2.40000E+001
```

The example program did not use WRITENUMBER. This is because there are no additional characters sent with the ASCII character sequence. Two numbers sent with two consecutive WRITENUMBERs might look like:

```
1.23456E+1239.87654E-321
```

Notice that there is no separator. The examples toward the end of this section will show examples of WRITENUMBER. Be sure that you remember that the real number can be preceded by a minus sign.

String Expressions

There are two output procedures for string expressions: WRITESTRING and WRITESTRINGLN. Both operate in an identical fashion except that WRITESTRINGLN appends a carriage return and line feed to the characters sent to the device. The form of these procedures is:

```
WRITESTRING    ( device_specifier , string_expression ) ;
WRITESTRINGLN  ( device_specifier , string_expression ) ;
```

Both procedures are in the I/O Library module GENERAL_2. The device specifier can be a simple interface select code or a device specifier. The string expression can be any valid expression including simple string variables , string constants, and string expressions. An example program follows:

```
PROGRAM strings (INPUT,OUTPUT);
IMPORT       IODECLARATIONS,
             GENERAL_2;
VAR s      : STRING[255];
    t      : STRING[32];
    device : TYPE_DEVICE;
BEGIN
  device:=701;
  s:='first string';
  t:='second string';
  WRITESTRING  (device,s);
  WRITESTRINGLN(device,t);
  WRITESTRING  (device,'this is a string constant and ');
  WRITESTRINGLN(device,'this is the '+s);
  WRITESTRINGLN(device,'both '+s+' and the '+t);
END.
```

This program will produce the following output:

```
first stringsecond string
this is a string constant and this is the first string
both first string and the second string
```

Characters

There is a single output procedure for single characters: WRITECHAR. The form of this procedures is:

```
WRITECHAR (interface_select_code, character_expression);
```

The procedure is in the I/O Library module GENERAL_1. The interface select code cannot be a device specifier (like 701). Refer to the HP-IB section regarding bus addressing. The character expression can be a character variable, character constant, or character expression. An example program follows:

```
PROGRAM characters (INPUT,OUTPUT);
IMPORT       IODECLARATIONS,
             GENERAL_1,
             GENERAL_2;
VAR c       : CHAR;
    i,j     : INTEGER;
    device  : TYPE_DEVICE;
    isc     : TYPE_ISC;
BEGIN
  isc:=7;
  device:=701;
  WRITESTRING(device,'some characters <');
  WRITECHAR(isc,'x');
  c:='y';
  WRITECHAR(isc,c);
  j:=ORD('z');
  WRITECHAR(isc,chr(j));
  FOR i:=65 TO 90 DO WRITECHAR(isc,chr(i));
  WRITESTRINGLN(isc,'>');
END.
```

This program will produce the following output:

```
some characters <xyzABCDEFGHIJKLMNOPQRSTUVWXYZ>
```

Words

There is a single output procedure for 16 bit words. It is WRITEWORD. The form of this procedures is:

```
WRITEWORD (interface_select_code, word_expression);
```

The procedure is in the I/O Library module GENERAL_1. The interface select code cannot be a device specifier (like 701). Refer to the HP-IB section regarding bus addressing. The word expression can be a word, integer, or integer subrange variable, integer constant, or integer expression. The evaluated value must be in the range of −32768 to 32767.

The procedure has two different behaviors, depending on what type of interface it is used with. When used with a GPIO interface (HP 98622), this procedure will send a single 16 bit quantity over the 16 data lines on the interface. This procedure will send two consecutive bytes for all other interface types — most significant byte first, least significant byte last. An example program for an HP-IB interface follows:

```
PROGRAM words (INPUT,OUTPUT);
IMPORT        IODECLARATIONS,
              GENERAL_1,
              GENERAL_2;
TYPE short = -32768..32767;
VAR c       : CHAR;
    i,j     : INTEGER;
    x       : IO_WORD;
    y       : short;
    device  : TYPE_DEVICE;
    isc     : TYPE_ISC;
BEGIN
  isc:=7;
  device:=701;
  WRITESTRING(device,'some characters <');
  x:=65*256+66;
  WRITEWORD(isc,x);
  WRITEWORD(isc,67*256+68);
  j:=69*256+70;
  WRITEWORD(isc,j);
  j:=ORD('z');
  FOR i:=65 TO 75 DO WRITEWORD(isc,j*256+i);
  WRITESTRINGLN(isc,'>');
END.
```

This program will produce the following output:

```
some characters <ABCDEFzAzBzCzDzEzFzGzHzIzJzK>
```

The following program is an example of how to use the "free field" procedures together to get effect of a full parameter list:

```
PROGRAM strings (INPUT,OUTPUT);
IMPORT        IODECLARATIONS,
              GENERAL_1,
              GENERAL_2;
VAR s,t     : STRING[255];
    x       : REAL;
    device  : TYPE_DEVICE;
    isc     : TYPE_ISC;
BEGIN
  device:=701;
  isc   :=7;
  s:='Range1;Trigger1;Number';
  x:=100;
  t:='Store';
  WRITESTRING  (device,s);
  WRITENUMBER  (isc   ,x);
  WRITESTRING  (isc   ,t);
  WRITECHAR    (isc   ,chr(10));
END.
```

This program will produce the following output sequence:

```
Range1;Trigger1;Number1.00000E+002Store
```

Formatted Output

The previous "free field" procedures are adequate for a large number of applications. There are, however, a large number of applications that need the "formatted" output capability. The I/O Library does not directly provide this capability. Formatted output is achieved with the use of the built in procedure STRWRITE.

STRWRITE

The STRWRITE procedure is a version of the standard Pascal procedure WRITE. The difference is that STRWRITE sends the character stream to a string variable, as opposed to an output file. The form of STRWRITE is as follows:

```
STRWRITE (string_variable, starting_char, next_char_var,...output list... );
```

The string variable is the destination for the output operation. The starting character position is an integer expression that indicates which character in the string is the start of the output area. The next character variable will contain, after the execution of STRWRITE, the next available character in the string for a successive STRWRITE or other string operation. For additional information, refer to *The Pascal Handbook.*

The following program is an example of how to use STRWRITE to produce formatted output:

```
PROGRAM formatted (INPUT,OUTPUT);
IMPORT        IODECLARATIONS,
              GENERAL_2;
TYPE color = ( blue , brown , green , red );
VAR s,name : STRING[255];
    pos,n  : INTEGER;
    eyes   : color;
    device : TYPE_DEVICE;
BEGIN
  device:=701;

  name  :='John Smith';
  n     :=12;
  eyes  :=blue;

  STRWRITE(s,1,pos, name,' is employee number ',n:4);
  SETSTRLEN(s,pos-1);
  WRITESTRINGLN(device,s);

  STRWRITE(s,1,pos, 'and has ',eyes,' eyes ');
  SETSTRLEN(s,pos-1);
  WRITESTRINGLN(device,s);
END.
```

This program will produce the following output:

```
John Smith is employee number   12
and has BLUE eyes
```

Notes

Chapter 8
Inputting Data

Introduction

There are two general classes of input operations. The first type, known as "free field" input, uses a default interpretation of the data to be input. The second class provides precise control over each character to be received and is called "formatted" input.

The I/O Library is a separate set of procedures and functions. As such, it does not have variable length or variable type parameter lists. In PASCAL there is a normal "input" facility called READ and READLN (for read line) that can have a variable list. Some examples are:

```
READ(name); FOR i:= 1 TO 100 DO READ(mychar[i]);
READ(voltage,frequency); READLN(prompt);
```

Note that there are no requirements for what types of variables are allowed in the list, nor are there any requirements on the order of variables on the list. Because of this restriction on the variability of lists, the I/O Library only normally supports a small set of input types. These types are:

- Real variables
- Strings (up to 255 characters)
- Characters (8 bits)
- Words (16 bits)

In addition to these data types, the I/O Library supports some field skipping facilities. The procedures that handle these types and facilities will only handle one operation at a time. These operations can be used in a series to get the effect of a list.

Free Field Input

As mentioned in the previous section, there are five main types supported directly by the I/O Library input facility. These are:

- Real Variables
- String Variables
- Characters
- Words
- Field Skipping

Real Variables

There are two input procedures for real variables: READNUMBER and READNUMBERLN. Both operate in an identical fashion except that READNUMBERLN searches for a line feed termination from the device. The form of these procedures is:

```
READNUMBER (device_specifier, real_variable);
READNUMBERLN (device_specifier, real_variable);
```

Fundamental to understanding how these procedures work is the concept of termination. The READNUMBER procedures will skip over any number of non-numeric characters until a numeric character is found. Then, up to 255 numeric characters will be read in as an ASCII representation of a real number. Numeric characters are defined to be:

0	5	E
1	6	e
2	7	+
3	8	–
4	9	period
		space

When reading numbers, the terminating conditions are:

- Any non-numeric character after numeric characters have been read, or
- 255 numeric characters read.

Both procedures are in the I/O Library module GENERAL_2. The device specifier can be a simple interface select code or a device specifier. The variable must be a real variable (including a real array element). An example program follows:

```
PROGRAM realvariable (INPUT,OUTPUT);
IMPORT       IODECLARATIONS,
             GENERAL_2;
VAR a      : REAL;
BEGIN
  { input comes from keyboard }

  WRITELN('type in a real number terminated by any non-numeric');
  READNUMBER(1,a);
  WRITELN;
  WRITELN('you typed in the value ',a);
```

(Continued)

```
  WRITELN('type in a real number terminated by a control-j');
  READNUMBERLN(1,a);
  WRITELN;
  WRITELN('you typed in the value ',a);

END.
```

String Variables

There are two input procedures for string variables: READSTRING and READSTRING_UNTIL. Both operate in a similar manner except that READSTRING_UNTIL searches for a specified termination character where the READSTRING uses some default terminations.

The form of the READSTRING procedure is:

```
READSTRING (device_specifier, string_variable);
```

The READSTRING procedure will read characters into a string until one of the following termination conditions are encountered:

- A line feed is received.
- A carriage return and a line feed are received.
- The string variable is filled.

The line feed or carriage return and line feed are NOT placed in the string variable. The form of the READSTRING_UNTIL procedure is:

```
READSTRING_UNTIL   (termination_character,
                    device_specifier, string_variable);
```

The READSTRING_UNTIL procedure will read in characters into a string until one of the following termination conditions are encountered:

- The match character is received.
- The string variable is filled.

The termination character is placed into the string variable.

Both procedures are in the I/O Library module GENERAL_2. An example program follows:

```
PROGRAM stringvariable (INPUT,OUTPUT);
IMPORT        IODECLARATIONS,
              GENERAL_2;
VAR s       : STRING[255];
    t       : STRING[  8];
BEGIN
  { the keyboard is the input device }

  WRITELN('enter a string terminated with a control-J');
  READSTRING(1,s);
  WRITELN('you entered <',s,'> as your string');

  WRITELN('enter a string of 8 characters');
  READSTRING(1,t);
  WRITELN('you entered <',t,'> as your string');

  WRITELN('enter a string terminated with an ENTER ( carriage return )');
  READSTRING_UNTIL(chr(13),1,s);
  WRITELN('you entered <',s,'> as your string');

END.
```

Characters

There is a single input procedure for single characters—READCHAR. The form of this procedures is:

```
READCHAR (interface_select_code, character_variable);
```

The procedure is in the I/O Library module GENERAL_1. The interface select code cannot be a device specifier (like 701). Refer to the HP-IB section regarding bus addressing. The variable must be a character variable. An example program follows:

```
PROGRAM characters (INPUT,OUTPUT);
IMPORT        IODECLARATIONS,
              GENERAL_1;
VAR c       : CHAR;
BEGIN
  REPEAT
    READCHAR(1,c);
    WRITELN;
    WRITELN('you typed ',c,' which is character ',ORD(c):3);
  UNTIL c=CHR(13);
  WRITELN('done');
END.
```

Words

READWORD is the input procedure for 16-bit words. The form of this procedures is:

```
READWORD (interface_select_code, integer_variable);
```

The procedure is in the I/O Library module GENERAL_1. The interface select code cannot be a device specifier (like 701). Refer to the HP-IB section regarding bus addressing. The variable must be an integer variable. The returned value will be in the range of −32 768 to 32 767.

The procedure has two different behaviors, depending on what type of interface it is used with. When used with an HP 98622 GPIO interface, this procedure will read a single 16-bit quantity from the 16 data lines on the interface. This procedure will read two consecutive bytes for all other interface types – most significant byte first, least significant byte last. An example program for an HP-IB interface follows:

```
PROGRAM words (INPUT,OUTPUT);
IMPORT        IODECLARATIONS,
              GENERAL_1;
VAR x       : INTEGER;
BEGIN
  READWORD(12,x);
  WRITELN('the word received was : ',x:7);
END.
```

Skipping Data

There are applications where you want to skip over a block of data and do not wish to store the information. The I/O Library has two procedures to support skipping over data: READUNTIL and SKIPFOR.

The READUNTIL procedure skips over data until a match character is received. It is of the form:

```
READUNTIL (termination_character, device_specifier);
```

The SKIPFOR procedure skips over a specified number of characters. It is of the form:

```
SKIPFOR (skip_count, device_specifier);
```

The skip count is an integer expression. Both procedures are in I/O Library module GENERAL_2.

Formatted Input

The previous "free field" procedures are adequate for a large number of applications. There are, however, a large number of applications that need the "formatted" input capability. The I/O Library does not directly provide this capability. Formatted input is achieved with the use of the built in procedure STRREAD.

STRREAD

The STRREAD procedure is a version of the standard Pascal procedure READ. The difference is that STRREAD reads the character stream from a string variable, as opposed to an input file. The form of STRREAD is as follows:

```
STRREAD (string_variable, starting_char, next_char_var, ...input list... );
```

The string variable is the source for the input operation. The starting character position is an integer expression that indicates which character in the string is the start of the data to be read. The next character variable will contain, after the execution of STRREAD, the next available character in the string for a successive STRREAD or other string operation. For additional information, refer to *The PASCAL Handbook*.

The following program is an example of how to use STRREAD to produce formatted input.

```
PROGRAM formatted (INPUT,OUTPUT);
IMPORT        IODECLARATIONS,
              GENERAL_2;
TYPE color = ( blue , brown , green , red );
VAR s       : STRING[12];
    t       : STRING[ 8];
    pos     : INTEGER;
    eyes    : color;
BEGIN

  WRITELN('enter 8 alphabetic characters');
  WRITELN('and then type the characters BLUE');

  READSTRING(1,s);

  STRREAD(s,1,pos, t,eyes);

  WRITELN('the string is ',t,' and the eyes are ',eyes);

END.
```

Chapter 9
Registers

Introduction

There are two classes of registers in the Pascal I/O Library: firmware registers and hardware registers. The firmware registers are accessed by the IOSTATUS function and the IOCONTROL procedure. The hardware registers are accessed by the IOREAD_BYTE and IOREAD_WORD functions and the IOWRITE_BYTE and IOWRITE_WORD procedures.

In most instances, it is unnecessary for the programmer to access the I/O system registers. Some of the more common register operations are supported in high level procedures and functions. It is best to use the high level procedures and functions when possible because these are more easily understood and are more transportable. Refer to the chapters that deal with the specific interface for the high level procedures and functions.

Firmware Registers

The firmware registers are called the status and control registers. In previous desktop computers and in the HP BASIC language these firmware registers are accessed with the BASIC STATUS and CONTROL statements. In the Pascal system, most of the firmware registers have the same definitions as the BASIC system. This is only mentioned in case you already have an understanding of the BASIC firmware registers.

The IOSTATUS Function

A status register is read with the IOSTATUS function. It is necessary to specify the interface and the register number of interest in the parameter list. Only a single register may be examined with each invocation of IOSTATUS.

Examples

```
interface := 12;
register  := 0;                          { reg 0 is card id }
i := IOSTATUS(interface,register);       { get interface id }

WRITELN('bus state is ',IOSTATUS(7,7));  { get HP-IB bus state }
```

The IOCONTROL Procedure

A control register is written with the IOCONTROL procedure. It is necessary to specify the interface and the register number, and the value to be written in the parameter list. Only a single register may be modified with each invocation of IOCONTROL.

Examples

```
interface := 12;
register  := 3;                          { reg 3 sets HP-IB addr }
IOCONTROL(interface,register,5);         { set my card to addr 5 }

IOCONTROL(7,0,1);                        { reset HP-IB card      }
```

Common Register Definitions

The status and control registers are very interface dependent both in number and definition of the registers. There are two registers that are defined for all except two interfaces:

- status register 0 (for card identification)
- control register 0 (to reset the interface card)

The keyboard and CRT (interface select codes 1 and 2) do not have status and control registers implemented.

Hardware Registers

The hardware registers are accessed by the system firmware. It is, therefore, dangerous for you to access these registers unless you have a complete understanding of both the register definition and of the consequences of accessing the hardware registers. Their locations and definitions are given in Appendix A of the Pascal Language System User's Manual. The IOREAD_BYTE and IOWRITE_BYTE perform an eight bit (byte) operation on the computer backplane. The IOREAD_WORD and IOWRITE_WORD perform a 16-bit (word) operation on the computer backplane.

Chapter 10
Errors and Timeouts

Introduction

There are two types of events supported in the Pascal I/O Library:

- I/O Errors
- I/O Timeouts

These I/O events are handled via the TRY/RECOVER event handling mechanism. Refer to the Compiler chapter of the Pascal System User's Manual for additional information on TRY/RECOVER.

Note that timeouts are only available on handshake operations. There is no timeout facility on the advanced transfers. Also note that the Datacomm interface control blocks use the TRY/RECOVER mechanism.

Pascal Event Processing

Pascal's event handling is very much different from that found in BASIC or HPL on the 9826 and 9836. BASIC and HPL are interpreted languages and at the end of each line there is a system code to check for event conditions and take the appropriate branch if necessary. The Pascal compiler does not generate code at the end of each line to check for conditions. Pascal takes advantage of a hardware feature that allows an event to escape from whatever code is currently being executed to a previously defined event handler. An example program using this event handling is:

```
$SYSPROG ON$                        { enable optional compiler features }
PROGRAM errors (INPUT,OUTPUT);
  VAR a : REAL;
  BEGIN
   TRY
     a := 1;
     a := a/0;                      { this should generate an error }
     WRITELN('This should not get executed');
   RECOVER                          { this is the event handler     }
     BEGIN
       WRITELN('I have gotten an error');
       WRITELN('The escape code is ',ESCAPECODE);
       ESCAPE(ESCAPECODE);          { pass error on                 }
     END;

  WRITELN('Program finished normally');
END.
```

When run, this program will generate a CRT screen similar to the following:

```
I have gotten an error
The escape code is           -5

------------------------------
error -5: divide by zero
PC value:      -444090
```

The error handling in Pascal depends on four language features:

- TRY
- RECOVER
- ESCAPECODE
- ESCAPE

These features are not in the normal Pascal language. To access these features it is necessary to turn on a compiler option called SYSPROG. This compiler option enables error handling and several other system features. Refer to the Compiler chapter of the Pascal System User's Manual for additional information about $SYSPROG ON$.

TRY

TRY defines the start of a block of code that is to be handled by a following RECOVER block. This block of code may contain anything including procedure and function calls. If any error occurs, it will be handled by the RECOVER block, unless there is a nested TRY/RECOVER block. TRY/RECOVER blocks may be nested to any level. The inner-most RECOVER block will receive control.

If no error occurs in a TRY/RECOVER block then the next statement following the RECOVER block is executed.

RECOVER

RECOVER defines the start of the error handling code. The RECOVER code must be a simple statement or a BEGIN/END block.

ESCAPECODE

ESCAPECODE is an INTEGER variable that contains the error code from the last error. System errors have negative values. User errors should have positive values.

ESCAPE

ESCAPE is a procedure that generates an error escape. It has a single INTEGER parameter. When ESCAPE is executed it places the parameter into the ESCAPECODE variable and generates an error. This error will be trapped by a RECOVER block, if any.

I/O Error Handling

I/O errors are just one of several error conditions that can occur in the Pascal system. Because of the multitude of errors that can happen within device I/O, only one ESCAPECODE has been allocated for use by the I/O Library. When ESCAPECODE has the value −26, the error was an I/O error.

The I/O Library uses some additional variables and functions for the various errors that it can generate:

- IOESCAPECODE
- IOE_RESULT
- IOE_ISC
- IOERROR_MESSAGE

IOESCAPECODE

IOESCAPECODE is an integer constant with the value −26. This constant is compared with the ESCAPECODE to determine if the ESCAPE was due to an I/O error. The constant IOESCAPECODE is defined in the I/O Library Module IODECLARATIONS.

IOE_RESULT

IOE_RESULT is an integer variable. This variable contains the specific I/O error code, if any. The variable IOE_RESULT is defined in the I/O Library Module IODECLARATIONS. A listing of current error codes and their messages is in the last section in this chapter. For each error code, the I/O Library has defined a constant for that error. For example, when IOE_RESULT has the value 11, the error is that there is no firmware to support the interface card in the system. This error has a constant defined in IODECLARATIONS called ioe_no_driver that is defined to have the decimal value 11.

IOE_ISC

IOE_ISC is an integer variable. This variable contains the interface select code of the last interface to generate an I/O error. If the error was not due to an interface problem, then IOE_ISC will contain the value 255 (which is NO_ISC). The variable IOE_ISC is defined in the I/O Library Module IODECLARATIONS.

IOERROR_MESSAGE

IOERROR_MESSAGE is a string function. This function has one INTEGER parameter that should contain the I/O error code IOE_RESULT. The function returns a string that is the English error message associated with the specific error code. The string function IOERROR_MESSAGE is in the I/O Library Module GENERAL_3. A listing of current error codes and their messages is in the last section in this chapter.

The following program is an example of handling an I/O error using the TRY/RECOVER mechanism used with the features of the I/O Library. This program attempts to write a string out to an HP-IB interface without first addressing the interface card as a talker.

```
$SYSPROG ON$                        { enable optional compiler features }
PROGRAM io_errors (INPUT,OUTPUT);
  IMPORT  IODECLARATIONS,
          GENERAL_1,
          GENERAL_2,
          GENERAL_3;
  BEGIN
   TRY
     IOINITIALIZE;                  { put I/O system into known state }
     WRITESTRINGLN(7,'I am not sending address information');
     WRITELN('This should not get executed');
   RECOVER                          { this is the event handler       }
     BEGIN
       WRITELN('I have gotten an error');
       WRITELN('The escape code is ',ESCAPECODE);
       IF ESCAPECODE=IOESCAPECODE
         THEN BEGIN
           WRITELN('The error was an I/O error');
           WRITELN(IOERROR_MESSAGE(IOE_RESULT),' on isc ',IOE_ISC);
         END
         ELSE BEGIN
           ESCAPE(ESCAPECODE);      { pass error on                   }
         END;
     END;
  WRITELN('Program finished normally');
END.
```

When run, this program will generate a CRT screen similar to the following:

```
I have gotten an error
The escape code is           -26
The error was an I/O error
not addressed as talker on isc           7
Program finished normally
```

Note that the program finished normally. The path that was executed inside the RECOVER block did not perform an ESCAPE. Therefore, the statement immediately following the RECOVER block is executed next.

It is important to structure your TRY/RECOVER blocks in a manner similar to the one just shown. This is necessary because **all** errors go through the TRY/RECOVER mechanism. If you do not check the cause of the error with ESCAPECODE, you might trap an error meant for some other TRY/RECOVER or an error you did not expect.

I/O Timeouts

A timeout occurs when the handshake response from any external device takes longer than a specified amount of time to complete. The time specified for the timeout is usually the maximum time that a device can be expected to take to respond to a handshake during an I/O statement.

Setting Up Timeout Events

The SET_TIMEOUT procedure in Module GENERAL_1 has two parameters, the interface select code and a single REAL parameter that is the time that the I/O Library will wait for an operation to complete. This parameter is the time in seconds. The parameter can range from 0 thru 8191 seconds with a resolution of .001 seconds. The default timeout value is 0, which is interpreted by the I/O Library as a timeout period of infinity—the system will wait forever for the operation to complete.

The timeout event is just another I/O error. The timeout error has the I/O error code (IOE_RESULT) of 17 (I/O error constant ioe_timeout).

A sample program trapping timeouts follows. This program will try to send some data to a device ten times and will then stop.

```
$SYSPROG ON$                          { enable optional compiler features }
PROGRAM timeouts (INPUT,OUTPUT);
  IMPORT  IODECLARATIONS,
          GENERAL_1,
          GENERAL_2,
          GENERAL_3;
  VAR attempt : INTEGER;
      success : BOOLEAN;
  BEGIN
   IOINITIALIZE;
   SET_TIMEOUT(7,1.0);                 { timeout of 1 second on isc 7   }
   attempt := 1;
   success := FALSE;
   REPEAT
     TRY
       WRITESTRINGLN(724,'This device does not exist on the bus');
       success := TRUE;
     RECOVER                          { this is the event handler      }
       BEGIN
         IF ESCAPECODE=IOESCAPECODE
           THEN BEGIN
             IF ( IOE_RESULT = IOE_TIMEOUT ) AND ( IOE_ISC = 7 )
               THEN BEGIN
                 IORESET(7);          { because interface is in a bad state }
                 WRITELN('timeout #',attempt:2);
                 attempt := attempt+1;
               END
               ELSE BEGIN
                 WRITELN(IOERROR_MESSAGE(IOE_RESULT),' on isc ',IOE_ISC);
                 ESCAPE(ESCAPECODE);
               END;
           END
           ELSE BEGIN
             ESCAPE(ESCAPECODE);      { pass error on                  }
           END;
       END;
  UNTIL ( attempt>10 ) OR success;
  WRITELN('Program finished');
  IOUNINITIALIZE;                     { clean up interface state       }
END.
```

When run, this program will generate a CRT screen similar to the following:

```
timeout # 1
timeout # 2
timeout # 3
timeout # 4
timeout # 5
timeout # 6
timeout # 7
timeout # 8
timeout # 9
timeout #10
Program finished
```

I/O Errors

The following list contains the error codes in the I/O Library. The error code value is stored in the system variable IOE_RESULT. This list also contains the text of the error message produced by the GENERAL_3 string function IOERROR_MESSAGE. The name of the error is a constant that is declared in the IODECLARATIONS Module. The errors from 306 through 327 are HP 98628A Datacomm interface errors.

Name	Value	Error Message
ioe_no_error	0	no error
ioe_no_card	1	no card at select code
ioe_not_hpib	2	interface should be hpib
ioe_not_act	3	not active controller
ioe_not_dvc	4	should be device not sc
ioe_no_space	5	no space left in buffer
ioe_no_data	6	no data left in buffer
ioe_bad_tfr	7	improper transfer attempted
ioe_isc_busy	8	the select code is busy
ioe_buf_busy	9	the buffer is busy
ioe_bad_cnt	10	improper transfer count
ioe_bad_tmo	11	bad timeout value
ioe_no_driver	12	no driver for this card
ioe_no_dma	13	no dma
ioe_no_word	14	word operations not allowed
ioe_not_talk	15	not addressed as talker
ioe_not_lstn	16	not addressed as listener
ioe_timeout	17	a timeout has occurred
ioe_not_sctl	18	not system controller
ioe_rds_wtc	19	bad status or control
ioe_bad_sct	20	bad set/clear/test operation
ioe_crd_dwn	21	interface card is dead
ioe_eod_seen	22	end/eod has occurred
ioe_misc	23	miscellaneous - value of param error
ioe_dc_fail	306	dc interface failure
ioe_dc_usart	313	USART receive buffer overflow
ioe_dc_ovfl	314	receive buffer overflow
ioe_dc_clk	315	missing clock
ioe_dc_cts	316	CTS false too long
ioe_dc_car	317	lost carrier disconnect
ioe_dc_act	318	no activity disconnect
ioe_dc_conn	319	connection not established
ioe_dc_conf	325	bad data bits/par combination
ioe_dc_reg	326	bad status /control register
ioe_dc_rval	327	control value out of range

Chapter 11
Advanced Transfer Techniques

Introduction

This chapter discusses advanced transfer techniques. These transfers are intended primarily for two main applications:

- Where the computer is much faster than the device being communicated with
- Where the computer is slower than the device being communicated with

This chapter includes discussions on buffers, serial transfers, overlap transfers and special forms of transfers.

Buffers

Buffers are the data area where the transfer procedures read and write the data that is being transferred. This area is actually in two pieces. One piece is the control block for the buffer. The other is the memory where data is actually stored.

The control block is a user variable. This variable must be of the type BUF_INFO_TYPE which is defined in the I/O Library module IODECLARATIONS. This block of information contains various fields including a pointer to the actual data area.

The data area is not allocated when the BUF_INFO_TYPE variable is declared. The data area is allocated at program execution time with the execution of a procedure called IOBUFFER. This procedure is of the form:

```
IOBUFFER (buffer_control_block, size_in_bytes);
```

The size in bytes is an integer value and can be of any size that the memory in your computer can create. The IOBUFFER procedure, at program execution time, will allocate the data area and initialize the various pointers in the buffer control block (a variable of BUF_INFO_TYPE). IOBUFFER and all other I/O Library transfer procedures are in the GENERAL_4 module.

The data area that is allocated is allocated with the NEW facility. Refer to the Pascal Handbook for more information on NEW and its related capabilities. In particular, be careful of the MARK and RELEASE facilities since these can affect the buffer space.

Once a buffer has been declared and allocated, it is necessary to be able to read and write the buffer. The I/O Library, as with normal input and output, has a small number of procedures and functions to access the buffer space. These procedures and functions are:

- BUFFER_RESET
- BUFFER_SPACE
- BUFFER_DATA
- READBUFFER
- WRITEBUFFER
- READBUFFER_STRING
- WRITEBUFFER_STRING

Buffer Control

Necessary aspects of buffer control are empty and fill pointers. When data is written into the buffer, the fill pointer is incremented. When data is read from the buffer the empty pointer is incremented. When these two pointers meet, there is no data in the buffer.

The procedure BUFFER_RESET puts the empty and fill pointers back to the start of the buffer—effectively clearing it of data. The form of this procedure is:

```
BUFFER_RESET (buffer_control_block);
```

The integer function BUFFER_SPACE returns the number of bytes that are available at the end of the buffer from the fill pointer to the end of the buffer. This function is of the form:

```
BUFFER_SPACE (buffer_control_block);
```

The integer function BUFFER_DATA returns the number of bytes of data that are available in the buffer from the empty pointer to the fill pointer. This function is of the form:

```
BUFFER_DATA (buffer_control_block);
```

Reading Buffer Data

There are two procedures that read buffer data: READBUFFER and READBUFFER_STRING. READBUFFER reads a single character. READBUFFER_STRING reads a string. The form of these procedures is:

```
READBUFFER (buffer_control_block, character_var);
READBUFFER_STRING (buffer_control_block, string_var,
                   character_count );
```

The READBUFFER_STRING will read the specified number of characters from the buffer into the string variable.

Writing Buffer Data

There are two procedures that write buffer data: WRITEBUFFER and WRITEBUFFER_STRING. WRITEBUFFER writes a single character. WRITEBUFFER_STRING writes a string. The form of these procedures is:

```
WRITEBUFFER (buffer_control_block, character);
WRITEBUFFER_STRING (buffer_control_block, string);
```

The WRITEBUFFER_STRING will write the entire number of characters from the string expression into the buffer.

The following is an example program showing the creation and use of a buffer:

```
PROGRAM buffers (INPUT,OUTPUT);
IMPORT        IODECLARATIONS,
              GENERAL_4;
VAR buffer : BUF_INFO_TYPE;
    i      : INTEGER;
    c      : CHAR;
BEGIN

  IOBUFFER(buffer,100);               { create a 100 character buffer }
  BUFFER_RESET(buffer);               { make sure it is empty         }

  FOR i:=65 TO 90 DO
    WRITEBUFFER(buffer,chr(i));       { put character data in the buf }
  WRITEBUFFER_STRING(buffer,'hello'); { put a string in the buffer    }

  WHILE BUFFER_DATA(buffer)>0 DO BEGIN
    READBUFFER(buffer,c);             { dump out the buffer by char   }
    WRITE(c);
  END;  { of WHILE DO BEGIN }
  WRITELN;

END.
```

This program will produce the following screen on the CRT:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZhello
```

Serial Transfers

Serial transfers are those that complete before the next Pascal line is executed. This is the normal approach that Pascal uses in program execution. This type of transfer is useful in the application where you have a high speed data transfer where the computer is slower than or the same speed as the device.

The procedure that performs a data transfer to and from a buffer is the TRANSFER procedure. It has the following form:

```
TRANSFER (device, transfer_mode, direction,
          buffer_control_block, count);
```

The device is the device specifier described in previous chapters (like 12 or 701). The count is the number of bytes to be transferred by the procedure. The buffer control block is the buffer variable of type BUF_INFO_TYPE.

The direction parameter is of a special type and can have two values: FROM_MEMORY and TO_MEMORY. So a direction of FROM_MEMORY is an output transfer and TO_MEMORY is an input transfer.

The transfer mode is also of a special type. For serial transfers it can have the values:

- SERIAL_DMA
- SERIAL_FHS
- SERIAL_FASTEST

The DMA mode specifies a dma transfer. The FHS mode specifies a fast handshake transfer. The FASTEST mode specifies that if DMA is installed and available for the transfer, then it should be used, otherwise a FHS transfer will occur. Some interfaces do not support DMA transfers (like the Serial Data Comm interface). Those interfaces, when a FASTEST transfer is requested, will give a FHS transfer since they cannot do DMA.

The DMA mode transfer can only transfer 1 through 65536 bytes of data. The fast handshake transfer can be of arbitrary size.

An example program using a serial transfer to a printer is:

```
PROGRAM transfers (INPUT,OUTPUT);
IMPORT        IODECLARATIONS,
              GENERAL_4;
VAR buffer : BUF_INFO_TYPE;
    i,j    : INTEGER;
    c      : CHAR;
BEGIN

  IOBUFFER(buffer,100);                { create a 100 character buffer }

  FOR j:=1 TO 5 DO BEGIN

    BUFFER_RESET(buffer);              { make sure it is empty          }
    FOR i:=65 TO 90 DO
      WRITEBUFFER(buffer,chr(i));      { put character data in the buf  }
    WRITEBUFFER(buffer,chr(13));       { put in a carriage return       }
    WRITEBUFFER(buffer,chr(10));       { put in a line feed             }
    TRANSFER(701,SERIAL_FASTEST,
             FROM_MEMORY,buffer,
             buffer_data(buffer));     { send all of the data in buf    }
    WRITELN('this line will not be printed until the transfer is done');

  END; { of FOR DO BEGIN }

END.
```

This program will produce the following on the CRT:

```
this line will not be printed until the transfer is done
this line will not be printed until the transfer is done
this line will not be printed until the transfer is done
this line will not be printed until the transfer is done
this line will not be printed until the transfer is done
```

and this on the PRINTER:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Overlap Transfers

Serial transfers are useful for high-speed applications. The computer will not continue execution of the program until the transfer is complete. For lower speed applications, this is not adequate. The Pascal I/O Library provides an overlap transfer mechanism. This mechanism allows for the program to continue execution while the transfer is continuing. The overlap transfer mechanism is identical to the serial transfer. Its form is:

```
TRANSFER (device, transfer_mode, direction,
          buffer_control_block, count);
```

All of the parameters are the same with the exception of the transfer_mode. The mode parameter can have the following values for overlap transfers:

Transfer Mode Value	Meaning
OVERLAP_INTR	Interrupt transfer
OVERLAP_DMA	dma transfer
OVERLAP_FHS	Interrupt on first byte fast handshake on rest
OVERLAP_FASTEST	dma if available, else use overlap_fhs
OVERLAP	dma if available, else use overlap_intr

The overlap fast handshake mode has also been called burst mode, because it does not consume any CPU time until the first byte is transferred. The overlap mode is provided so that if your application requires a data transfer to execute concurrently with the program execution, then you will get the most efficient method available.

The DMA mode transfer can only transfer 1 through 65 536 bytes of data. The other transfer modes can be of arbitrary size.

When is the Transfer Finished?

There are two BOOLEAN functions which can tell you if a transfer is still occurring between a buffer and an interface. These are:

```
BUFFER_ACTIVE( buffer_control_block);
```

and

```
ISC_ACTIVE( interface_select_code );
```

Either function returns TRUE if the transfer is still active.

The following program is an example of an overlap transfer. This program does not do anything useful with the spare time available to it.

```
PROGRAM overlaped (INPUT,OUTPUT);
IMPORT        IODECLARATIONS,
              GENERAL_4;
VAR buffer : BUF_INFO_TYPE;
    i,j    : INTEGER;
    c      : CHAR;
BEGIN

  IOBUFFER(buffer,100);                { create a 100 character buffer }

  FOR j:=1 TO 5 DO BEGIN

    WHILE  BUFFER_ACTIVE( buffer ) DO
      BEGIN
        WRITELN('waiting for transfer to finish');
      END;

    BUFFER_RESET(buffer);              { make sure it is empty          }
    FOR i:=65 TO 90 DO
      WRITEBUFFER(buffer,chr(i));      { put character data in the buf }
    WRITEBUFFER(buffer,chr(13));       { put in a carriage return       }
    WRITEBUFFER(buffer,chr(10));       { put in a line feed             }
    TRANSFER(701,OVERLAP_INTR,
             FROM_MEMORY,buffer,
             buffer_data(buffer));     { send all of the data in buf    }

  END; { of FOR DO BEGIN }

END.
```

This program will produce the following on the PRINTER:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Special Transfers

In addition to the block transfers that were described above, there are three additional versions of transfer. They are:

- word transfers
- match character transfers
- END condition transfers

Word Transfer

The GPIO interface can support 16 bit data transfers. The TRANSFER_WORD procedure simultaneously transfers 2 bytes over the GPIO interface. The form of this procedure is:

```
TRANSFER_WORD  (device, transfer_mode, direction,
                              buffer_control_block, count);
```

All of the parameters are the same with the exception of the count which now contains the 16-bit word count to be transferred. All the transfer types, overlap and serial, are the same as a regular transfer.

Match Character Transfer

This transfer procedure will transfer data into the computer until a match character is found. Note that this transfer, called TRANSFER_UNTIL, is an input only transfer. The form of the procedure is:

```
TRANSFER_UNTIL (termination_char, device, transfer_mode,
                              direction, buffer_control_block);
```

The termination character is the match character that will stop the transfer. The transfer will also stop when the there is no more room in the buffer. All of the other parameters are the same. Most of the transfer types, overlap and serial, are the same as a regular transfer - except that DMA transfers are not allowed. Note that there is NO count parameter. The direction must be TO_MEMORY.

END Condition Transfer

This transfer procedure will transfer data into the computer until an interface condition occurs or it will transfer data out with the last data byte being sent with an interface condition. This transfer is TRANSFER_END and has the form:

```
TRANSFER_END   (device, transfer_mode, direction,
                     buffer_control_block);
```

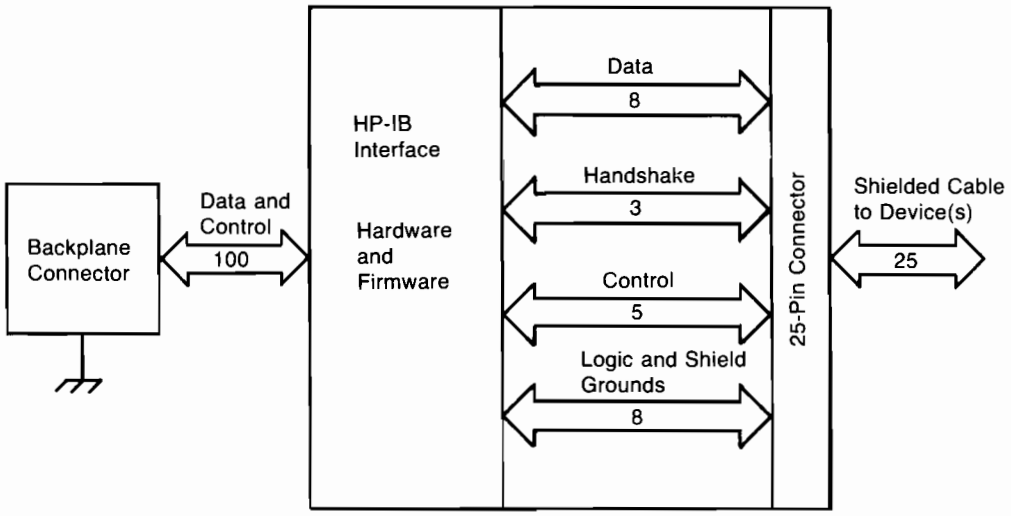
All of the parameters are the same. Note that there is NO count. The transfer will send all the available data followed by the condition or will receive data until the end condition occurs or the buffer fills up. All the transfer types, overlap and serial, are the same as a regular transfer. An example of an end condition is the EOI condition on HP-IB.

Chapter 12
The HP-IB Interface

Introduction

This chapter describes the techniques necessary for programming the HP-IB interface. Many of the elementary concepts have been discussed in previous chapters. This chapter describes the specific details of how this interface works and how it is used to communicate with and control systems consisting of various HP-IB devices.

The HP-IB (Hewlett-Packard Interface Bus), commonly called the "bus", provides compatibility between the computer and external devices conforming to the IEEE 488-1978 standard. Electrical, mechanical, and timing compatibility requirements are all satisfied by this interface.



The HP-IB interface is both easy to use and allows great flexibility in communicating data and control information between the computer and external devices. It is one of the easiest methods to connect more than one device to the same interface.

Initial Installation

Refer to the HP-IB Installation Note for information about setting the switches and installing an external HP-IB interface. Once the interface has been properly installed, you can verify that the switch settings are what you intended by running the following program. The defaults of the internal HP-IB interface can also be checked with the program. The results are displayed on the CRT.

```
PROGRAM check_hpib ( INPUT , OUTPUT );
  IMPORT IODECLARATIONS,
         HPIB_1;
  VAR  isc : TYPE_ISC;
  BEGIN
    WRITELN('Enter HP-IB interface select code');
    READLN(isc);

    IF ISC_TABLE[isc].CARD_TYPE <> HPIB_CARD
      THEN BEGIN
        WRITELN('The interface at isc ',isc:2,' is not an HP-IB interface');
      END
      ELSE BEGIN
        WRITELN('The interface at isc ',isc:2,' is an HP-IB interface');

        IF ISC_TABLE[isc].CARD_ID = HP98624
          THEN WRITELN('   and is an  optional, external interface')
          ELSE WRITELN('   and is the standard, built in interface');

        WRITE('The interface is ');
        IF NOT SYSTEM_CONTROLLER(isc) THEN WRITE('NOT ');
        WRITELN('the system controller');

        WRITE('The interface has a bus address of ',my_address(isc):2);

      END; { of IF THEN/ELSE }
  END.
```

The terms system controller and bus address are described in the following sections. The internal HP-IB has a jumper that is set at the factory to make it a system controller. This jumper is located below the lowest interface slot at the computer backplane. The lowest interface (or memory board) in the backplane must be removed to access this jumper. If the jumper in the center of the clear plastic cover is placed on the middle and right most pins, as seen from the rear of the computer, the computer is set to be a system controller. If the jumper is on the middle and leftmost pins, then the computer is not system controller and will have a bus address of 20.

Communicating with Devices

This section describes programming techniques used to output data to and enter data from HP-IB devices. General bus operation is also briefly described.

HP-IB Device Selectors

Since the HP-IB allows the interconnection of several devices, each device must have a means of being uniquely accessed. Specifying just the interface select code of the HP-IB interface through which a device is connected is not sufficient to identify that device on the bus.

Each device connected to the bus has an address by which it can be identified. This address must be unique to allow individual access of each device. Most HP-IB devices have a set of switches that are used to set its address. Those that do not have switches, like the built in HP-IB interface in the computer, have a pre-set bus address. So, when a particular HP-IB device is to be accessed, it must be identified with both its interface and its bus address.

The interface select code is the first part of an HP-IB device selector. The interface select code of the internal HP-IB is 7. The second part of an HP-IB device specifier is the device's bus address. This address is the range of 0 through 30. As described in the Directing Data Flow chapter, interface 7, device address 17 would have a device specifier of 717. Interface 10, device address 2 would have a device specifier of 1002.

Moving Data Through the HP-IB

Data is output from and entered into the computer through the output and input procedures described in Chapters 7 and 8. All the information in these chapters applies directly to the HP-IB interface. The advanced transfer techniques of Chapter 11 also apply to the HP-IB interface.

Example

```
PROGRAM hpib_io (INPUT,OUTPUT);
  IMPORT        GENERAL_2;
  VAR a       : REAL;
      i       : INTEGER;
  BEGIN
    WRITESTRINGLN(701,'message to a printer');
    WRITESTRINGLN(724,'R1T1N1S');
    FOR i:= 1 TO 100 DO BEGIN
      READNUMBER    (724,a);
      WRITELN('the reading from the voltmeter is ',a:6:2);
    END; { of FOR DO BEGIN }
  END.
```

General Structure of the HP-IB

Communications through the HP-IB are made according to a precisely defined set of rules. These rules help to ensure that only orderly communication may take place on the bus. For conceptual purposes, the organization of the HP-IB can be compared to that of a committee. A committee has certain "rules of order" that govern the manner in which business is to be conducted. For the HP-IB, these rules of order are the IEEE 488-1978 standard.

One member, designated the "committee chairman," is set apart for the purpose of conducting communications between members during the meetings. This chairman is responsible for overseeing the actions of the committee and generally enforces the rules of order to ensure the proper conduct of business. If the committee chairman cannot attend a meeting, he designates some other member to be "acting chairman."

On the HP-IB, the **system controller** corresponds to the committee chairman. The system controller is generally designated by setting a switch on the interface and cannot be changed under program control. However, it is possible to designate an "acting chairman" on the HP-IB. On the HP-IB, this device is called the **active controller**, and may be any device capable of directing HP-IB activities, such as a desktop computer.

When the system controller is first turned on or reset, it assumes the role of active controller. Thus, only one device can be designated system controller. These responsibilities may be subsequently passed to another device while the system controller tends to other business. This ability to pass control allows more than one computer to be connected to the HP-IB at the same time.

In a committee, only one person at a time may speak. It is the chairman's responsibility to "recognize" which one member is to speak. Usually, all committee members present always listen; however, this is not always the case on the HP-IB. One of the most powerful features of the bus is the ability to selectively send data to individual (or groups of) devices.

Imagine slow note takers and fast note takers on the committee. Suppose that the speaker is allowed to talk no faster than the slowest note taker can write. This would guarantee that everybody gets the full set of notes and that no one misses any information. However, requiring all presentations to go at that slow pace certainly imposes a restriction on our committee, especially if the slow note takers do not need the information. Now, if the chairman knows which presentations are not important to the slow note takers, he can direct them to put away their notes for those presentations. That way, the speaker and the fast note taker(s) can cover more items in less time.

A similar situation may exist on the HP-IB. Suppose that a printer and a flexible disc are connected to the bus. Both devices do not need to listen to all data messages sent through the bus. Also, if all the data transfers must be slow enough for the printer to keep up, saving a program on the disc would take as long as listing the program on the printer. That would certainly not be a very effective use of the speed of the disc drive if it was the only device to receive the data. Instead, by "unlistening" the printer whenever it does not need to receive a data message, the computer can save a program as fast as the disc can accept it.

During a committee meeting, the current chairman is responsible for telling the committee which member is to be the talker and which is (are) to be the listener(s). Before these assignments are given, he must get the **attention** of all members. The talker and listener(s) are then designated, and the next data message is presented to the listener(s) by the talker. When the talker has finished the message, the designation process may be repeated.

On the HP-IB, the active controller takes similar action. When talker and listener(s) are to be designated, the **attention signal line** (ATN) is asserted while the talker and listener(s) are being addressed. ATN is then cleared, signaling that those devices not addressed to listen may ignore all subsequent data messages. Thus, **the ATN line separates data from commands**; commands are accompanied by the ATN line being true, while data messages are sent with the ATN line false.

On the HP-IB, devices are **addressed to talk** and **addressed to listen** in the following orderly manner. The active controller first sends a single command which causes all devices to **unlisten**. The talker's address is then sent, followed by the address(s) of the listener(s). After all listeners have been addressed, the data can be sent from the talker to the listener(s). Only device(s) addressed to listen accept any data that is sent through the bus (until the bus is reconfigured by subsequent addressing commands).

The data transfer, or **data message**, allows for the exchange of information between devices on the HP-IB. Our committee conducts business by exchanging ideas and information between the speaker and those listening to his presentation. On the HP-IB, **data is transferred from the active talker to the active listener(s) at a rate determined by the slowest active listener on the bus**. This restriction on the transfer rate is necessary to ensure that no data is lost by any device addressed to listen. The **handshake** used to transfer each data byte ensures that all data output by the talker is received by all active listeners.

Examples of Bus Sequences

Most data transfers through the HP-IB involve a talker and only one listener. For instance, when an input or output procedure is used to send data to or from a device, the following sequence of commands is sent through the bus.

```
WRITESTRINGLN(701,'Data');
```

1. The unlisten command is sent.
2. The talker's address is sent (the computer's talk address).
3. The listener's address is sent (address 01).
4. The data bytes "D","a","t","a",carriage return and line feed are sent.

```
READSTRING(724,Message);
```

1. The unlisten command is sent.
2. The talker's address is sent (talk address for device 24).
3. The listener's address is sent (the computer listen address).
4. The data bytes are transferred.

Addressing Multiple Listeners

HP-IB allows more than one device to listen as data is sent through the bus. The PASCAL I/O Library supports this capability in the following way. It is necessary for you to address the bus yourself. The procedures to do this addressing exist in the module HPIB_2. The following example shows how to address the computer as a talker and several devices as listeners.

```
UNLISTEN(isc);
TALK     (isc,my_address(isc));
LISTEN   (isc,address_1);
LISTEN   (isc,address_2);
LISTEN   (isc,address_3);
WRITESTRINGLN(isc,'message to three devices');
```

An example where the computer is one of several devices listening to some incoming data is :

```
UNLISTEN(isc);
TALK     (isc,address_1);
LISTEN   (isc,my_address(isc));
LISTEN   (isc,address_2);
LISTEN   (isc,address_3);
READSTRING(isc,str);
```

The UNLISTEN, TALK and LISTEN procedures are in the I/O Library module HPIB_2.

Addressing a Non-Controller Computer

The bus standard states that a non-active controller cannot perform any bus addressing. When only the interface select code is specified in an input or output procedure, no bus addressing occurs.

If the computer currently is not the active controller, it can still act as a talker or listener, provided it has been previously addressed. So, if an input or output procedure is executed while the computer is not an active controller, the computer first determines whether or not it is an active talker or listener. If not addressed to talk or listen, the computer waits until it is properly addressed and then performs the operation. Examples of non-controller I/O are:

```
READCHAR(7,c);
WRITESTRINGLN(7,'I am just a device');
READSTRING_UNTIL(chr(13),7,str);
```

If the computer is the active controller, it proceeds with the data transfer without addressing which devices are talker and listener(s). If the bus has not been configured properly (the controller not being addressed as a talker or listener), an error is reported. The escapecode is −26 (I/O) and the io error is 15 or 16 (not addressed as a talker or listener). The following program shows a typical use of this non-addressing approach.

```
WRITESTRINGLN(705,'I go to device 5 on isc 7');
LISTEN(7,1); WRITESTRINGLN(7,'I go to device 1 and 5');
LISTEN(7,20);
FOR i:=1 TO 10 DO WRITESTRINGLN('10 lines to devices 1,
       5,20');
```

PASCAL Control of HP-IB

The PASCAL I/O Library has a number of procedures and functions for control of the HP-IB bus. You have seen a number of them already in the preceding examples. The normal bus control capabilities are broken down into two major groups - status and control.

HP-IB Status

Normal use of HP-IB requires three main status facilities:

- What is my address?
- Am I system controller?
- Am I active controller?

The function MY_ADDRESS returns the current device address of the specified interface. This integer function is in module HPIB_1. It has the form:

```
MY_ADDRESS ( interface_select_code );
```

The function SYSTEM_CONTROLLER returns a TRUE or FALSE depending on whether the interface is a system controller. This boolean function is in module HPIB_1 and has the form:

```
SYSTEM_CONTROLLER ( interface_select_code );
```

The function ACTIVE_CONTROLLER returns a TRUE or FALSE depending on whether the interface is a active controller. This boolean function is in module HPIB_1 and has the form:

```
ACTIVE_CONTROLLER ( interface_select_code );
```

HP-IB Control

Normal use of HP-IB requires five main control facilities:

- Send untalk
- Send unlisten
- Send a talk command
- Send a listen command
- Send a secondary command

The UNTALK and UNLISTEN procedures send the appropriate command on the bus. These procedures are in the HPIB_2 module. The interface must be active controller for them to complete. They have the form:

```
UNTALK      ( interface_select_code );

UNLISTEN    ( interface_select_code );
```

The TALK, LISTEN and SECONDARY commands send a talk, listen or secondary command. These procedures are in the HPIB_2 module. The interface must be an active controller form for them to complete. They have the form:

```
TALK        ( interface_select_code , address );

LISTEN      ( interface_select_code , address );

SECONDARY   ( interface_select_code , address );
```

General Bus Management

The HP-IB standard provides several mechanisms that allow managing the bus and the devices on the bus. Here is a summary of the procedures that invoke these control mechanisms.

ABORT_HPIB is used to abruptly terminate all bus activity and reset all devices to power-on states.

CLEAR is used to set all (or only selected) devices to a pre-defined, device-dependent state.

LOCAL is used to return all (or selected) devices to local (front-panel) control.

LOCAL_LOCKOUT is used to disable all devices' front-panel controls.

PASS_CONTROL is used to pass active control to another device on the bus.

PPOLL is used to perform a parallel poll on all devices (which are configured and capable of responding).

PPOLL_CONFIGURE is used to setup the parallel poll response of a particular device.

PPOLL_UNCONFIGURE is used to disable the parallel poll response of a device (or all devices on an interface).

REMOTE is used to put all (or selected) devices into their device-dependent, remote modes.

SEND_COMMAND is used to manage the bus by sending explicit command messages.

SPOLL is used to perform a serial poll of the specified device (which must be capable of responding).

TRIGGER is used to send the trigger message to a device (or selected group of devices).

These procedures (and functions) are described in the following discussion. However, the actions that a device takes upon receiving each of the above commands are, in general, different for each device. Refer to a particular device's manuals to determine how it will respond. Detailed descriptions of the actual sequence of bus messages invoked by these statements are contained in "Advanced Bus Management" near the end of this chapter.

Remote Control of Devices

Most HP-IB devices can be controlled either from the front panel or from the bus. If the device's front-panel controls are currently functional, it is in the Local state. If it is being controlled through the HP-IB, it is in the Remote state. Pressing the front-panel "Local" key will return the device to Local (front-panel) control, unless the device is in the Local Lockout state (described in a subsequent discussion).

The Remote message is automatically sent to all devices whenever the system controller is powered on, reset, or sends the Abort message. A device also enters the Remote state automatically whenever it is addressed. The REMOTE procedure also outputs the Remote message, which causes all (or specified) devices on the bus to change from local control to remote control. The 9826 must be the system controller to execute the REMOTE procedure. The REMOTE procedure is in module HPIB_2.

Examples

```
REMOTE (7) ;

REMOTE (700) ;
```

Locking Out Local Control

The Local Lockout message effectively locks out the "local" switch present on most HP-IB device front panels, preventing a device's user from interfering with system operations by pressing buttons and thereby maintaining system integrity. As long as Local Lockout is in effect, no bus device can be returned to local control from its front panel.

The Local Lockout message is sent by executing the LOCAL_LOCKOUT procedure. This message is sent to all device on the specified HP-IB interface, and it can only be sent by the 9826 when it is the active controller. This procedure is in module HPIB_2.

Examples

```
LOCAL_LOCKOUT (7) ;
```

The Local Lockout message is cleared when the Local message is sent by executing the LOCAL procedure. However, executing the ABORT_HPIB procedure does not cancel the Local Lockout message.

Enabling Local Control

During system operation, it may be necessary for an operator to interact with one or more devices. For instance, an operator might need to work from the front panel to make special tests or to troubleshoot. And, in general, it is good systems practice to return all devices to local control upon conclusion of remote-control operations. Executing the LOCAL procedure returns the specified devices to local (front-panel) control. The 9826 must be the active controller to send the LOCAL message. This procedure is in module HPIB_2

Examples

```
LOCAL (7) ;

LOCAL (801) ;
```

If primary addressing is specified, the Go-to-Local message is sent only to the specified device(s). However, if only the interface select code is specified, the Local message is sent to all devices on the specified HP-IB interface and any previous Local Lockout message (which is still in effect) is automatically cleared. The 9826 must be the system controller to send the Local message (by specifying only the interface select code).

Triggering HP-IB Devices

The TRIGGER procedure sends a Trigger message from the controller to a selected device or group of devices. The purpose of the Trigger message is to initiate some device-dependent action; for example, it can be used to trigger a digital voltmeter to perform its measurement cycle. Because the response of a device to a Trigger Message is strictly device-dependent, neither the Trigger message nor the interface indicates what action is initiated by the device. This procedure is in module HPIB_2.

Examples

```
TRIGGER (7) ;

TRIGGER (707);
```

Specifying only the interface select code outputs a Trigger message to all devices currently addressed to listen on the bus. Including device addresses in the statement triggers only those devices addressed by the statement.

Clearing HP-IB Devices

The CLEAR procedure provides a means of "initializing" a device to its predefined, device-dependent state. When the CLEAR procedure is executed, the Clear message is sent either to all devices or to the specified device, depending on the information contained within the device selector. If only the interface select code is specified, all devices on the specified HP-IB interface are cleared. If primary-address information is specified, the Clear message is sent only to the specified device. Only the active controller can send the Clear message. This procedure is in module HPIB_2.

Examples

```
CLEAR (7) ;

CLEAR (700) ;
```

Aborting Bus Activity

The ABORT_HPIB procedure may be used to terminate all activity on the bus and return all the HP-IB interfaces of all devices to a reset (or power-on) condition. Whether this affects other modes of the device depends on the device itself. The 9826 must be either the active or the system controller to perform this function. If the system controller (which is not the current active controller) executes this statement, it regains active control of the bus. This procedure is in module HPIB_2. **Only the interface select code may be specified**; device selectors which contain primary-addressing information (such as 724) may not be used. This procedure is in module HPIB_2.

Examples

```
ABORT_HPIB (7) ;
```

Passing Control

The PASS_CONTROL procedure will pass current active control to another device on the bus. The interface must be active controller. This procedure is in module HPIB_2.

Examples

```
PASS_CONTROL (720) ;
```

Polling HP-IB Devices

The parallel poll is the fastest means of gathering device status when several devices are connected to the bus. Each device (with this capability) can be programmed to respond with one bit of status when parallel polled, making it possible to obtain the status of several devices in one operation. If a device responds affirmatively to a parallel poll, more information as to its specific status can be obtained by conducting a serial poll of the device.

Configuring Parallel Poll Responses

Certain devices can be remotely programmed by the active controller to respond to a parallel poll. A device which is currently configured for a parallel poll responds to the poll by placing its current status on one of the bus data lines. The logic sense of the response and the data-bit number can be programmed by the PPOLL_CONFIGURE procedure. If more than one device is to respond on a single bit, each device must be configured with a separate PPOLL_CONFIGURE procedure. This procedure is in module HPIB_2.

Note

Use of PPOLL_CONFIGURE may interfere with the Pascal Operating System, especially if an external disk is being used. **Be very careful.**

Example

```
PPOLL_CONFIGURE (705,mask) ;
```

The value of the mask (any numeric expression can be specified) is first rounded and then used to configure the device's parallel response. The least significant 3 bits (bits 0 through 2) of the expression are used to determine which data line the device is to respond on (place its status on). Bit 3 specifies the "true" state of the parallel poll response bit of the device. A value of 0 implies that the device's response is 0 when its status-bit message is true.

Example

The following statement configures device at address 01 on interface select code 7 to respond by placing a 0 on bit 4 when its status response is "true".

```
PPOLL_CONFIGURE (701,4) ;
```

Conducting a Parallel Poll

The PPOLL function returns a single byte containing up to 8 status bit messages of all devices on the bus capable of responding to the poll. Each bit returned by the function corresponds to the status bit of the device(s) configured to respond to the parallel poll. (Recall that one or more devices can respond on a single line.) The PPOLL function can only be executed by the 9826 when it is the active controller. This function is in module HPIB_3.

Example

```
Response:=PPOLL(7) ;
```

Disabling Parallel Poll Responses

The PPOLL_UNCONFIGURE procedure gives the 9826 (as active controller) the capability of disabling the parallel poll responses of one or more devices on the bus.

Note

Use of PPOLL_UNCONFIGURE may interfere with the Pascal Operating System, especially if an external disk is being used. **Be very careful.**

Examples

The following statement disables device 5 only.

```
PPOLL_UNCONFIGURE (705) ;
```

This statement disables all devices on interface select code 8 from responding to a parallel poll.

```
PPOLL_UNCONFIGURE (8) ;
```

If no primary addressing is specified, all bus devices are disabled from responding to a parallel poll. If primary addressing is specified, only the specified devices (which have the parallel poll configure capability) are disabled.

Conducting a Serial Poll

A sequential poll of individual devices on the bus is known as a serial poll. One entire byte of status is returned by the specified device in response to a serial poll. This byte is called the Status Byte message and, depending on the device, may indicate an overload, a request for service, or a printer being out of paper. The particular response of each device depends on the device.

The SPOLL function performs a serial poll of the specified device; the 9826 must be the active controller. This function is in module HPIB_3.

Examples

```
Response:=SPOLL(724) ;
```

HP-IB Interface Conditions

The HP-IB interface can be in various states at various times. It is desirable for the programmer to know about this state information. The major conditions of interest are:

- Is a device requesting service?
- Am I a talker?
- Am I a listener?
- What remote/local state am I in?

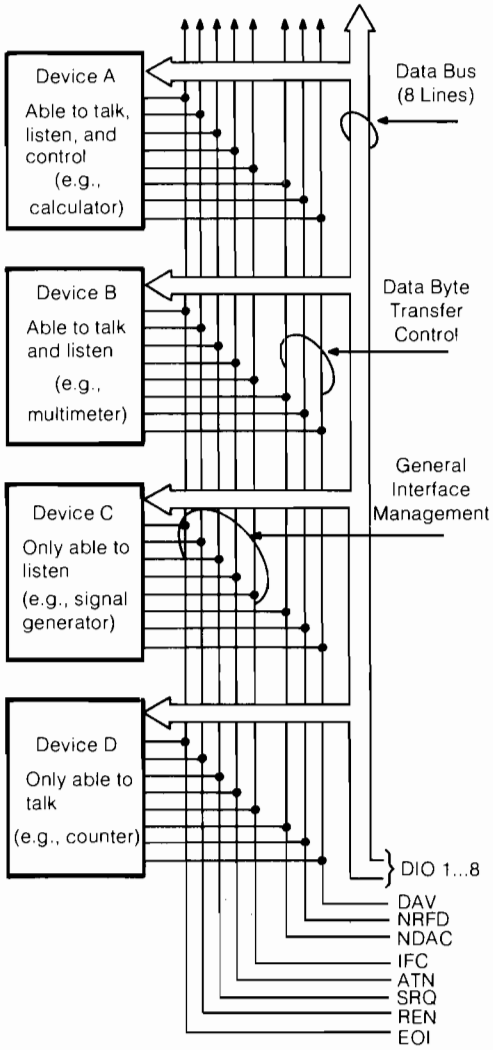
These conditions are supported by the following I/O Library functions in the HPIB_3 module. All of these functions are boolean functions and will return an appropriate TRUE or FALSE indication depending of the condition state.

function		meaning
REQUESTED	(interface_select_code)	Is SRQ asserted?
TALKER	(interface_select_code)	Am I a talker?
LISTENER	(interface_select_code)	Am I a listener?
REMOTED	(interface_select_code)	Is REN asserted?
LOCKED_OUT	(interface_select_code)	Am I in a locked out state?

The REQUESTED function requires that the interface be active controller. The REMOTED function requires that the interface not be system controller. The LOCKED_OUT function requires that the interface not be active controller. An example program segment follows.

```
WHILE REQUESTED(isc) DO
  FOR i:=0 TO 7 DO BEGIN
    IF BIT_SET(SPOLL(isc*100+i),6)
      THEN WRITELN('device ',i:2,' requesting service ');
  END; { of FOR DO BEGIN }
```

HP-IB Control Lines



Handshake Lines

The preceding figure shows the names given to the eight control lines that make up the HP-IB. Three of these lines are designated as the "handshake" lines and are used to control the timing of data byte exchanges so that the talker does not get ahead of the listener(s). The three handshake lines are as follows.

DAV	Data Valid
NRFD	Not Ready for Data
NDAC	Not Data Accepted

The **HP-IB interlocking handshake** uses the lines as follows. All devices currently designated as active listeners would indicate when they are ready for data by using the NRFD line. A device not ready would pull this line low (true) to signal that it is not ready for data, while any device that is ready would let the line float high. Since an active low overrides a passive high, this line will stay low until all active listeners are ready for data.

When the talker senses that all devices are ready, it places the next data byte on the data lines and then pulls DAV low (true). This tells the listeners that the information on the data lines is valid and that they may read it. Each listener then accepts the data and lets the NDAC line float high (false). As with NRFD, only when all listeners have let NDAC go high will the talker sense that all listeners have read the data. It can then float DAV (let it go high) and start the entire sequence over again for the next byte of data.

The Attention Line (ATN)

Command messages are encoded on the data lines as 7-bit ASCII characters, and are distinguished from normal data characters by the logic state of the attention line (ATN). That is, when ATN is **false**, the states of the data lines are interpreted as **data**. When ATN is **true**, the data lines are interpreted as **commands**. The set of 128 ASCII characters that can be placed on the data lines during this ATN-true mode are divided into four classes by the states of data lines DIO6 and DIO7. These classes of commands are shown in a table in the section called "Advanced Bus Management".

The Interface Clear Line (IFC)

Only the system controller can set the IFC line true. By asserting IFC, all bus activity is unconditionally terminated, the system controller regains the capability of active controller (if it has been passed to another device), and any current talker and listeners become unaddressed. Normally, this line is only used to terminate all current operations, or to allow the system controller to regain control of the bus. It overrides any other activity that is currently taking place on the bus.

The Remote Enable Line (REN)

This line is used to allow instruments on the bus to be programmed remotely by the active controller. Any device that is addressed to listen while REN is true is placed in the Remote mode of operation.

The End or Identify Line (EOI)

Normally, data messages sent over the HP-IB are sent using the standard ASCII code and are terminated by the ASCII line-feed character, CHR(10). However, certain devices may wish to send blocks of information that contain data bytes which have the bit pattern of the line-feed character but which are actually part of the data message. Thus, no bit pattern can be designated as a terminating character, since it could occur anywhere in the data stream. For this reason, the EOI line is used to mark the end of the data message.

The EOI line is not directly supported by the input and output procedures of Chapters 7 and 8. It is supported in advanced transfers by the TRANSFER_END procedure.

The I/O Library does provide access to the EOI line at a lower level. The state of the EOI line after the last byte read is stored in the system and can be viewed with the END_SET boolean function which is module HPIB_1. An example of this function is:

```
UNLISTEN(7);
TALK(7,20);
LISTEN(7,MY_ADDRESS(7));
REPEAT
  READCHAR(7,c[i]);
UNTIL END_SET(7);
```

The I/O Library also provides a facility for setting the EOI line with a byte to be sent. This is provided with the procedure SET_HPIB which is in module HPIB_0. An example use of this procedure is:

```
UNLISTEN(7);
TALK(7,MY_ADDRESS(7));
LISTEN(7,11);
FOR i:=1 TO STRLEN(str)-1 DO WRITECHAR(7,str[i]);
SET_HPIB(7,EOI_LINE);
WRITECHAR(7,str[STRLEN]);
```

After the character output occurs, the EOI line will be set false automatically.

The Service Request Line (SRQ)

The active controller is always in charge of the order of events that occur on the HP-IB. If a device on the bus needs the controller's help, it can set the service request line true. This line sends a request, not a demand, and it is up to the controller to choose when and how it will service that device. The REQUESTED function tells the controller whether it is being requested. The procedure to request the service is the REQUEST_SERVICE procedure in the module HPIB_3. This module is of the form:

```
REQUEST_SERVICE ( interface_select_code , response_byte );
```

The response byte is an integer value in the range of 0 through 255. If bit 6 of this byte is set, the SRQ line will be asserted by this interface. If bit 6 is not set, then this device will not assert the SRQ line. The interface must not be active controller to request service.

Determining Bus-Line States

IOSTATUS register 7 contains the current states of all bus hardware lines. Reading this register returns the states of these lines.

```
bus_lines := IOSTATUS(7,7);
```

Status Register 7

Bus Control and Data Lines

Most significant Bit — Least Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ATN True	DAV True	NDAC* True	NRFD* True	EOI True	SRQ** True	IFC True	REN True
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

* Only if addressed to TALK, else not valid.
* * Only if Active Controller, else not valid.

Note

Due to the way the bi-directional buffers work, NDAC and NRFD are not accurately read by this IOSTATUS function unless the interface is currently addressed to talk. Also, SRQ is not accurately shown unless the interface is currently the active controller.

Advanced Bus Management

Bus communication involves both sending data to devices and sending commands to devices and the interface itself. "General Structure of the HP-IB" stated that this communication must be made in an orderly fashion and presented a brief sketch of the differences between data and commands. However, most of the bus operations described so far in this chapter involve sequences of commands and/or data which are sent automatically by the computer when HP-IB statements are executed. This section describes both the commands and data sent by HP-IB statements and how to construct your own, custom bus sequences.

The Message Concept

The main purpose of the bus is to send information between two (or more) devices. These quantities of information sent from talker to listener(s) can be thought of as messages. However, before data can be sent through the bus, it must be properly configured. A sequence of commands is generally sent before the data to inform bus devices which is to send and which is (or are) to listen to the subsequent message(s). These commands can also be thought of as messages.

Most bus messages are transmitted by sending a byte (or sequence of bytes) with numeric values of 0 through 255 through the bus data lines. When the Attention line (ATN) is true, these bytes are considered commands; when ATN is false, they are interpreted as data. Bus command groups and their ASCII characters and codes are shown in "Bus Commands and Codes".

Types of Bus Messages

The messages can be classified into twelve types. This computer is capable of implementing all twelve types of interface messages. The following list describes each type of message.

1. A Data message consists of information which is sent from the talker to the listener(s) through the bus data lines.
2. The Trigger message causes the listening device(s) to initiate device-dependent action(s).
3. The Clear message causes either the listening device(s) or all of the devices on the bus to return to their device-dependent "clear" states.
4. The Remote message causes listening devices to change to remote program control when addressed to listen.
5. The Local message clears the Remote message from the listening device(s) and returns the device(s) to local front-panel control.
6. The Local Lockout message disables a device's front-panel controls, preventing a device's operator from manually interfering with remote program control.
7. The Clear Lockout/Local message causes all devices on the bus to be removed from Local Lockout and to revert to the Local state. This message also clears the Remote message from all devices on the bus.
8. The Service Request message can be sent by a device at any time to signify that the device needs to interact with the the active controller. This message is cleared by sending the device's Status Byte message, if the device no longer requires service.

9. A Status Byte message is a byte that represents the status of a single device on the bus. This byte is sent in response to a serial poll performed by the active controller. Bit 6 indicates whether the device is sending the Service Request message, and the remaining bits indicate other operational conditions of the device.
10. A Status Bit message is a single bit of device-dependent status. Since more than one device can respond on the same line, this Status Bit may be logically combined and/or concatenated with Status Bit messages from many devices. Status Bit messages are returned in response to a parallel poll conducted by the active controller.
11. The Pass Control message transfers the bus management responsibilities from the active controller to another controller.
12. The Abort message is sent by the system controller to assume control of the bus unconditionally from the active controller. This message terminates all bus communications, but is not the same as the Clear message.

These messages represent the full implementation of all HP-IB system capabilities; all of these messages can be sent by this computer. However, each device in a system may be designed to use only the messages that are applicable to its purpose in the system. It is important for you to be aware of the HP-IB functions implemented on each device in your HP-IB system to ensure its operational compatibility with your system.

Bus Commands and Codes

The table below shows the decimal values of IEEE-488 command messages. Remember that **ATN is true** during all of these commands. Notice also that these commands are separated into four general categories: Primary Command Group, Listen Address Group, Talk Address Group, and Secondary Command Group. Subsequent discussions further describe these commands.

Decimal Value	ASCII Character	Interface Message	Description
		PCG	**Primary Command Group**
1	SOH	GTL	Go to Local
4	EOT	SDC	Selected Device Clear
5	ENQ	PPC	Parallel Poll Configure
8	BS	GET	Group Execute Trigger
9	HT	TCT	Take Control
17	DC1	LLO	Local Lockout
20	DC4	DCL	Device Clear
21	NAK	PPU	Parallel Poll Unconfigure
24	CAN	SPE	Serial Poll Enable
25	EM	SPD	Serial Poll Disable
		LAG	**Listen Address Group**
32-62	Space through > (Numbers & Special Chars.)		Listen Addresses 0 through 30
63	?	UNL	Unlisten
		TAG	**Talk Address Group**
64-94	@ through ↑ (Uppercase ASCII)		Talk Addresses 0 through 30
95	_ (underscore)	UNT	Untalk
		SCG	**Secondary Command Group**
96-126	` through ~ (Lowercase ASCII)		Secondary Commands 0 through 30
127	DEL		Ignored

Address Commands and Codes

The following table shows the ASCII characters and corresponding codes of the Listen Address Group and Talk Address Group commands. The next section describes how to send these commands.

Address Characters		Address Code	Address Switch Settings				
Listen	**Talk**	**Decimal**	**(5)**	**(4)**	**(3)**	**(2)**	**(1)**
Space	@	0	0	0	0	0	0
!	A	1	0	0	0	0	1
"	B	2	0	0	0	1	0
#	C	3	0	0	0	1	1
$	D	4	0	0	1	0	0
%	E	5	0	0	1	0	1
&	F	6	0	0	1	1	0
'	G	7	0	0	1	1	1
(	H	8	0	1	0	0	0
)	I	9	0	1	0	0	1
*	J	10	0	1	0	1	0
+	K	11	0	1	0	1	1
,	L	12	0	1	1	0	0
–	M	13	0	1	1	0	1
.	N	14	0	1	1	1	0
/	O	15	0	1	1	1	1
0	P	16	1	0	0	0	0
1	Q	17	1	0	0	0	1
2	R	18	1	0	0	1	0
3	S	19	1	0	0	1	1
4	T	20	1	0	1	0	0
5	U	21	1	0	1	0	1
6	V	22	1	0	1	1	0
7	W	23	1	0	1	1	1
8	X	24	1	1	0	0	0
9	Y	25	1	1	0	0	1
:	Z	26	1	1	0	1	0
;	[	27	1	1	0	1	1
<	/	28	1	1	1	0	0
=	]	29	1	1	1	0	1
>	↑	30	1	1	1	1	0

Explicit Bus Messages

Any "ATN" command can be sent in any order with a procedure called SEND_COMMAND. This procedure will send the specified command on the bus. The interface must be active controller. The form of the procedure is:

```
SEND_COMMAND ( interface_select_code , command_character );
```

The command character is a normal character expression in the range of CHR(0) through CHR(255). You should be very careful when using this procedure because you can put devices into bad or unknown states. The procedure is in module HPIB_1.

Example

```
SEND_COMMAND(7,'?');     { send unlisten }
SEND_COMMAND(7,'_');     { send untalk   }
SEND_COMMAND(7,'!');     { send dvc 01 listen }
SEND_COMMAND(7,'U');     { send dvc 21 talk   }
```

Summary of HP-IB IOSTATUS and IOCONTROL Registers

Status Register 0 — Card Identification

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 0 — Interface Reset

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Any Bit Will Reset Interface							
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 1 — Interrupt and DMA Status

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrrupts Enabled	Interrupt Requested	Interrupt Level		0	0	DMA Channel 1 Enabled	DMA Channel 0 Enabled
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 1 — Serial Poll Response Byte

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device Dependent Status	SRQ 1 = I did it 0 = I didn't	Device Dependent Status					
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 2 — **Parallel Poll Response Byte**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8 1 = True	DIO7 1 = True	DIO6 1 = True	DIO5 1 = True	DIO4 1 = True	DIO3 1 = True	DIO2 1 = True	DIO1 1 = True
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 3 — **Controller Status and Address**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary Address of Interface				
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 3 — **Set My Address**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			Primary Address				
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 4 — Interrupt Status

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/ Local Change	Talker/ Listener Address Change
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 5 — Interrupt Enable Mask

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/ Local Change	Talker/ Listener Address Change
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 6 — Interface Status

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
REM	LLO	ATN True	LPAS	TPAS	LADS	TADS	*
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary Address of Interface				
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

* Least-significant bit of last address recognized

Status Register 7 — Bus Control and Data Lines

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ATN True	DAV True	NDAC* True	NRFD* True	EOI True	SRQ** True	IFC True	REN True
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

* Only if addressed to TALK, else not valid.
** Only if Active Controller, else not valid.

Status Register 8 — Unrecognized Command

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Summary of HP-IB IOREAD_BYTE and IOWRITE_BYTE Registers

IOREAD Registers

Register 1 — Card Identification
Register 3 — Interrupt and DMA Status
Register 5 — Controller Status and Address
Register 17 — Interrupt Status 0[1]
Register 19 — Interrupt Status 1[1]
Register 21 — Interface Status
Register 23 — Control-Line Status
Register 29 — Command Pass-Through
Register 31 — Data-Line Status[1]

HP IOREAD_BYTE Register 1 — Card Identification

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Future Use Jumper Installed	0	0	0	0	0	0	1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Bit 7 is set (1) if the "future use" jumper is installed and clear (0) if not.

Bits 6 through 0 constitute a card identification code (= 1 for all HP-IB cards).

Note

This register is only implemented on external HP-IB cards. The internal HP-IB, at interface select code 7, "floats" this register (i.e., the states of all bits are indeterminate).

HP-IB IOREAD_BYTE Register 3 — Interrupt and DMA Status

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupt Enabled	Interrupt Request	Interrupt Level		X	X	DMA1	DMA0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

1 Indicates that an IOREAD_BYTE operation will change the state of the interface.

Bit 7 is set (1) if interrupts are currently enabled.

Bit 6 is set (1) when the card is currently requesting service.

Bits 5 and 4 constitute the card's hardware interrupt level (a switch setting on all external cards, but fixed at level 3 on the internal HP-IB).

Bit 5	Bit 4	Hardware Interrupt Level
0	0	3
0	1	4
1	0	5
1	1	6

Bits 3 and 2 are not used (indeterminate).

Bit 1 is set (1) if DMA channel one is currently enabled.

Bit 0 is set (1) if DMA channel zero is currently enabled.

Note

Bits 7, 5, 4, 3, 2, and 1 are not implemented on the internal HP-IB (interface select code 7).

HP-IB IOREAD_BYTE Register 5 **Controller Status and Address**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Not Active Controller	X	← HP-IB Primary Address of Interface → (MSB)				(LSB)
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Bit 7 is set (1) if the interface is the System Controller.

Bit 6 is set (1) if the interface is **not** the current Active Controller and clear (0) if it **is** the Active Controller.

Bit 5 is not used.

Bits 4 through 0 contain the card's Primary Address switch setting. The following bit patterns indicate the specified addresses.

Bit 4 3 2 1 0	Primary Address
0 0 0 0 0	0
0 0 0 0 1	1
. . .	. . .
1 1 1 0 1	29
1 1 1 1 0	30
1 1 1 1 1	(not allowed)

Note

Bits 5 through 0 are not implemented on the internal HP-IB.

HP-IB IOREAD_BYTE Register 17 **MSB of Interrupt Status**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MSB Interrupt	LSB Interrupt	Byte Received	Ready for Next Byte	End Detected	SPAS	Remote/ Local Change	My Address Change
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Bit 7 set (1) indicates that an interrupt has occurred whose cause can be determined by reading the contents of this register.

Bit 6 set (1) indicates that an interrupt has occurred whose cause can be determined by reading Interrupt Status Register 1 (IOREAD_BYTE Register 19).

Bit 5 set (1) indicates that a data byte has been received.

Bit 4 set (1) indicates that this interface is ready to accept the next data byte.

Bit 3 set (1) indicates that an End (EOI with ATN = 0) has been detected.

Bit 2 set (1) indicates that the Serial-Poll-Active State has been entered.

Bit 1 set (1) indicates that a Remote/Local State change has occurred.

Bit 0 set (1) indicates that a change in My Address has occurred.

HP-IB IOREAD_BYTE Register 19 **LSB of Interrupt Status**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Command Group	Secondary Command While Addressed	Clear Received	My Address Received (MLA or MTA)	SRQ Received	IFC Received
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Bit 7 set (1) indicates that a Group Execute Trigger command has been received.

Bit 6 set (1) indicates that an Incomplete-Source-Handshake error has occurred.

Bit 5 set (1) indicates that an unidentified command has been received.

Bit 4 set (1) indicates that a Secondary Address has been sent in while in the extended-addressing mode.

Bit 3 set (1) indicates that the interface has entered the Device-Clear-Active State.

Bit 2 set (1) indicates that My Address has been received.

Bit 1 set (1) indicates that a Service Request has been received.

Bit 0 set (1) indicates that the Inteface Clear message has been received.

HP-IB IOREAD_BYTE Register 21 **Interface Status**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
REM	LLO	ATN True	LPAS	TPAS	LADS	TADS	LSB of Last Address
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Bit 7 set (1) indicates that this Interface is in the Remote State.

Bit 6 set (1) indicates that this interface is in the Local Lockout State.

Bit 5 set (1) indicates that the ATN signal line is true.

Bit 4 set (1) indicates that this interface is in the Listener-Primary-Addressed State.

Bit 3 set (1) indicates that this interface is in the Talker-Primary-Addressed State.

Bit 2 set (1) indicates that this interface is in the Listener-Addressed State.

Bit 1 set (1) indicates that this interface is in the Talker-Addressed State.

Bit 0 set (1) indicates that this is the least-significant bit of the last address recognized by this interface.

HP-IB IOREAD_BYTE Register 23 — Control-Line Status

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ATN True	DAV True	NDAC* True	NRFD* True	EOI True	SRQ** True	IFC True	REN True
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

*Only if addressed to TALK, else not valid.
**Only if Active Controller, else not valid.

A set bit (1) indicates that the corresponding line is currently true; a 0 indicates that the line is currently false.

HP-IB IOREAD_BYTE Register 29 — Command Pass-Through

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

This register can be read during a bus holdoff to determine which Secondary Command has been detected.

HP-IB IOREAD_BYTE Register 31 — Bus Data Lines

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

A set bit (1) indicates that the corresponding HP-IB data line is currently true; a 0 indicates the line is currently false.

HP-IB IOWRITE_BYTE Registers

Register 3 — Interrupt Enable
Register 17 — MSB of Interrupt Mask
Register 19 — LSB of Interrupt Mask
Register 23 — Auxiliary Command Register
Register 25 — Address Register
Register 27 — Serial Poll Response
Register 29 — Parallel Poll Response
Register 31 — Data Out Register

HP-IB IOWRITE_BYTE Register 3 — Interrupt Enable

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable Interrupt	X	X	X	X	X	Enable Channel 1	Enable Channel 0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Bit 7 enables interrupts from this interface if set (1) and disables interrupts if clear (0).

Bits 6 through 2 are "don't cares" (i.e., their values have no effect on the interface's operation).

Bit 1 enables DMA channel 1 if set (1) and disables if clear (0).

Bit 0 enables DMA channel 0 if set (1) and disables if clear (0).

Note

Bits 7 through 1 are not implemented on the internal HP-IB interface and thus have no effect on the interface's operation.

IOWRITE_BYTE Register 17 — MSB of Interrupt Mask

Setting a bit of this register enables an interrupt for the specified condition. The bit assignments are the same as for the MSB of Interrupt Status Register (IOREAD Register 17), except that bits 7 and 6 are not used.

IOWRITE_BYTE Register 19 — LSB of Interrupt Mask

Setting a bit of this register enables an interrupt for the specified condition. The bit assignments are the same as for the LSB of Interrupt Status Register (IOREAD Register 19).

HP-IB IOWRITE_BYTE Register 23 **Auxiliary Command Register**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Set	X	X	Auxiliary Command Function				
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Bit 7 is set (1) for a Set operation and clear (0) for a Clear operation.

Bits 6 and 5 are "don't cares".

Bits 4 through 0 are Auxiliary-Command-Function-Select bits. The following commands can be sent to the interface by sending the specified numeric values.

Decimal Value		Description of Auxiliary Command
0	—	Clear Chip Reset.
128	—	Set Chip Reset.
1	—	Release ACDS holdoff. If Address Pass Through is set, it indicates an invalid secondary has been received.
129	—	Release ACDS holdoff; If Address Pass Through is set, indicates a valid secondary has been received.
2	—	Release RFD holdoff.
130	—	Same command as decimal 2 (above).
3	—	Clear holdoff on all data.
131	—	Set holdoff on all data.
4	—	Clear holdoff on EOI only.
132	—	Set holdoff on EOI only.
5	—	Set New Byte Available (nba) false.
133	—	Same command as decimal 5 (above).
6	—	Pulse the Group Execute Trigger line, or clear the line if it was set by decimal command 134.
134	—	Set Group Execute Trigger line.
7	—	Clear Return To Local (rtl).
135	—	Set Return To Local (must be cleared before the device is able to enter the Remote state).
8	—	Causes EOI to be sent with the next data byte.
136	—	Same command as decimal 8 (above).
9	—	Clear Listener State (also cleared by decimal 138).
137	—	Set Listener State.
10	—	Clear Talker State (also cleared by decimal 137).
138	—	Set Talker State.

(Continued)

Decimal Value		Description of Auxiliary Command
11	—	Go To Standby (gts; controller sets ATN false).
139	—	Same command as decimal 11 (above).
12	—	Take Control Asynchronously (tca; ATN true).
140	—	Same command as decimal 12 (above).
13	—	Take Control Synchronously (tcs; ATN true).
141	—	Same command as decimal 13 (above).
14	—	Clear Parallel Poll.
142	—	Set Parallel Poll (read Command-Pass-Through register before clearing).
15	—	Clear the Interface Clear line (IFC).
143	—	Set Interface Clear (IFC maintained >100 μs).
16	—	Clear the Remote Enable (REN) line.
144	—	Set Remote Enable.
17	—	Request control (after TCT is decoded, issue this to wait for ATN to drop and receive control).
145	—	Same command as decimal 17 (above).
18	—	Release control (issued after sending TCT to complete a Pass Control and set ATN false).
146	—	Same command as decimal 18 (above).
19	—	Enable all interrupts.
147	—	Disable all interrupts.
20	—	Pass Through next Secondary Command.
148	—	Same command as decimal 20 (above).
21	—	Set T1 delay to 10 clock cycles (2 μs at 5 MHz).
149	—	Set T1 delay to 6 clock cycles (1.2 μs at 5 MHz).
22	—	Clear Shadow Handshake.
150	—	Set Shadow Handshake.

HP-IB IOWRITE_BYTE Register 25 **Address Register**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable Dual Addressing	Disable Listen	Disable Talker	Primary Address				
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Bit 7 set (1) enables the Dual-Primary-Addressing Mode.

Bit 6 set (1) invokes the Disable-Listen function.

Bit 5 set (1) invokes the Disable-Talker function

Bits 4 through 0 set the device's Primary Address (same address bit definitions as READIO Register 5).

HP-IB IOWRITE_BYTE Register 27 **Serial Poll Response Byte**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device Dependent Status	Request Service	Device-Dependent Status					
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Bits 7 and 5—0 specify the Device-Dependent Status.

Bit 6 sends an SRQ if set (1).

Note

Given an unknown state of the Serial Poll Response Byte, it is necessary to write the byte with bit 6 set to zero followed by a write of the byte with bit 6 set to the desired final value. This will insure that a SRQ will be generated if one was desired.

HP-IB IOWRITE_BYTE Register 29 **Parallel Poll Response**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

A 1 sets the appropriate bit true during a Parallel Poll; a 0 sets the corresponding bit false. Initially, and when Parallel Poll is not configured, this register must be set to all zeros.

HP-IB IOWRITE_BYTE Register 31 **Data-Out Register**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Summary of Bus Sequences

The following tables show the bus activity invoked by executing HP-IB statements and functions. The mnemonics used in these tables were defined in the previous section of this chapter.

Note that the bus messages are sent by using single lines (such as the ATN line) and multi-line commands (such as DCL). The information shows the state of and changes in the state of the ATN line during these bus sequences. The tables implicitly show that these **changes in the state of ATN remain in effect unless another change is explicitly shown in the table**. For example, if a statement sets ATN (true) with a particular command, it remains true unless the table explicitly shows that it is set false (ATN). The ATN line is implememted in this manner to avoid unnecessary transitions in this signal whenever possible. It should not cause any dilemmas in most cases.

ABORT_HPIB

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	IFC (duration ≥100μsec) REN ATN	Error	ATN MTA UNL ATN	Error
Not Active Controller	IFC (duration ≥100 μsec)* REN ATN		No Action	

* The IFC message allows a non-active controller (which is the system controller) to become the active controller.

CLEAR

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN DCL	ATN MTA UNL LAG SDC	ATN DCL	ATN MTA UNL LAG SDC
Not Active Controller	Error			

LOCAL

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	$\overline{REN}$ $\overline{ATN}$	ATN MTA UNL LAG GTL	ATN GTL	ATN MTA UNL LAG GTL
Not Active Controller	$\overline{REN}$	Error	Error	

LOCAL_LOCKOUT

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN LLO	Error	ATN LLO	Error
Not Active Controller	Error			

PASS_CONTROL

	System Controller		Net System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN TCT ATN	ATN UNL TAG TCT ATN	ATN TCT ATN	ATN UNL TAG TCT ATN
Not Active Controller	Error			

PPOLL

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN & EOI (duration≥25μs) Read byte $\overline{EOI}$ Restore ATN to previous state	Error	ATN & EOI (duration≥25μs) Read byte $\overline{EOI}$ Restore ATN to previous state	Error
Not Active Controller	Error			

PPOLL_CONFIGURE

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	Error	ATN MTA UNL LAG PPC PPE	Error	ATN MTA UNL LAG PPC PPE
Not Active Controller	Error			

PPOLL_UNCONFIGURE

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN PPU	ATN MTA UNL LAG PPC PPD	ATN PPU	ATN MTA UNL LAG PPC PPD
Not Active Controller	Error			

REMOTE

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	REN $\overline{\text{ATN}}$	REN ATN MTA UNL LAG	Error	
Not Active Controller	REN	Error	Error	

SPOLL

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	Error	ATN UNL MLA TAD SPE $\overline{\text{ATN}}$ Read data ATN SPD UNT	Error	ATN UNL MLA TAD SPE $\overline{\text{ATN}}$ Read data ATN SPD UNT
Not Active Controller	Error			

TRIGGER

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN GET	ATN UNL LAG GET	ATN GET	ATN MTA UNL LAG GET
Not Active Controller	Error			

Chapter 13
The Datacomm Interface

Introduction

The HP 98628 Data Communications Interface enables your desktop computer to communicate with any device that is compatible with standard asynchronous or HP Data Link data communication protocols. Devices can include various modems or link adapters, as well as equipment with standard RS-232C or current loop links.

This chapter discusses both asynchronous and Data Link protocols, and programming techniques. Subject areas that are similar for both protocols are combined, while information that is unique to one protocol or the other is separated according to application.

Prerequisites

It is assumed that you are familiar with the information presented in Data Communication Basics (98046-90005), and that you understand data communication hardware well enough to determine your needs when configuring the datacomm link. Configuration parameters include such items as half/full duplex, handshake, and timeout requirements. If you have any questions concerning equipment installation or interconnection, consult the appropriate interface or adapter installation manuals.

The datacomm interface supports several cable and adapter options. They include:

- RS-232C Interface cable and connector wired for operation with data communication equipment (male cable connector) or with data terminal equipment (female cable connector).
- HP 13264A Data Link Adapter for use in HP 1000- or HP 3000-based Data Link network applications
- HP 13265A Modem for asynchronous connections up to 300 baud, including built-in autodial capability[1].
- HP 13266A Current Loop Adapter for use with current loop links or devices.

Some of the information contained in this chapter pertains directly to certain of these devices in specific applications.

1 The HP 13265A modem is compatible with Bell 103 and Bell 113 Modems, and is approved for use in the USA and Canada. Most other countries do not allow use of user-owned modems. Contact your local HP Sales and Service office for information about local regulations.

Before you begin datacomm operation, be sure all interfaces, cables, connectors, and equipment have been properly plugged in. Power must be on for all devices that are to be used. Consult applicable installation manuals if necessary.

Protocol

Two protocols are switch selectable on the datacomm interface. They are also software selectable during normal program operation. The switch setting on the interface determines the default protocol when the computer is first powered up. Protocol is changed between Async and Data Link during program operation by selecting the new protocol, waiting for the message to reach the card, then resetting the card. The exact procedure is explained in the IOCONTROL register operations section of this chapter.

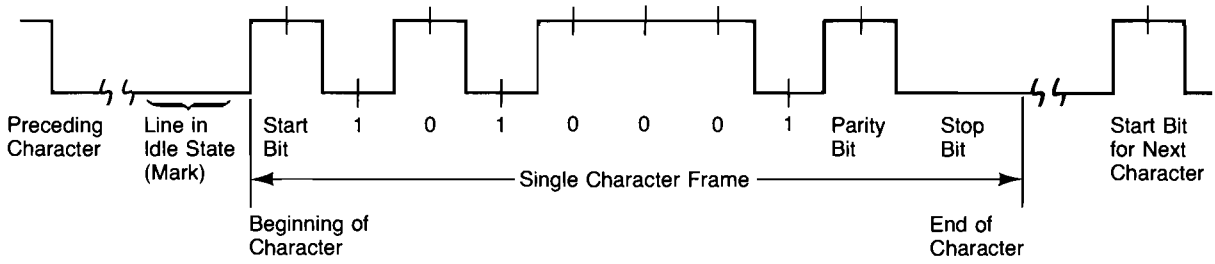
Asynchronous Communication Protocol

Asynchronous data communication is the most widely used protocol, especially in applications where high data integrity is not mandatory. Data is transmitted, one character at a time, with each character being treated as an individual message. Start and stop bits are used to maintain timing coordination between the receiver and transmitter. A parity bit is sometimes included to detect character transmission errors.Asynchronous character format is as follows: Each character consists of a start bit, 5 to 8 data bits, an optional parity bit, and 1, 1.5, or 2 stop bits, with an optional time gap before the beginning of the next character. The total time from the beginning of one start bit to the beginning of the next is called a character frame.

Parity options include:

- NONE No parity bit is included.
- ODD Parity set if EVEN number of "1"s in character bits.
- EVEN Parity set if ODD number of "1"s in character bits.
- ONE Parity bit is set for all characters.
- ZERO Parity bit is zero for all characters.

Here is a simple diagram showing the structure of an asynchronous character and its relationship to previous and succeeding characters:

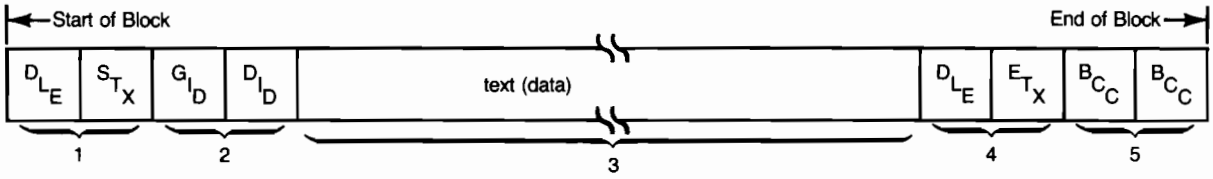


Data Link Communication Protocol

Data Link protocol overcomes the data integrity limitations of Async by handling data in blocks. Each block is transmitted as a stream of individual asynchronous characters, but protocol control characters and block check characters are also transmitted with the data. The receiver uses the protocol control characters to determine block boundaries and data format. Block check characters are used to detect transmission errors. If an error occurs, the block is retransmitted until it is successfully received. Block protocol and format is similar to Binary Synchronous Communication (BSC or Bisync, for short).

Data Link protocol provides for two transmission modes: Transparent, and Normal. In transparent mode, any data format can be transferred because datacomm control characters are preceded by a DLE character. If a control character is sent without an accompanying DLE, it is treated as data. When normal mode is used, only ASCII data can be sent, and datacomm control characters are not allowed in the data stream. The HP 1000 and HP 3000 computers usually transmit in transparent mode. All transmissions from your desktop computer are sent as transparent data. If your application involves non-ASCII data transfers (discussed later in this chapter), be sure the HP 1000 or HP 3000 network host is using transparent mode for all transmissions to your computer.

Each data block sent to the network host by the datacomm interface is structured as follows:



1. The "start transmission" control characters identify the beginning of valid data. If a DLE is present, the data is transparent; If absent, data is normal. All data from your desktop computer is transparent.
2. The terminal identification characters are included in blocks sent to the network host. Blocks received from the network host do not contain these two characters.
3. Data characters are transmited in succession with no time lapse between characters.
4. The "end transmission" control characters identify the end of data. DLE ETX or DLE ETB indicate transparent data. ETX or ETB indicates normal data.
5. Block check characters (usually two characters) are used to verify data integrity. If the value received does not match the value calculated by the receiver, the entire block is rejected by the receiver. Block check includes GID and DID characters in transmissions to the network host.

Protocol control characters are stripped from the data transfer, and are not passed from the interface to the computer. For information about network polling, terminal selection and other Data Link operations, consult the Data Link network manuals supplied with the HP 1000 or HP 3000 network host computer.

Data Transfers Between Computer and Interface

Data transfers between your desktop computer and its datacomm interface involve two message types: control blocks, and data. Both types are encountered in both output and input operations as follows:

- Outbound control blocks are created by IOCONTROL procedures.
- Outbound data messages are created by the output procedures.
- Inbound control blocks are created by certain protocol operations such as Data Link block boundaries, or Async prompt, end-of-line, parity/framing error, or break detection.
- Inbound data messages are created by the interface as messages are received from the remote. They are transferred to the Pascal programs via the input procedures.

Outbound Control Blocks

Outbound control blocks are messages from your computer to the datacomm interface that contain interface control information. They are usually generated by IOCONTROL procedures, although TRANSFER_END creates a control block that terminates a given Async transmission or forces a block to be sent on the Data Link. Outbound control blocks are serially queued with data. An exception to the queued control block rule is output to Control Register 0 (card reset) which is executed immediately.

Note

When an interface card reset is executed by use of a IOCONTROL procedure, the control block that results is transmitted directly to the interface. It is not queued up, so any previously queued data and control blocks are destroyed. To prevent loss of data, be sure that all queued messages have been sent before resetting the datacomm interface. IOStatus Register 38 returns a value of 1 when the outbound queue is empty. Otherwise, its value is 0. To prevent loss of inbound data, IOStatus Register 5 must return a value of zero prior to reset.

Inbound Control Blocks

Inbound control blocks are messages from the interface to the computer that identify protocol control information. Which item(s) are allowed to create a control block is determined by the contents of IOControl Register 14. IOStatus Registers 9 and 10 identify the contents of the block, and IOControl Register 24 defines what protocol characters are also included with inbound Async data messages. Refer to the IOControl and IOStatus Register section at the end of this chapter for details about register contents for various control block types.

Two types of information are contained in each control block: Type and Mode. The TYPE is contained in IOSTATUS register 9; the MODE in IOSTATUS register 10. Type and Mode values can be used to interpret datacomm operation as follows:

Async Protocol Control Blocks

Type	Mode	Interpretation
250	1	Break received (channel A).
251	1[1]	Framing error in the following character.
251	2[1]	Parity error in the following character.
251	3[1]	Both Framing and Parity error in the following character.
252	1	End-of-line terminator detected.
253	1	Prompt received from remote.

Data Link Protocol Control Blocks

Type	Mode	Interpretation
254	1	Preceding block terminated by ETB character.
254	2	Preceding block terminated by ETX character.
253[2]		(See following table for Mode interpretation.)

Mode Bit(s)	Interpretation
0	1 = Transparent data in following block. 0 = Normal data in following block.
2,1	00 = Device Select (most common). 01 = Group Select 10 = Line Select
3	1 = Command Channel 0 = Data Channel

For Data Link applications, control blocks are normally set up for end-of-block (ETB or ETX). Control blocks are then used to terminate TRANSFER_END operation, or are trapped via an I/O escape. Control block contents are not important for most applications unless you are doing sophisticated protocol-control programming.

For Async applications, terminal emulator programs usually use prompt and end-of-line control blocks. Use of other functions such as break or error detection depend on the requirements of the individual application.

1 Parity/framing error control blocks are not generated when characters with parity and/or framing errors are replaced by an underscore (_) character.

2 This type is used mainly in specialized applications. In most cases, you can expect a Mode value of zero or one for Type 253 Data Link control blocks. For most Data Link applications, control blocks are not used by programmers.

Outbound Data Messages

Outbound data messages are created when an output procedure is executed. Here is a short summary of how output parameters can affect datacomm operation.

- Async protocol: Data is transmitted directly from the outbound queue. When operating in half-duplex, TRANSFER_END causes the interface to turn the line around and allow the remote device to send information back (line turn-around is initiated when the interface sets the Request-to-send line low). TRANSFER_END has no effect when operating in full duplex.
- Data Link protocol: Data messages are concatenated until at least 512 characters are available, then a block of 512 characters is sent. Block boundaries may or may not coincide with the end of a given output message.
  You can force transmission of shorter blocks by using the TRANSFER_END procedure. The interface then transmits the last pending block regardless of its length. This technique is useful for ensuring that block boundaries coincide with message boundaries, or for sending one message string per block when you are transmitting short records.

Inbound Data Messages

Inbound data messages are created by the datacomm interface as information is received from the remote. Input procedures are terminated when a control block is encountered or the input variable is filled. Whether control characters are included in the data stream depends on the configuration of Control Register 24 (Async operation only). Control information is never included in inbound data messages when using Data Link protocol.

With this brief introduction to the data communications capabilities of the HP 98628 Datacomm Interface, you are ready to begin programming your desktop computer for datacomm operation. The next section of this chapter introduces Pascal datacomm programming techniques.

Overview of Datacomm Programming

Your desktop computer uses several I/O Library facilities for data communication with various computers, terminals, and other peripheral devices. Datacomm programs will include part or all of the following elements:

- Input procedures (including transfers)
- Output procedures (including transfers)
- IOSTATUS functions
- IOCONTROL procedures
- High level control procedures.

The input and output procedures are described in the previous chapters. Later sections of this chapter discuss the IOSTATUS and IOCONTROL operations. The I/O Library provides several high level control procedures to set up the serial interface card and its parameters. These procedures are in the module SERIAL_3 and consist of the following procedures. Note that these procedures are for ASYNC operations ONLY.

Set Baud Rate

This procedure will set the interface baud rate. It is of the form:

```
SET_BAUD_RATE ( isc , rate );
```

The rate is a real expression with the range of 0 through 19 200.

Set Stop Bits

This procedure will set the number of stop bits on the interface. The procedure is of the form:

```
SET_STOP_BITS   ( isc , number_of_bits );
```

The number of bits is a real expression with valid values of 1, 1.5 and 2.

Set Character Length

This procedure will set the number of bits in a character on the specified interface. The procedure is of the form:

```
SET_CHAR_LENGTH ( isc , number_of_bits );
```

The number of bits is an integer expression with valid values of 5, 6, 7, and 8 bits per character.

Set Parity

This procedure sets the parity mode of the specified interface. The procedure is of the form:

```
SET_PARITY ( isc , parity );
```

The parity parameter is an enumerated type with the following values:

no_parity
odd_parity
even_parity
zero_parity
one_parity

Example Terminal Emulator

The following program is a very simple terminal emulator. It uses overlap transfers to bring data into the computer and uses handshake I/O to send data from the computer. This is not a supported product — merely an example program.

```
$SYSPROG ON$
$UCSD ON$
$DEBUG ON$

PROGRAM TERMINAL(INPUT,OUTPUT,KEYBOARD);

  IMPORT iodeclarations,
         general_0,
         general_1,
         general_2,
         general_3,
         general_4,
         serial_0,
         serial_3;

  CONST  mysc         = 20;
         bufsize      = 1000;
         Kbdunit      = 2;
  VAR    i            : INTEGER;
         mybuf        : buf_info_type;
         bufchar      : CHAR;
         oldbufchar   : CHAR;
         Kbdchar      : CHAR;
         half_duplex  : BOOLEAN;
         auto_lf      : BOOLEAN;

  BEGIN

    TRY

      ioinitialize;

      iocontrol       (mysc,22,0);  { no protocol }
      iocontrol       (mysc,23,0);  { no handshake }
      iocontrol       (mysc,24,127);{ pass all chars }
      iocontrol       (mysc,28,0);  { card EOL = none }

      set_baud_rate   (mysc,2400);
      set_parity      (mysc,odd_parity);
      set_char_length(mysc,7);
      set_stop_bits   (mysc,1);

      iocontrol       (mysc,8,63);  { set all modem lines }

      iocontrol       (mysc,12,1);  { connect the card }


      half_duplex := TRUE ;
      auto_lf     := TRUE ;
```

```
  iobuffer(mybuf,bufsize);
  transfer(mysc,overlap,to_memory,mybuf,bufsize);
  WRITELN('TERMINAL EMULATOR READY');
  REPEAT
    IF NOT ( UNITBUSY(kbdunit) )
      THEN BEGIN
        IF EOLN(keyboard)
          THEN BEGIN
           READ(keyboard,kbdchar);
           kbdchar := io_carriage_rtn;
          END
          ELSE BEGIN
           READ(keyboard,kbdchar);
          END; { of IF EOLN }
        IF half_duplex
          THEN BEGIN
            WRITE(kbdchar);
          END;
        IF auto_lf AND ( kbdchar = io_carriage_rtn )
          THEN BEGIN
            writechar(mysc,kbdchar);
            kbdchar := io_line_feed;
          END;
        writechar(mysc,kbdchar);
      END;

    IF buffer_data(mybuf) <> 0
      THEN BEGIN
        oldbufchar := bufchar;
        readbuffer(mybuf,bufchar);
        IF bufchar    = io_line_feed
          THEN BEGIN
            IF oldbufchar = io_carriage_rtn
              THEN BEGIN
                { nothing }
              END
              ELSE BEGIN
                WRITE(io_carriage_rtn);
              END;
          END
          ELSE BEGIN
            WRITE(bufchar);
          END;
      END;
    IF (mybuf.active_isc = no _isc) AND (buffer_data (mybuf) = 0)
      THEN BEGIN
        transfer(mysc,overlap,to_memory,mybuf,bufsize);
      END;
UNTIL FALSE;

RECOVER BEGIN

  PAGE(output);
  WRITELN;
  WRITELN('escape code : ',escapecode);
  IF ESCAPECODE=ioescapecode
    THEN BEGIN
      WRITELN('some I/O problem has occurred');
      WRITELN(ioerror_message(ioe_result));
      WRITELN('on select code ',ioe_isc:4);
    END
    ELSE BEGIN
      IF ESCAPECODE<>-20
        THEN BEGIN
          WRITELN('some non-I/O problem has occurred');
        END
        ELSE BEGIN
```

continued

```
            WRITELN('stop key pressed');
          END;
      END;

    ESCAPE(ESCAPECODE);

  END;

END.
```

Establishing the Connection

Determining Protocol and Link Operating Parameters

Before information can be successfully transferred between two devices, a communication link must be established. You must include the necessary protocol parameters to ensure compatibility between the communicating machines. To determine the proper parameters for your application, select Async or Data Link protocol, then answer the following questions:

For BOTH Async and Data Link Operation:

- Is a modem connection being used? What handshake provisions are required? (Data Link does not use modems, but multi-point Async modem connections use a protocol compatible with Data Link.)
- Is half-duplex or full-duplex line protocol being used?

For Async Operation ONLY:

- What line speed (baud rate) is being used for transmitting?
- What line speed is being used for receiving?
- How many bits (excluding start, stop, and parity bits) are included in each character?
- What parity is being used: none, odd, even, always zero, or always one?
- How many stop bits are required on each character you transmit?
- What line terminator should you use on each outgoing line?
- How much time gap is required between characters (usually 0)?
- What prompt, if any, is received from the remote device when it is ready for more data?
- What line terminator, if any, is sent at the end of each incoming line?

For Data Link Operation ONLY:

- What line speed (baud rate) is being used? (Data Link uses the same speed in both directions.)
- What parity is being used: none (HP 1000 network host), or odd (HP 3000 network host)?
- What is the device Group IDentifier (GID) and Device IDentifier (DID) for your terminal?
- What is the maximum block length (in bytes) the network host can accept from your terminal?

All these parameters are configured under program control by use of IOCONTROL procedures. Alternately, default values for line speed, modem handshake, parity, and Async or Data Link protocol selection can be set using the datacomm interface configuration switches. Other default parameters are preset by the datacomm interface to accommodate common configurations. You can use the defaults, or you can override them with IOCONTROL procedures for program clarity and immunity to card settings. Default IOControl Register values are shown in

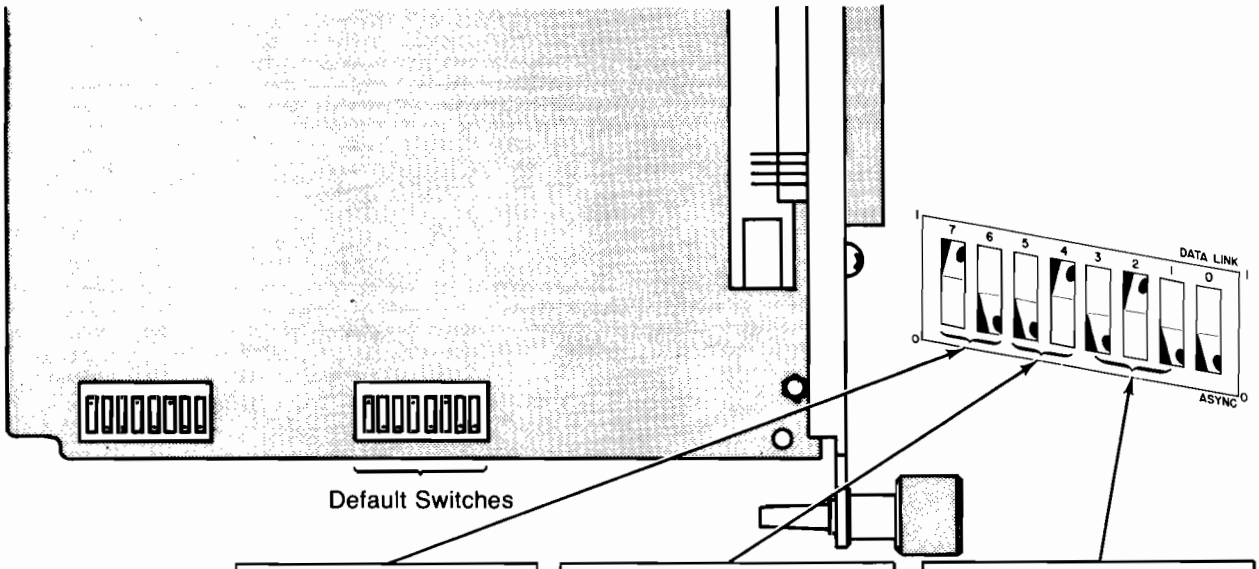
the IOCONTROL and IOSTATUS register tables in the back of this chapter. The HP 98628 Datacomm Interface Installation manual (98628-90000) explains how to set the default switches on the interface.

The next section of this chapter shows a summary of the available default options and switch settings for both Async and Data Link.

Using Defaults to Simplify Programming

The datacomm interface includes two switch clusters. One cluster is used to program the select code and interrupt level. The other cluster sets defaults for protocol, line speed (baud rate), modem handshake, and parity. Setting the defaults on the card eliminates the need to program the corresponding interface IOCONTROL registers. These defaults are useful in applications where the configuration of the link is rarely altered, and the program is not used on other machines with dissimilar configurations. They also enable a beginning programmer to use output and input procedures to perform simple datacomm operations without using IOCONTROL or IOSTATUS statements. On the other hand, where link configuratiion may vary, or where programs are used on several different machines with dissimilar configurations, it is usually worthwhile to override the defaults with IOCONTROL procedures. This assures known datacomm behavior, independent of interface defaults.

Here, for your convenience is a brief summary of the default switch options:



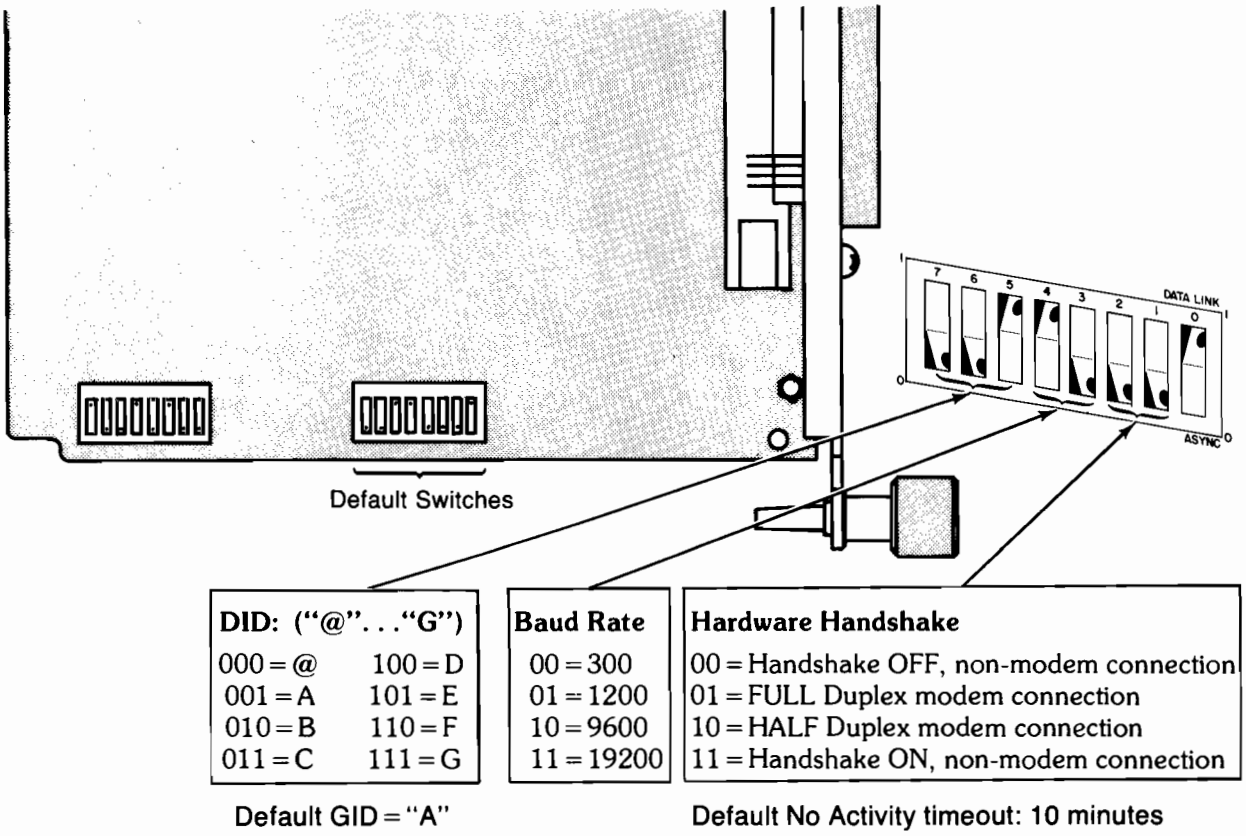
Parity	Bits/Char
00 = None	8
01 = None	7
10 = Odd	7
11 = Even	7

Hardware Handshake
00 = Handshake OFF, non-modem connection[1]
01 = FULL Duplex modem connection[2]
10 = HALF Duplex modem connection [2]
11 = Handshake ON, non-modem connection[1]

Baud Rate	Stop Bits
000 = 110	2
001 = 150	2
010 = 300	1
011 = 600	1
100 = 1200	1
101 = 2400	1
110 = 4800	1
111 = 9600	1

Async Default Configuration Switches

[1] Default No Activity timeout: Disabled
[2] Default No Activity timeout: 10 minutes



Data Link Default Configuration Switches

Resetting the Datacomm Interface

Before you establish a connection, the datacomm interface must be in a known state. **The datacomm interface does not automatically disconnect from the datacomm link when the computer reaches the end of a program.** To prevent potential problems caused by unknown link conditions left over from a previous session, it is a good practice to reset the interface card at the beginning of your program before you start configuring the datacomm connection. Resetting the card causes it to disconnect from the line and return to a known set of initial conditions.

Example

```
IORESET (20) ;
```

Protocol Selection

During power-up and reset, the card uses the default switches to preset the card to a known state. The protocol select switch defines which protocol the card uses at power-up only. If the default protocol is the same as you are using, you can skip the protocol selection statements. However, if the switch might be set to the wrong protocol, or if you want to change protocol in the middle of a program, you can use a IOCONTROL procedure to select the protocol. After the protocol is selected, reset the card again to make the change. Here is how to do it:

Select the protocol to be used:

```
IOCONTROL (SC,3,1); {Select Async Protocol}
```

or

```
IOCONTROL (Sc,3,2); {Select Data Link Protocol}
```

Wait until the protocol select message has been sent to the card, then reset the card. The Reset command restarts the interface microcomputer using the selected protocol.

```
REPEAT
UNTIL IOSTATUS(Sc,38) =1 ;
IORESET (Sc) ;
```

Note

Be careful when resetting the interface card during normal program operation. Data and Control information are sent to the card in the same sequence as the statements originating the information are executed. When a card reset is initiated by a IOCONTROL procedure, the reset is not placed in the queue with outbound data, but is executed immediately. Therefore, if there is other information in the output queue waiting to be sent, a reset can cause the data to be lost. To prevent loss of data, use IOSTATUS function (register 38) to verify that all data transfers have run to completion before you reset the interface.

You are now ready to program datacomm options that are related to the selected protocol. In applications where defaults are used, the options are very simple. The following pair of examples shows how to set up datacomm options for each protocol.

Datacomm Options for Async Communication

This section explains how to configure the datacomm interface for asynchronous data communication. The example used shows how to set up all configurable options without considering default values. Some statements in the example are redundant because they override interface defaults having the same value. Others may or may not be redundant because they override configuration switch options. The remaining statements are necessary because they override the default values, replacing them with non-default values required for proper operation of the example program. If you are not familiar with Asynchronous protocol, consult the section on protocol for the needed background information.

Control Block Contents

Configuration of the link begins with register 14 which determines what information is placed in the control blocks that appear in the input (receive) queue. In this example, only the end-of-line position and prompts are identified. Parity or framing errors in received data, and received breaks are not identified in the queue. This register interacts with Control registers 28 thru 33.

Datacomm Line Timeouts

Registers 16-19 set timeout values to force an automatic disconnect from the datacomm link when certain time limits are exceeded. For most applications, the default values are adequate. A value of zero disables the timeout for any register where it is used. Each register accepts values of 0 thru 255; units vary with the register function.

- Register 16 (Connection timeout) sets the time limit (in seconds) allowed for connecting to the remote device. It is useful for aborting unsuccessful attempts to dial up a remote computer using public telephone networks.
- Register 17 (No Activity timeout) sets an automatic disconnect caused by no datacomm activity for the specified number of minutes. Default value is determined by default handshake switch setting. Default is not affected by IOCONTROL procedures to IOControl Register 23 (hardware handshake).
- Register 18 (Lost Carrier timeout) disconnects when:

  Full Duplex: Data Set Ready (Data Mode) or Data Carrier Detect go false, or
  Half Duplex: Data Set Ready goes false,

  indicating that the carrier from the remote modem has disappeared from the line. Value is in multiples of 10 milliseconds.

- Register 19 (Transmit timeout) disconnects when a loss-of-clock occurs or a clear-to-send (CTS) is not returned by the modem within the specified number of seconds.

Line Speed (Baud Rate)

The transmit and receive line speed(s) are set by IOControl Registers 20 and 21, respectively. Each is independent of the other, and they are not required to have identical values. The following baud rates are available for Async communication:

Register Value	Baud Rate	Register Value	Baud Rate	Register Value	Baud Rate	Register Value	Baud Rate
0	0[1]	4	134.5	8	600[2]	12	3600
1	50	5	150[2]	9	1200[2]	13	4800[2]
2	75	6	200	10	1800	14	9600[2]
3	110[2]	7	300[2]	11	2400[2]	15	19 200

All configurable line speeds are available to IOCONTROL Registers 20 and 21. Only the eight speeds indicated can be selected using the default switches (see the switch configuration diagram earlier in this chapter). When the configuration switch defaults are used, transmit and receive speeds are identical. The selected line speed must not exceed the capabilities of the modem or link.

1 An external clock must be provided for this option.

2 These speeds can be programmed using the default switches on the interface card. Other speeds are accessed by CONTROL statements. (The HP 13265A Modem can be operated up to 300 baud.)

Handshake

Registers 22 and 23 configure handshake parameters. There are two types of handshake:

- Software or protocol handshake specifies which of the participants is allowed to transmit while the other agrees to receive until the exchange is reversed. Options include:

    1. No handshake, commonly used with connections to non-interactive devices such as printers.

    2. Enq/Ack (EQ/AK) or DC1/DC3 handshake, with the desktop computer configured either as a host or a terminal. Handshake characters are defined by registers 26 and 27.

    3. DC1/DC3 handshake with the desktop computer as both a host AND a terminal. Handshake characters are defined by registers 26 and 27. This option simplifies communication between two desktop computers.

- Hardware or modem handshake that establishes the communicating relationship between the interface and the associated datacomm hardware such as a modem or other link device. The four available options are:

    1. Handshake Off, non-modem connection – most commonly used for 3-wire direct connections to a remote device.

    2. Full Duplex modem connection – used with full-duplex modems or equivalent connections.

    3. Half Duplex modem connection – used with half-duplex modems or equivalent connections.

    4. Handshake On, non-modem connection – used with printers and other similar devices that use the Data Carrier Detect (DCD) and Clear-to-send (CTS) lines to signal the interface card. When DCD is held down by the peripheral, the interface ignores incoming data. When CTS is held down, the interface does not transmit data to the device until CTS is raised.

Options 2 and 3 are usually associated with modems or similar devices, but may be used occasionally with direct connections when the remote device provides the proper signals. Refer to the table at the end of this chapter for a list of handshake signals and how they are handled for each cable or adapter option.

Handling of Non-data Characters

Register 24 specifies what non-data characters are to be included in the input queue. For each bit that is set, the corresponding information is passed along with the incoming data. If the bit is not set, the information is discarded, and is not included in the inbound data stream that is passed to the desktop computer by the interface.

Bit 0: Include handshake characters in data stream. They are defined by Control Registers 26 and 27.

Bit 1: Include incoming end-of-line character(s). EOL characters are defined by Control Registers 28-30.

Bit 2: Include incoming prompt character(s). Prompt is defined by Control Registers 31-33.

Bit 3: Include any null characters encountered.

Bit 4: Include any DEL (rubout) characters in data.

Bit 5: Include any CHR$(255) encountered. This character is encountered ONLY when 8-bit characters are received.

Bit 6: Change any characters received with parity or framing errors to an underscore. If this bit is not set, all inbound characters are transferred exactly as received, with or without errors.

Register 25 is not used.

Protocol Handshake Character Assignment

Registers 26 and 27 establish what characters are to be used for handshaking between communicating machines. You can select the values of 6 (AK) or 17 (DC1) for register 26, and 5 (EQ) or 19 (DC3) for register 27. Any ASCII value from 0 thru 255 can be used, but non-standard values should be reserved for exceptional situations.

End-of-line Recognition

Registers 28, 29, and 30 operate in conjunction with registers 14 (control block mask) and 24 (non-data character stripping) and defines the end-of-line sequence used to identify boundaries between incoming records. Register 28 (value of 0, 1 or 2) defines the number of characters in the sequence, while registers 29 and 30 contain the decimal equivalent of the ASCII characters. If register 28 is set for one character, register 30 is not used. Register 29 contains the first EOL character, and register 30, if used, contains the second. If register 28 is zero, registers 29 and 30 are ignored and the interface cannot recognize line separators.

Prompt Recognition

Registers 31, 32, and 33 operate in conjunction with registers 14 and 24 and define the prompt sequence that identifies a request for data by the remote device. As with end-of-line recognition, the first register defines the number of characters (0, 1, or 2), while the second and third registers contain the decimal equivalents of the prompt character(s). Register 33 is not used with single-character prompts. If register 31 is zero, registers 32 and 33 are ignored and the interface is unable to recognize any incoming prompts.

Character Format Definition

Registers 34 through 37 are used to define the character format for transmitted and incoming data.

- Register 34 sets the character length to 5, 6, 7, or 8 bits. The value used is the number of bits per character minus five (0 = 5 bits, 3 = 8 bits). When 8-bit format is specified, parity must be Odd, Even, or None (parity "1" or "0" cannot be used).
- Register 35 specifies the number of stop bits sent with each character. Values of 0, 1, or 2 are used to select 1, 1.5, or 2 stop bits, respectively.
- Register 36 specifies the parity to be used. Options include:

Register Value	Parity	Result
0	None	Characters are sent with no parity bit. No parity checks are made on incoming data.
1	Odd[1]	Parity bit is set if there is an EVEN number of ones in the character code. Incoming characters are also checked for odd parity.
2	Even[1]	Parity bit is set if there is an ODD number of ones in the character code.
3	0	Parity bit is present, but always zero. No parity checks are made on incoming data.
4	1	Parity bit is present, but always one. No parity checks are made on incoming data.

Parity must be odd, even, or none when 8-bit characters are being transferred.

- Register 37 sets the time gap (in character times, including start, stop, and parity bits) between one character and the next in a transmission. It is usually included to allow a peripheral, such as a teleprinter, to recover at the end of each character and get ready for the next one. A value of zero causes the start bit of a new character to immediately follow the last stop bit of the preceding character.

Control Register 38 is not used.

Break Timing

Register 39 sets the break time (2-255 character times). A Break is a time gap sent to the remote device to signify a change in operating conditions. It is commonly used for various interrupt functions. The interface does not accept values less than 2. Register 6 is used to transmit a break to the remote computer or device.

Datacomm Options for Data Link Communication

This section explains how to configure the datacomm interface for Data Link operation. If you are not familiar with Data Link protocol and terminology, consult the section on protocol for the needed background information.

1 Parity sense is based on the number of ones in the character including the parity bit. An EVEN number of ones in the character, plus the parity bit set produces an ODD parity. An ODD number of ones in the character plus the parity bit set produces an EVEN parity.

Control Block Contents

Data Link configuration begins with IOControl Register 14. This register determines what information is to be placed in control blocks and included with inbound data transferred from the interface to the desktop computer.

- ETX (Bit 1) identifies the end of a transmission block that contains one or more complete records.
- ETB (Bit 2) identifies the end of a transmission block where the last record is continued in the next block of data.
- Bit 0 causes a control block to be inserted that identifies the beginning of a new block of data.

Datacomm Line Timeouts, and Line Speed

Registers 15 through 19 are functionally identical for both Async and Data Link. Refer to the preceding Async section for more information. Register 20 sets the line speed for both transmitting and receiving (Data Link does not accommodate split-speed operation). The following line speed options are available:

Register Value	Baud Rate	Register Value	Baud Rate	Register Value	Baud Rate	Register Value	Baud Rate
0	0[1]	9	1200[2]	12	3600	15	19 200[2]
7	300[2]	10	1800	13	4800		
8	600	11	2400	14	9600[2]		

Terminal Identification

Registers 21 and 22 specify the terminal identifier characters for the datacomm interface. Register 21 contains the GID (Group IDentifier), and register 22 contains the DID (Device IDentifier. Values of 0-26 correspond to the characters @, A, B, . . ., Z. These registers must be configured to match the terminal identification pair assigned to your device by the Data Link Network Manager. In the example, Line 1320 is redundant because it duplicates the default GID value. Line 1330 overrides the DID default switch on the interface card, and may or may not be necessary. Alternate methods for assigning different GID/DIDs are shown following the group of configuration IOCONTROL procedures.

Handshake

Register 23 establishes the hardware handshake type. There is no formal software handshake with Data Link because the network host controls all data transfers. Hardware or modem handshake options are identical to Asynchronous operation. Handshake should be OFF (register set to 0) when using the HP 13264A Data Link Adapter. When you are using non-standard interconnections such as direct or modem links to the network host, select the handshake option that fits your application. Refer to the table at the end of this chapter for a list of handshake signals and how they are handled for each cable or adapter option.

1 An external clock must be provided for this option.

2 These speeds can be programmed using the default switches on the interface card. Other speeds are accessed by CONTROL statements.

Transmitted Block Size

Register 24 defines the maximum transmitted block length. When transmitting blocks of data to the network host, the block length must not exceed the available buffer space on the receiving device. Block size can be specified for increments of two from 2 to 512 characters per block. A value of zero forces the block length to a maximum of 512 bytes. For other values, the block length limit is twice the value sent to the register. For example, a register value of 130 produces a transmitted block length not exceeding 260 characters (bytes).

Parity

Register 36 defines the parity to be used. Unlike Async, Data Link has only two parity options: None, or Odd. Odd parity is:

Register Value	Parity	Application
0	NONE	Required for operation with HP 1000 network host
1	ODD	Required for operation with HP 3000 network host

Registers 25 through 35, and 37 and above are not used.

Connecting to the Line

Interface configuration is now complete. You are ready to begin connecting to the datacomm line. The exact procedure used to connect to the line varies slightly, depending on the type of link being used. Before you connect, you must know what the link requirements are, including dialing procedures, if any.

Switched (Public) Telephone Links

When you are using a public or switched telecommunications link, the modem connection between computers must be established. The HP 13265A Modem can be used in any Async application that requires a Bell 103- or Bell 113-compatible modem operating at up to 300 baud line speed. However, the HP 13265A Modem is not suitable for data rates exceeding 300 baud. For higher baud rates, use a modem that is compatible with the one at the remote computer site. Modems cannot be used for remote connections from a terminal to the data link.

Private Telecommunications Links

Private (leased) links require modems unless the link is short enough for direct connection (up to 50 feet, depending on line speed). The HP 13265A Modem can be used at data rates up to 300 baud. For higher speeds, a different modem must be used.

Direct Connection Links

For short distances, a direct connection may be used without modems or adapters, provided both machines use compatible interfaces. Async connections normally use RS-232C interfaces. You can also operate as a Data Link terminal directly connected to an HP 1000 or HP 3000 host computer through a dedicated Multipoint Async interface on the network host, although such connections are unusual.

Data Link Connections

Most Data Link connections use an HP 13264A Data Link Adapter to connect directly to the Data Link. In special situations, a modem may be used to communicate with a Multipoint Async interface on the HP 1000 or HP 3000 network host. When the Data Link Adapter is used, no special procedures are required. If you are using a leased or switched telecommunications link, the procedures are the same as when using point-to-point Async with modems.

Connection Procedure

This section describes procedures for modem connections using telephone telecommunications circuits. If you are NOT using a switched, modem link, skip to the next section: Initiating the Connection.

Dialing Procedure for Switched (Public) Modem Links

Except for dialing, connection procedures do not usually vary between switched and dedicated links. Dialing procedures depend on whether the modem is designed for manual or automatic dialing. Automatic dialing can be used with the HP 13265A Modem, but other modems must be operated with manual dialing unless you design your own interface to an Automatic Calling Unit. For manual dialing procedures, consult the operating manual for the modem you are using.

Automatic Dialing with the HP 13265A Modem:

The automatic dialer in the HP 13265A Modem is accessed by Control Register 12. The IOCONTROL procedure is followed by an output procedure that contains the telephone number string, including dial rate and timing characters. The two statements set up the automatic dialer, but dialing is not started until a "start connection" command is sent to IOControl Register 12. Here is an example sequence:

```
IOCONTROL (Sc,12,2) ;
WRITESTRING (Sc,'> 9 @@@ (303)-555-1234');
```

Unrecognized characters are ignored.

3-second wait for secondary dial tone.

Select FAST dial rate.

The output procedure contains several essential elements.

- The first character (">"), if included, specifies a fast dialing rate. If it is omitted, the default slow dialing rate is used.
- A time delay character "@" may be inserted anywhere in the string. A one-second time delay is executed in the dialing sequence each time a delay character is encountered.
- Numeric character sequences define the telephone number. Multiple dial-tone sequences, such as when calling out from a PBX (Private Branch Exchange), can be used by inserting a suitable delay to wait for the next dial tone.
- Unrecognized characters such as parentheses, hyphens, and spaces can be included for clarity. They are ignored by the automatic dialer.
- Up to 500 characters can be included in the telephone number string.

Here is how an autodial connection is executed:

- The `IOCONTROL (Sc,12,2)` places a "start dialing" control block in the outbound queue to the interface. The OUTPUT statement places the telephone number string (including spaces and other characters) in the queue after the control block. When the interface encounters the control block, it transfers the string to the HP 13265A Modem's autodial circuit. No other action is taken at this time.
- When `IOCONTROL (Sc,12,1)` is executed, another control block is queued up. When the interface encounters the block, it sends a "start connection" command to the modem. The modem then disconnects from the line, waits two seconds, then reconnects. The autodialer waits 500 milliseconds, then starts executing the telephone number string. The string is executed character-by-character in the same sequence as sent by the output procedure.
- If your application requires more than 500 milliseconds to guarantee a dial tone is present, you can increase the delay by adding delay characters ("@") where needed, one second per character. Be sure to provide adequate delays in multiple dial tone sequences, such as when calling through a private branch exchange (PBX) to a public telephone network.
- When dialing is complete, the modem is connected to the line, and you are ready to start communication. The next section explains how to determine when connection is complete.

Two dialing rates are available: slow (default) and fast. To select the fast rate, you must include the fast rate character (">") as the FIRST character in the telephone number string. Here is a summary of differences between the two options:

Parameter	Slow Dialing	Fast Dialing
Click Length	60 milliseconds	32.5 milliseconds
Click Gap	40 milliseconds	17.5 milliseconds
Number Gap	700 milliseconds	300 milliseconds

One to ten dial pulses (clicks) are sent for each digit 1 through 0, respectively. The number gap is the time lag between the end of the last click of one number and the beginning of the first click of the next number.

Most Bell System facilities can handle both fast and slow dialing rates, but private or independent telephone systems or companies may require slow dialing.

Initiating the Connection

After you have executed the necessary dialing procedures, if any, you are ready to initiate the connection. The following statement is used to start the connection:

```
IOCONTROL (Sc,12,1) !{Start Connection.}
```

This statement sends a control block to the interface telling it to connect to the datacomm line. If the HP 13265A Modem is being used, and the autodialer is enabled, it starts dialing the number. Otherwise, the interface executes a direct connection to the line, or tells the modem or data link adapter to connect.

The status of the connection process can be monitored by using the IOSTATUS function. The following lines hold the computer in a continuous loop until the connection is complete:

```
REPEAT

   State:=IOSTATUS(Sc,12);
   IF State=2 THEN WRITELN C"Dialing");
   IF State=1 THEN WRITELN C"Trying to Connect");
UNTIL State=3;
WRITELN ("Connected") ;
```

Refer to the IOStatus and IOControl Register section for interpretation of the values in IOStatus Register 12. Only values of 1, 2, or 3 are usually encountered at this stage of the program.

As soon as IOStatus Register 12 indicates that connection is complete, you are ready to continue into the main body of the terminal emulator or other program you are writing. This completes the datacomm initialization and connection phase of the program.

Datacomm Errors and Recovery Procedures

Several errors can be encountered during datacomm operation. They are listed here with probable causes and suggested corrective action.

Error	Description and Probable Cause
306	Interface card failure. This error occurs during interface self-test, and indicates an interface card hardware malfunction. You can repeat the power-up self-test by pressing (SHIFT)-(PAUSE). If the error persists, replace the defective card. Using a defective card may result in improper datacomm operation, and should be considered only as a last resort.
313	USART receive buffer overflow. The SIO buffer is not being cleared fast enough to keep up with incoming data. This error is uncommon, and is usually caused by excessive processing demands on the interface microprocessor. To correct the problem, examine Pascal program flow to reduce interference with normal interface operation. This error causes the interface to disconnect from the datacomm line and go into a SUSPENDED state. Clear or reset the interface card to recover.
314	Receive Buffer overflow. Data is not being consumed fast enough by the desktop computer. Consequently, the buffer has filled up causing data loss. This is usually caused by excessive program demands on the desktop computer CPU, or by poor program structure that does not allow the desktop computer to properly service incoming data when it arrives. Modify the Pascal program(s) to allow more frequent interrupt processing by the desktop computer, or change to a lower baud rate and/or use protocol handshaking to hold off incoming data until you are ready to receive it. This error causes the interface to disconnect from the datacomm line and go into a SUSPENDED state. Clear or reset the interface to recover.
315	Missing Clock. A transmit timeout has occurred because the transmit clock has not allowed the card to transmit for a specified time limit (Control Register 19). This error can occur when the transmit speed is 0 (external clock), and no external clock is provided, or be caused by a malfunction. The interface is disconnected from the datacomm line and is SUSPENDED. To recover, correct the cause, then reset the card.

Error	Description and Probable Cause
316	CTS false too long. Due to clear-to-send being false on a half-duplex line, the interface card was unable to transmit for a specified time limit (Control Register 19). The card has disconnected from the datacomm line, and is in a SUSPENDED state. To recover, determine what has caused the problem, correct it, then reset or clear the interface card.
317	Lost Carrier disconnect. Data Set Ready (DSR) (and/or Data Carrier Detect, if full-duplex) went inactive for the specified time limit (Control Register 18). This condition is usually caused by the telecommunications link or associated equipment. The card has disconnected from the datacomm line and is in a SUSPENDED state. To recover, clear or reset the interface card.
318	No Activity Disconnect. The interface card disconnected from the datacomm line automatically because no information was transmitted or received within the time limit specified by Control Register 17. The card is in a SUSPENDED state. Clear or reset the interface to recover.
319	Connection not established. The card attempted to establish connection, but Data Set Ready (DSR) (and Data Carrier Detect, if full duplex) was not active within the time limit specified by Control Register 16. The card has disconnected from the datacomm line and is in a SUSPENDED state. Clear or reset the interface to recover.
325	Illegal DATABITS/PARITY combination. IOCONTROL procedures have attempted to program 8 bits per character and parity "1" or "0". The IOCONTROL procedure causing the error is ignored, and the previous setting remains unchanged. To correct the problem, change the IOCONTROL procedure(s) and/or interface default switch settings.
326	Register address out of range. An IOCONTROL or STATUS function has attempted to address a non-existing register. The command is ignored, and the interface card state remains unchanged.
327	Register value out of range. An IOCONTROL procedure attempted to place an illegal value in a defined register. The command is ignored, and the interface card state remains unchanged.

Error Recovery

When any error from Error 313 through Error 319 occurs, it forces the interface card to disconnect from the datacomm line. When a forced disconnect terminates the connection, the interface is placed in a SUSPENDED state, indicated by Status Register 12 returning a value of 4. The interface cannot be reconnected to the datacomm line when it is SUSPENDED. ABORT_SERIAL and IORESET are used to recover from the suspended state and resume normal card operation.

To recover from a SUSPENDED interface, two programmable options are available, all of which destroy any existing data in the transmit and receive queues. They are:

- The ABORT_SERIAL procedure clears the receive and transmit queues.
- RESET interface (IOControl Register 0 or IORESET) clears all buffers and queues, and resets all IOCONTROL options to their power-up state EXCEPT the protocol which is determined by the most recent IOCONTROL statement (if any) addressed to register 3 since power-up.

A fourth (keyboard only) option is available. (CLR IO) causes a hardware reset to be sent to ALL peripherals. This completely resets the datacomm interface to its power-up state with protocol and other options determined by the default switch settings.

Datacomm Programming Helps

This section is designed to assist you in writing datacomm programs for special applications by discussing selected techniques and characteristics that can present obstacles to the beginning programmer.

Terminal Prompt Messages

Care must be exercised to ensure that messages are never transmitted to the network host if the host is not prepared to properly handle the message. Receipt of a poll from the host does not necessarily mean that the host can handle the message properly when it is received. Therefore, prompts or interpretation of messages from the host are used to determine the status of the host operating system.

Prompts are message strings sent to the terminal by a cooperating program. They are well-defined and predictable, and are usually tailored to specific applications. When the terminal interacts directly with RTE or one or more subsystems, the process becomes less straightforward. Each subsystem usually has its own prompt which is not identical to other subsystem prompts. To maintain orderly communication with subsystems, you must interpret each message string from the host to determine whether it is to be treated as a prompt.

Prevention of Data Loss on the HP 1000

On the HP 1000, the RTE Operating System manages information transfer between programs or subsystems and system I/O devices, including DSN/DL. Terminals are continually polled by the host's data link interface (unless auto-poll has been disabled by use of an HP 1000 File Manager CN command). Since there is no relationship between automatic polling and HP 1000 program and subsystems execution, it is possible to poll a terminal when there is no need for information from that terminal. If the terminal sends a message in response to a poll when no data is being requested, the HP 1000 discards the message, causing the data to be lost, and treats it as an asynchronous interrupt. A break-mode prompt is then sent to the terminal by the host.

The terminal must determine that the host is ready to receive a message in order to ensure that messages are properly handled by the host. This is done by checking all messages from the host (CREAD until queue is empty) and not transmitting (CWRITE) until a prompt message or its equivalent has been received (unless you want to enter break-mode operation). Since the HP 1000 does not generate a consistent prompt message for all programs and subsystems, it is easiest to use cooperating programs to generate a predictable prompt. If your application requires interaction with other subsystems, prompts can usually be most easily identified by the ABSENCE of the sequence: $^{C}R^{L}F^{E}C\_$ at the end of a message. When a proper sequence has been identified, you are reasonably certain that the host is ready for your next message block.

Here is an example of host messages where a prompt is sent by the File Manager (FMGR) and answered by a RUN,EDITR command. Note that the prompt from the interactive editor fits the description of a prompt because a line-feed is not included after the carriage-return in the sequence.

:^{E}C_	Prompt is sent by FMGR to terminal.
RU,EDITR	EDITR Run command is sent to host.
SOURCE FILE NAME?$^{C}R^{L}F^{E}C$_	File name message is sent by the host, followed by a prompt sequence which has no line-feed. Sequence is different from FMGR prompt.
$^{C}R/^{B}L^{E}C$_	

Whenever an unexpected message from a terminal is received by RTE, it is treated as an asynchronous interrupt which terminates normal communication with that terminal. A break-mode prompt is sent to the terminal by RTE, and the next message is expected to be a valid break-mode command. If the the message is not a valid command (such as data in a file being transferred), the data is discarded, and an error message is sent to the terminal. If, in the meantime, the cooperating program or subsystem generates an input request, the next data block is sent to the proper destination, but is out of sequence because at least one block has been lost. You can prevent such data losses and the mass confusion that usually ensues (especially during high-speed file transfers to the host), by disabling auto-poll on the HP 1000 data link interface. With auto-poll OFF, no polls are sent to your terminal unless the host is prepared to receive data.

Disabling Auto-poll on the HP 1000

To operate with auto-poll OFF, log on to the network host, disable auto-poll, perform all datacomm activities and file transfers, enable auto-poll, then log off. **If you don't enable auto-poll at the end of a session, polling is suspended to your terminal after log-off, and you cannot reestablish communication with the host unless polling is restored from another terminal or the network host System Console.**

The auto-poll ON/OFF commands are:

CN,LU#,23B,101401B	Auto-poll OFF[1]
CN,LU#,23B,001401B	Auto-poll ON[1]

where LU# us the logical unit number assigned to your terminal.

When auto-poll is disabled, no polls are sent to your terminal unless an input request is initiated by the cooperating program or subsystem on the network host. When the request is made, a poll is scheduled, and polling continues until a reply is received from the terminal. When the reply is received, and acknowledged, polling is suspended until the next input is scheduled. Operating with auto-poll OFF is especially useful when transferring files TO the HP 1000. Otherwise, in most applications, it is practical to leave auto-poll ON.

1 The File Manager CN (Control) command parameters for the multipoint interface are described in more detail in the 91730A Multipoint Terminal Interface Subsystem User's Guide (91730-90002).

Prevention of Data Loss on the HP 3000
Neither the HP 1000 nor the HP 3000 provide a DC1 poll character when they are ready for data inputs from DSN/DL. The HP 3000, like the HP 1000, also discards data if it has not requested the transfer. Since the HP 3000 does not provide an auto-poll disable command, you must interpret messages from the HP 3000 to determine that it is ready for the next data block before you transmit the block.

Secondary Channel, Half-duplex Communication

Half-duplex telecommunications links frequently use secondary channel communication to control data transmission and provide for proper line turn-around. This is done by using Secondary Request-to-send (SRTS) and Secondary Data Carrier Detect (SDCD) modem signals.

Consider two devices communicating with each other: Each connects to the datacomm link, then waits for SDCD to become active (true). As each device connects to the line, Secondary Request-to-send is enabled, causing each modem to activate its secondary carrier output. The Secondary Data Carrier Detect is, in turn, activated by each modem as it receives the secondary data carrier from the other end.

When communication begins, the first device to transmit (assumed to be your computer, in this case) clears its Secondary Request-to-send modem line. This removes the secondary data carrier from the line, causing the other modem to clear SDCD to its terminal or computer, telling it that you have the line. (The modems also maintain proper line switching and prevent timing conflicts so both ends don't try to get the line simultaneously.) The other device receives data, and must not attempt to transmit until you relinquish control of the line as indicated by SDCD true. After you finish transmitting, you must again activate SRTS so that SDCD can be activated to the other device, allowing it to use the line if it has a message.

Communication Between Desktop Computers

Two desktop computers can be connected, directly, or by use of modems. DC1/DC3 handshake protocol can be used conveniently to enable each computer to transmit at will without risk of buffer or queue overruns. To ensure proper operation, the following guidelines apply:

- Set up IOControl Register 22 with a value of 5. This allows both computers to act either as host or terminal in any given situation, depending on which one initiates the action.
- Set up IOControl Registers 26 and 27 for DC1 and DC3 respectively, or use two other characters if necessary.
- Data to be transmitted must NOT contain any characters matching the contents of IOControl Register 26 or 27. This prevents the receiving interface from confusing data with control characters.
- If both computers attempt to transmit large amounts of data at the same time, a lock-up condition may result where each side is waiting for the other to empty its buffers.

Cable and Adapter Options and Functions

The HP 98628A Datacomm Interface is available with RS-232C DTE and DCE cable configurations, or it can be connected to various modems or adapters for other applications.

DTE and DCE Cable Options

DTE and DCE cable options are designed to simplify connecting two desktop computers without the use of modems. The DTE cable (male RS-232 connector) is configured to make the datacomm interface look like standard data terminal equipment when it is connected to an RS-232C modem. The DCE cable (female RS-232 connector) is configured so that it eliminates the need for modems in a direct connection. When you connect two computers to each other in a direct non-modem connection, both datacomm interfaces are functionally identical. The DCE cable acts as an adapter so that both interfaces behave exactly as they would if they were connected to a pair of modems by means of DTE cables.

Several signal lines are rerouted in the DCE cable so that, in direct connections, outputs from one interface are connected to the corresponding inputs on the other interface. Certain outputs on each interface are also connected to inputs on the same card by "loop-back" connections in the DCE cable.

The schematic diagram in this section shows two datacomm interfaces directly connected through a DTE-DCE cable pair. Note that the DCE cable wiring complements the DTE cable so that output signals are properly routed to their respective destinations. Signal names at the RS-232C connector interface are the same as the signal names for the DTE interface. However, because the DCE cable adapts signal paths, the signal name at the RS-232C connector does not necessarily match the signal name at the DCE interface. Connector pin numbers are included in the diagram for your convenience.

RS-232C DTE (male) Cable Signal Identification Tables

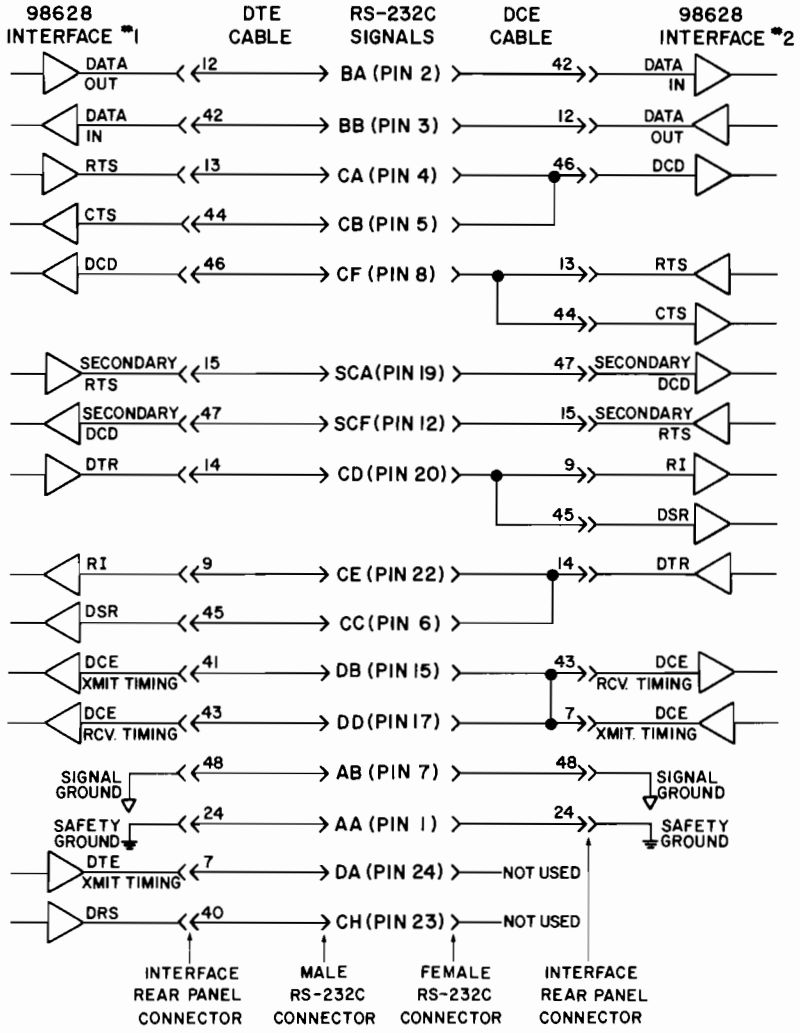
Signal		Interface	RS-232C			
RS-232C	V.24	Pin #	Pin #	Mnemonic	I/O	Function
AA	101	24	1	–	–	Safety Ground
BA	103	12	2		Out	Transmitted Data
BB	104	42	3		In	Received Data
CA	105	13	4	RTS	Out	Request to Send
CB	108	44	5	CTS	In	Clear to Send
CC	107	45	6	DSR	In	Data Set Ready
AB	102	48	7	–	–	Signal Ground
CF	109	46	8	DCD	In	Data Carrier Detect
SCF (OCR2)	122	47	12	SDCD	In	Secondary DCD
DB	114	41	15		In	DCE Transmit Timing
DD	115	43	17		In	DCE Receive Timing
SCA (OCD2)	120	15	19	SRTS	Out	Secondary RTS
CD	108.1	14	20	DTR	Out	Data Terminal Ready
CE (OCR1)	125	9	22	RI	In	Ring Indicator
CH (OCD1)	111	40	23	DRS	Out	Data Rate Select
DA	113	7	24		Out	Terminal Transmit Timing

Optional Circuit Driver/Receiver Functions

Two optional drivers and receivers are used with the RS-232C cable options. Their functions are as follows:

Drivers		Receivers	
Name	**Function**	**Name**	**Function**
OCD1	Data Rate Select	OCR1	Ring Indicator
OCD2	Secondary Request-to-send	OCR2	Secondary Data Carrier Detect
OCD3	Not used		
OCD4	Not used		

OCD2 is used for autodial pulsing in the HP 13265A Modem. None of the optional drivers and receivers are used for Data Link and Current Loop Adapters.



DTE/DCE Interface Cable Wiring

HP 98628 Datacomm Interface IOStatus and IOControl Register Summary

PASCAL Register Map - Control Registers

Register =	Use
000 .. 127	Buffered Control - Queued up with data
257 .. 383	Direct Control - Occurs immediately (meaning is the same as buffered ctl register + 256)
512	Immediate transfer in Abort
513	Immediate transfer out Abort

Unless indicated otherwise, the Status Register returns the current value for a given parameter; the Control Register sets a new value.

Register	Function
0	Control: Interface Reset; Status: Interface Card ID
1 (Status only)	Hardware Interrupt Status: 1 = Enabled, 0 = Disabled
2 (Status only)	Datacomm activity: 0 = inactive, 1 = ENTER in process, 2 = OUTPUT in process
3	Select Protocol: 1 = Async, 2 = Data Link
4 (Status only)	Interrupt Status. Interrupt operations are not currently supported at a user level in Pascal.
5	Control: Terminate transmission; Status: Inbound queue status
6	Control: Send BREAK to remote; Status: 1 = BREAK pending
7 (Status only)	Current modem receiver line states
8	Modem driver line states
9 (Status only)	Control block TYPE
10 (Status only)	Control block MODE
11 (Status only)	Available outbound queue space
12	Control: Connect/Disconnect line; Status: Line connection status
13	Interrupt mask. Interrupt operations are not currently supported at a user level in Pascal.
14	Control Block mask
15	Modem line interrupt mask. Interrupt operations are not currently supported at a user level in Pascal.
16	Connection timeout limit
17	No Activity timeout limit
18	Lost Carrier timeout limit
19	Transmit timeout limit
20	Async: Transmit baud rate (line speed) Data Link: Set Transmit/Receive baud rate (line speed)
21	Async: Incoming (receiver) baud rate (line speed) Data Link: GID address (0 thru 26 corresponds to "@" thru "Z")
22	Async: Protocol handshake type Data Link: DID address (0 thru 26 corresponds to "@" thru "Z")
23	Hardware handshake type: ON/OFF, HALF/FULL duplex, Modem/Non-modem

Register	Function
24	Async: Control Character mask Data Link: Block Size limit
25 (Status only)	Number of received errors since last interface reset
26	Async: First protocol character (ACK/DC1) Data Link: NAKs received since last interface reset

Registers 27-35, 37, and 39 are used with Async protocol only. They are not accessible during Data Link operation.

27	Second protocol handshake character (ENQ/DC3)
28	Number of characters in End-of-line sequence
29	First character in EOL sequence
30	Second character in EOL sequence
31	Number of characters in PROMPT sequence
32	First character in PROMPT sequence
33	Second character in PROMPT sequence
34	Data bits per character excluding start, stop and parity
35	Stop bits per character (0 = 1, 1 = 1.5, and 2 = 2 stop bits)
36	Parity sense: 0 = NONE, 1 = ODD, 2 = EVEN, 3 = ZERO, 4 = ONE Data Link: 0 = NONE (HP 1000 host), 1 = ODD (HP 3000 host)
37	Inter-character time gap in character times (Async only)
38 (Status only)	Transmit queue status (1 = empty)
39	BREAK time in character times (Async only)
125 (Control only)	Abort both input and output transfers.
512 (Control only)	Immediate transfer in Abort.
513 (Control only)	Immediate transfer out Abort.

HP 98628 Data Communications Interface IOSTATUS and IOCONTROL Registers

General Notes: Control registers accept values in the range of zero through 255. Some registers require specified values, as indicated. Illegal values or values less than zero or greater than 255, cause ERROR 327. Accessing a non-existent register generates ERROR 326.

Reset value, shown for various Control Registers, is the default value used by the interface after a reset or power-up until the value is overridden by an IOCONTROL procedure.

Status 0 Card Identification
Value returned: 52 (if 180 is returned, check select code switch cluster and make sure switch R is ON).

Control 0 Card Reset
Any value, 1 thru 255, resets the card. Immediate execution. Data in queues is destroyed.

Status 1 Hardware Interrupt Status (not used in most applications)
1 = Enabled 0 = Disabled

Status 2 Datacomm Activity
0 = No activity pending on this select code.
Bit 0 set: input in process.
Bit 1 set: output in process.
(Non-zero ONLY during multi-line function calls.)

Status 3 Current Protocol Identification:
1 = Async, 2 = Data Link Protocol

Control 3 Protocol to be used after next card reset `(CONTROL Sc,0;1)`
1 = Async Protocol 2 = Data Link Protocol
This register overrides default switch configuration.

Status 4 Interrupt status. Interrupt operations are not currently supported at a user level in Pascal.

Status 5 Inbound queue status

Value	Interpretation
0	Queue is empty
1	Queue contains data but no control blocks
2	Queue contains one or more control blocks but no data
3	Queue contains both data and one or more control blocks

Control 5 Terminate Transmission

Data Link: Sends previous data as a single block with an ETX terminator, then idles the line with an EOT.

Async: Tells card to turn half-duplex line around. Does nothing when line is full-duplex. The next data output automatically regains control of the line by raising the RTS (request-to-send) modem line.

Status 6 Break status: 1 = BREAK transmission pending, 0 = no BREAK pending.

Control 6 Send Break; causes a Break to be sent as follows:

Data Link Protocol: Send Reverse Interrupt (RVI) reply to inbound block, or CN character instead of data in next outbound block.

Async Protocol: Transmit Break. Length is defined by Control Register 39.

Note that the value sent to the register is arbitrary.

Status 7 Modem receiver line states (values shown are for male cable connector option for connection to modems).

Bit 0: Data Mode (Data Set Ready) line
Bit 1: Receive ready (Data Carrier Detect line)
Bit 2: Clear-to-send (CTS) line
Bit 3: Incoming call (Ring Indicator line)
Bit 4: Depends on cable option or adapter used

Status 8 Returns modem driver line states.

Control 8 Sets modem driver line states (values shown are for male cable connector option for connection to modems).

Bit 0: Request-to-send (RS or RTS) line — 1 = line set (active)
Bit 1: Data Terminal Ready (DTR) line — 0 = line clear (inactive)
Bit 2: Driver 1: Data Rate Select
Bit 3: Driver 2: Depends on cable option or adapter used
Bit 4: Driver 3: Depends on cable option or adapter used
Bit 5: Driver 4: Depends on cable option or adapter used
Bits 6,7: Not used

Reset value = 0 prior to connect. Post-connect value is handshake dependent.

Note that RTS line cannot be altered (except by OUTPUT or OUTPUT...END) for half-duplex modem connections.

Status 9 Returns control block TYPE if last input terminated on a control block. See Status Register 10 for values.

Status 10 Returns control block MODE if last input terminated on a control block.

Async Protocol Control Blocks

Type	Mode	Interpretation
250	1	Break received (Channel A)
251	1[1]	Framing error in the following character
251	2[1]	Parity error in the following character
251	3[1]	Parity and framing errors in the following character
252	1	End-of-line terminator detected
253	1	Prompt received from remote
0	0	No Control Block encountered

1 Parity/framing error control blocks are not generated when characters with parity and/or framing errors are replaced by an underscore (_) character.

Data Link Protocol Control Blocks

Type	Mode	Interpretation
254	1	Preceding block terminated by ETB character
254	2	Preceding block terminated by ETX character
253[2]	—	(see following table for Mode interpretation)
0	0	No Control Block encountered.

Mode Bit(s)	Interpretation
0	1 = Transparent data in following block
	0 = Normal data in following block
2,1	00 = Device select
	01 = Group select
	10 = Line select
3	1 = Command channel
	0 = Data channel

Status 11 Returns available outbound queue space (in bytes), provided there is sufficient space for at least three control blocks. If not, value is zero.

Status 12 Datacomm Line connection status

Value	Interpretation
0	Disconnected
1	Attempting Connection
2	Dialing
3	Connected[1]
4	Suspended
5	Currently receiving data (Data Link only)
6	Currently transmitting data (Data Link only)

Note

When the datacomm line is suspended, ABORT_SERIAL, or IORESET must be executed before the line can be reconnected.

Reset value = 0 if R on interface select code switch cluster is ON (1).

Control 12 Connects, disconnects, initiates auto-dialing as follows:

Value	Interpretation
0	Disconnects
1	Connects
2	Initiates

Status 13 Interrupt mask. Interrupt operations are not currently supported at a user level in Pascal.

Control 13 Interrupt mask. Interrupt operations are not currently supported at a user level in Pascal.

2 This type is used primarity in specialized applications.

1 When using Data Link: Connected - datacomm idle

Status 14 Returns current Control Block mask.

Control 14 Sets Control Block mask. Control block information is queued sequentially with incoming data as follows:

Bit	Value	Async Control Block Passed	Data Link Control Block Passed
0	1	Prompt position	Transparent/Normal Mode[1]
1	2	End-of-line position	ETX Block Terminator[2]
2	4	Framing and/or Parity error[3]	ETB Block Terminator[2]
3	8	Break received	

Reset Value: **0** (Control Blocks disabled) **6** (ETX/ETB Enabled)

Bits 4, 5, 6, and 7 are not used.

Status 15 Modem line interrupt mask. Interrupt operations are not currently supported at a user level in Pascal.

Control 15 Modem line interrupt mask. Interrupt operations are not currently supported at a user level in Pascal.

Status 16 Returns current connection timeout limit.

Control 16 Sets Attempted Connection timeout limit.
Acceptable values: 1 thru 255 seconds. 0 = timeout disabled.
Reset Value = 25 seconds

Status 17 Returns current No Activity timeout limit.

Control 17 Sets No Activity timeout limit.
Acceptable values: 1 thru 255 minutes. 0 = timeout disabled.
Reset Value = 10 minutes (disabled if Async, non-modem handshake).

Status 18 Returns current Lost Carrier timeout limit.

Control 18 Sets Lost Carrier timeout limit in units of 10 ms.
Acceptable values: 1 thru 255. 0 = timeout disabled.
Reset Value = 40 (400 milliseconds)

Status 19 Returns current Transmit timeout limit.

Control 19 Sets Transmit timeout limit (loss of clock or CTS not returned by modem when transmission is attempted).
Acceptable values: 1 thru 255.0 = timeout disabled.
Reset Value = 10 seconds

1 Transparent/Normal format identification control block occurs at the BEGINNING of a given block of data in the receive queue.
2 ETX and ETB Block Termination identification control blocks occur at the END of a given block of data in the receive queue.
3 This control block precedes each character containing a parity or framing error.

Status 20 Returns current transmission speed (baud rate). See table for values.
Control 20 Sets transmission speed (baud rate) as follows:

Register Value	Baud Rate	Register Value	Baud Rate
0	External Clock	8	600
*1	50	9	1200
*2	75	10	1800
*3	110	11	2400
*4	134.5	12	3600
*5	150	13	4800
*6	200	14	9600
7	300	15	19200

* Async only. These values cannot be used with Data Link. These values set transmit speed ONLY for Async; transmit AND receive speed for Data Link. Default value is defined by the interface card configuration switches.

Status 21 Protocol dependent. Returns receive speed (Async) or GID address (Data Link) as specified by Control Register 21.
Control 21 Protocol dependent. Functions are as follows:

Data Link: Sets Group IDentifier (GID) for terminal. Values 0 thru 26 correspond to identifiers @, A, B,...Y, Z, respectively. Other values cause an error. Default value is 1 ("A").

Async: Sets datacomm receiver speed (baud rate). Values and defaults are the same as for Control Register 20.

Status 22 Protocol dependent. Returns DID (Data Link) or protocol handshake type (Async) as specified by Control Register 22.
Control 22 Protocol dependent. Functions are as follows:

Data Link: Sets Device IDentifier (DID) for terminal. Values are the same as for Control Register 21. Default is determined by interface card configuration switches.

Async: Defines protocol handshake type that is to be used.

Value	Handshake type
0	Protocol handshake disabled
1	ENQ/ACK with desktop computer as the host
2	ENQ/ACK, desktop computer as a terminal
3	DC1/DC3, desktop computer as host
4	DC1/DC3, desktop computer as a terminal
5	DC1/DC3, desktop computer as both host and terminal

Status 23 Returns current hardware handshake type.
Control 23 Sets hardware handshake type as follows:

0 = Handshake OFF, non-modem connection.
1 = FULL-DUPLEX modem connection.
2 = HALF-DUPLEX modem connection.
3 = Handshake ON, non-modem connection.
Reset Value is determined by interface configuration switches.

Status 24 Protocol dependent. Returns value set by preceding IOCONTROL procedure to Control Register 24.

Control 24 Protocol dependent. Functions as follows:

Data Link protocol: Set outbound block size limit.

Value	Block size	Value	Block size
0	512 bytes	4	8 bytes
1	2 bytes	⋮	⋮
2	4 bytes		
3	6 bytes	255	510 bytes

Reset outbound block size limit = 512 bytes

Async Protocol: Set mask for control characters included in receive data message queue.
Bit set: transfer character(s).
Bit cleared: delete character(s).

Bit set	Value	Character(s) passed to receive queue
0	1	Handshake characters (ENQ, ACK, DC1, DC3)
1	2	Inbound End-of-line character(s)
2	4	Inbound Prompt character(s)
3	8	NUL (CHR(0))
4	16	DEL (CHR(127))
5	32	CHR(255)
6	64	Change parity/framing errors to underscores (_) if bit is set.
7	128	Not used

Reset value = 127 (bits 0 thru 6 set)

Status 25 Returns number of received errors since power up or reset.

Note

Control Registers 26 through 35, Status Registers 27 through 35, and Control and Status Registers 37 and 39 are used for ASYNC protocol ONLY. They are not available during Data Link operation.

Status 26 Protocol dependent

Data Link protocol: Returns number of transmit errors (NAKs received) since last interface reset.

Async protocol: Returns first protocol handshake character (ACK or DC1).

Control 26 (Async only) Sets first protocol handshake character as follows:
6 = ACK, 17 = DC1. Other values used for special applications only. **Reset value = 17** (DC1). Use ACK when Control Register 22 is set to 1 or 2. Use DC1 when Control Register 22 is set to 3, 4, or 5.

Status 27 (Async only) Returns second protocol handshake character.

Control 27 (Async only) Sets second protocol handshake character as follows:
5 = ENQ, 19 = DC3. Other values used for special applications only. **Reset value = 19** (DC3). Use ENQ when Control Register 22 is set to 1 or 2. Use DC3 when Control Register 22 is set to 3, 4, or 5.

Status 28 Returns number of characters in inbound
(Async only) End-of-line delimiter sequence.

Control 28 Sets number of characters in End-of-line delimiter sequence
(Async only) Acceptable values are 0 (no EOL delimiter), 1, or 2. **Reset Value = 2**

Status 29 Returns first End-of-line character.
(Async only)

Control 29 Sets first End-of-line character. **Reset Value = 13** (carriage return)
(Async only)

Status 30 Returns second End-of-line character.
(Async only)

Control 30 Sets second End-of-line character. **Reset Value = 10** (line feed)
(Async only)

Status 31 Returns number of characters in Prompt sequence.
(Async only)

Control 31 Sets number of characters in Prompt sequence.
(Async only) Acceptable values are 0 (Prompt disabled), 1 or 2.
Reset Value = 1

Status 32 Returns first character in Prompt sequence.
(Async only)

Control 32 Sets first character in Prompt sequence.
(Async only) **Reset Value = 17** (DC1)

Status 33 Returns second character in Prompt sequence.
(Async only)

Control 33 Sets second character in Prompt sequence.
(Async only) **Reset Value = 0** (null)

Status 34 Returns the number of bits per character.
(Async only)

Control 34 Sets the number of bits per character as follows:
(Async only) 0 = 5 bits/character 2 = 7 bits/character
1 = 6 bits/character 3 = 8 bits/character)
When 8 bits/char, parity must be NONE, ODD, or EVEN.
Reset Value is determined by interface card default switches.

Status 35 Returns the number of stop bits per character.
(Async only)

Control 35 Sets the number of stop bits per character as follows:
(Async only) 0 = 1 stop bit 1 = 1.5 stop bits 2 = 2 stop bits
Reset Value: 2 stop bits if 150 baud or less, otherwise 1 stop bit.
Reset Value is determined by interface configuration switch settings.

Status 36 Returns current Parity setting.

Control 36 Sets Parity for transmitting and receiving as follows:

Data Link Protocol: 0 = NO Parity; Network host is HP 1000 Computer.
1 = ODD Parity; Network host is HP 3000 Computer.
Reset Value = 0

Async Protocol : 0 = NONE; no parity bit is included with any characters.
1 = ODD; Parity bit SET if there is an EVEN number of "1"s in the character body.
2 = EVEN; Parity bit OFF if there is an ODD number of "1"s in the character body.
3 = "0"; Parity bit is always ZERO, but parity is not checked.
4 = "1"; Parity bit is always SET, but parity is not checked.

Default is determined by interface configuration switches. If 8 bits per character, parity must be NONE, ODD, or EVEN.

Status 37 (Async only) Returns inter-character time gap in character times.

Control 37 (Async only) Sets inter-character time gap in character times.
Acceptable values: 1 thru 255 character times.
0 = No gap between characters. **Reset Value = 0**

Status 38 Returns Transmit queue status.
If returned value = 1, queue is empty, and there are no pending transmissions.

Status 39 (Async only) Returns current Break time (in character times).

Control 39 (Async only) Sets Break time in character times.
Acceptable values are: 2 thru 255. **Reset Value = 4.**

Control 125 Abort both input and output transfers.

Control 512 Immediate transfer in Abort.

Control 513 Immediate transfer out Abort.

Chapter 14
The GPIO Interface

Introduction

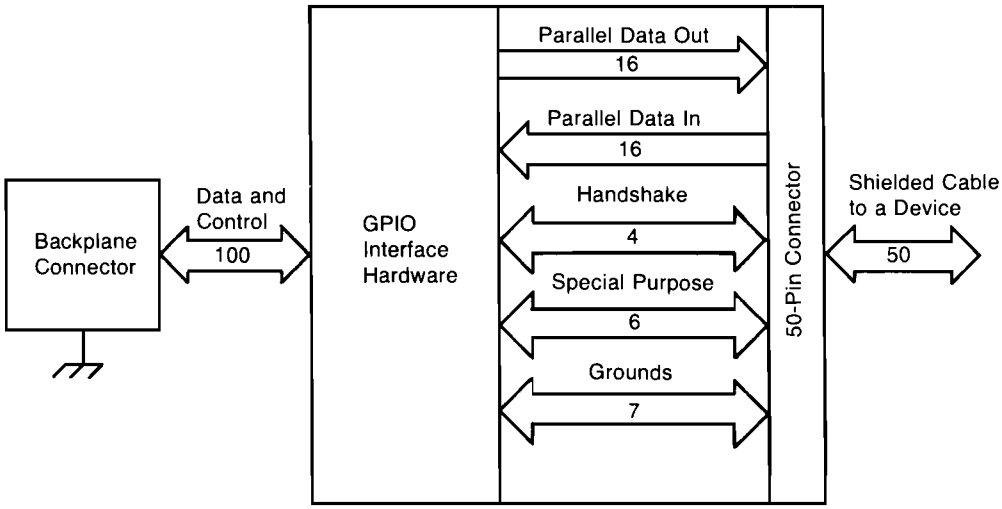
This chapter should be used in conjunction with the *HP 98622A GPIO Interface Installation* manual. **The best way to use these two documents is to read this chapter before attempting to configure and connect the interface** according to the directions given in the installation manual. The reason for this order of use is that knowing how the interface works and how it is driven by Pascal programs will help you to decide how to connect it to your peripheral device.

The HP 98622 Interface is a very flexible parallel interface that allows you to communicate with a variety of devices. The interface sends and receives up to 16 bits of data with a choice of several handshake methods. The interface is known as the General-Purpose Input/Output (GPIO) Interface. This chapter describes the use of the interface's features from Pascal programs.

Interface Description

The main function of any interface is to transfer data between the computer and a peripheral device. This section briefly describes the interface lines and how they function. Using the lines from Pascal programs is more fully described in subsequent sections.

The GPIO Interface provides **32 lines for data input and output**: 16 for input (DI0 — DI15), and 16 for output (DO0 — DO15).



Block Diagram of the GPIO Interface

Three lines are dedicated to **handshaking** the data from source to destination device. The Peripheral Control line (PCTL) is controlled by the interface and is used to initiate data transfers. The Peripheral Flag line (PFLG) is controlled by the peripheral device and is used to signal the peripheral's readiness to continue the transfer process.

Four general-purpose lines are available for any purpose that you may desire; two are controlled by the computer and sensed by the peripheral (CTL0 and CTL1), and two are controlled by the peripheral device and sensed by the computer (STI0 and STI1).

Both Logic Ground and Safety Ground are provided by the interface. Logic Ground provides the reference point for signals, and Safety Ground provides earth ground for cable shields.

Interface Configuration

This section presents a brief summary of selecting the interface's configuration-switch settings. It is intended to be used as a checklist and to begin to acquaint you with programming the interface. **Refer to the installation manual for the exact location and setting of each switch.**

Interface Select Code

In Pascal, allowable interface select codes range from 8 through 31; codes 1 through 7 are already used for built-in interfaces. The GPIO interface has a factory default setting of 12, which can be changed by re-configuring the "SEL CODE" switches on the interface.

Hardware Interrupt Priority

Two switches are provided on the interface to allow selection of hardware interrupt priority. The switches allow hardware priority levels 3 through 6 to be selected. **Hardware priority** determines the order in which simultaneously occurring interrupt events are processed.

Data Logic Sense

The data lines of the interface are **normally low-true**; in other words, when the voltage of a data line is low, the corresponding data bit is interpreted to be a 1. This logic sense may be changed to high-true with the Option Select Switch. Setting the switch labeled "DIN" to the "0" position selects high-true logic sense of Data In lines. Conversely, setting the switch labeled "DOUT" to the "1" position inverts the logic sense of the Data Out lines. The default setting is "1" for both.

Data Handshake Methods

This section describes the data handshake methods available with the GPIO Interface. A general description of the handshake modes and clock sources is given first. A more detailed discussion of each handshake is then given to allow you to choose the handshake mode, clock source, and handshake-line logic sense that is compatible with your peripheral device.

As a brief review, a data handshake is a method of synchronizing the transfer of data from the sending to the receiving device. In order to use any handshake method, **the computer and peripheral device must be in agreement as to how and when several events will occur**. With the GPIO Interface, the following events must take place to synchronize data transfers; the first two are optional.

- The computer may optionally be directed to perform a one-time "OK check" of the peripheral before beginning to transfer any data.
- The computer may also optionally check the peripheral to determine whether or not the peripheral is "ready" to transfer data.
- The computer must indicate the direction of transfer and then initiate the transfer.
- During output operations, the peripheral must read the data sent from the computer while valid; similarly, the computer must clock the peripheral's data into the interface's Data In registers while valid during input operations.
- The peripheral must acknowledge that it has received the data.

Handshake Lines

The GPIO handshakes data with three signal lines. The Input/Output line, **I/O**, is driven by the computer and is used to signal the direction of data transfer. The Peripheral Control line, **PCTL**, is also driven by the computer and is used to initiate all data transfers. The Peripheral Flag line, **PFLG**, is driven by the peripheral and is used to acknowledge the computer's requests to transfer data.

Handshake Logic Sense

Logic senses of the PCTL and PFLG lines are selected with switches of the same name. The logic sense of the I/O line is High for input operations and Low for output operations; this logic sense cannot be changed. The available choices of handshake logic sense and handshake modes allow nearly all types of peripheral handshakes to be accommodated by the GPIO Interface.

Handshake Modes

There are two general handshake modes in which the PCTL and PFLG lines may be used to synchronize data transfers: Full-Mode and Pulse-Mode Handshakes. If the peripheral uses pulses to handshake data transfers **and** meets certain hardware timing requirements, the Pulse-Mode Handshake may be used. The Full-Mode Handshake should be used if the peripheral does not meet the Pulse-Mode timing requirements.

The handshake mode is selected by the position of the "HSHK" switch on the interface, as described in the installation manual. Both modes are more fully described in subsequent sections.

Data-In Clock Source

Ensuring that the data are valid when read by the receiving device is slightly different for output and input operations. During outputs, the interface generally holds data valid while PCTL is in the Set state, so the peripheral must read the data during this period. During inputs, the data must be held valid by the peripheral until the peripheral signals that the data are valid (which clocks the data into interface Data In registers) or until the data is read by the computer. The point at which the data are valid is signalled by a transition of PFLG. The PFLG transition that is used to signal valid data is selected by the "CLK" switches on the interface. Subsequent diagrams and text further explain the choices.

Peripheral Status Check

Many peripheral devices are equipped with a line which is used to indicate the device's current "OK-or-Not-OK" status. If this line is connected to the Peripheral Status line (PSTS) of the GPIO Interface, and the computer determines the status of the peripheral device by checking the state of PSTS. The logic sense of this line may be selected by setting the "PSTS" switch.

The computer performs a check of the Peripheral Status line (PSTS) **before initiating any transfers** as part of the data-transfer handshake. If PSTS indicates "Not OK," an error is reported with ioe_result set to 21; otherwise, the transfer proceeds normally. This feature is available with both Full-Mode and Pulse-Mode Handshakes. See "Using the PSTS Line" for further details.

Full-Mode Handshakes

The Full-Mode Handshake mode is described first for two reasons. The first reason is that the PCTL and PFLG transitions must always occur in the order shown, so only one sequence of peripheral handshake responses needs to be shown. Secondly, this mode will generally work when the Pulse-Mode Handshake may not be compatible with the peripheral's handshake signals. The Pulse-Mode Handshake is described in the next section.

The following diagrams show the order of events of the Full-Mode output and input Handshakes. These drawings are not drawn to any time scale; only the order of events is important. The I/O line has been omitted to simplify the diagrams; in all cases, it is driven Low before any output is initiated by the computer and High before any input is initiated.

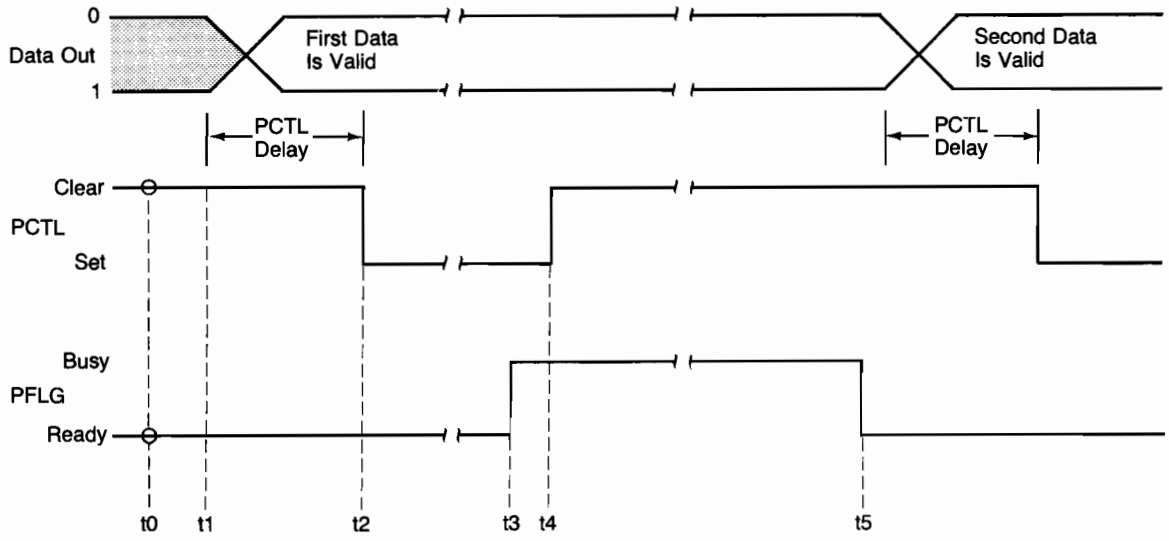


Diagram of Full-Mode OUTPUT Handshakes

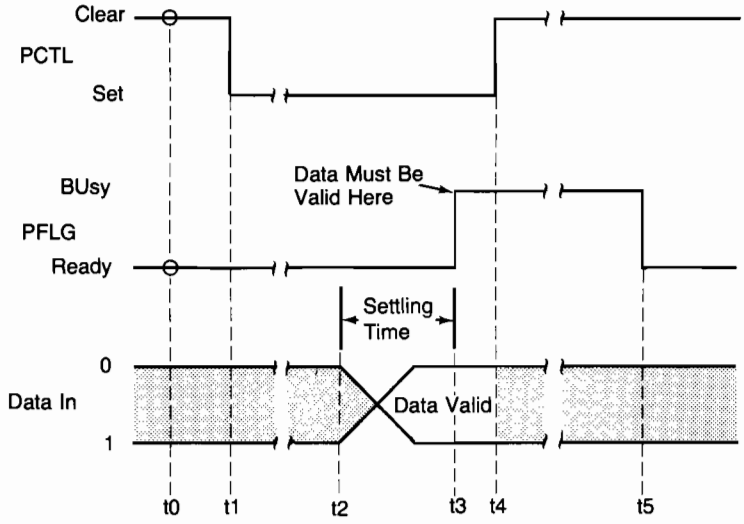
With Full-Mode Handshakes, the computer first checks to see that the peripheral device is Ready before initiating the transfer of each byte/word (t0); with this handshake mode, the peripheral indicates **Ready when both PCTL is Clear and PFLG is Ready**. If the peripheral does not indicate Ready, the computer waits until a Ready is indicated.

When a Ready is sensed, the computer places data on the Data Out lines (t1) and drives the I/O line Low (not shown). The interface then waits the PCTL Delay time before initiating the transfer by placing PCTL in the Set state (t2).

The peripheral acknowledges the computer's request by placing the PFLG line Busy (t3); this PFLG transition automatically Clears the PCTL line (t4). However, the computer cannot intitate further transfers until the peripheral is Ready with Full-Mode Handshake; the peripheral is not Ready until both PCTL is Clear and PFLG is Ready (t5).

The data on the Data Out lines is held valid from the time PCTL is Set until after the peripheral indicates Ready. The peripheral may read the data any time within this time period.

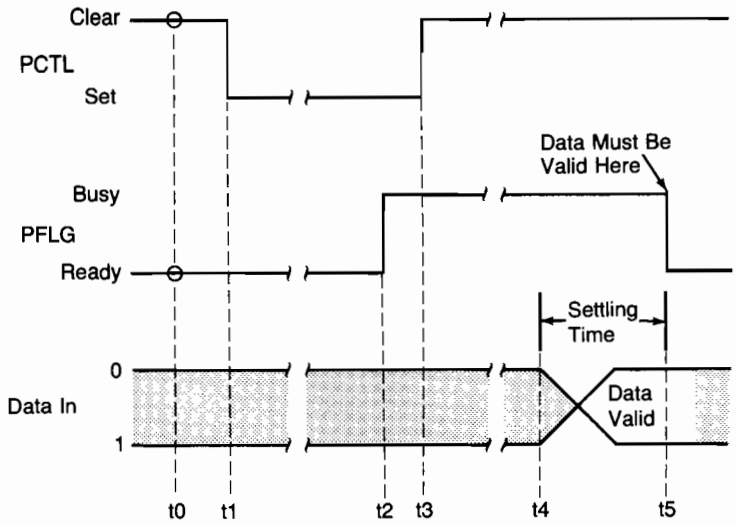
The PCTL and PFLG lines are used in the same manner in Full-Mode input Handshakes as in Full-Mode output Handshakes. However, there are three options available as to when the peripheral's data may be valid: at the Ready-to-Busy transition of PFLG (BSY clock source), at the Busy-to-Ready transition of PFLG (RDY clock source), and when the Data In lines are read with an IOSTATUS function (READ clock source). The first two of these options are shown in the following two diagrams.



Full-Mode Input Handshake with BSY Clock Source

As with Full-Mode output Handshakes, the computer first checks to see if the peripheral is Ready (t0); since PCTL is Clear and PFLG is Ready, the handshake may proceed. The computer places the I/O line in the High state (not shown) and then initiates the handshake by placing PCTL in the Set state (t1).

With the "BSY" clock source, the PFLG transition to the Busy state clocks the peripheral's data into the interface's Data-In registers; consequently, the peripheral must place data on the Data-In lines (t2), allowing enough time for the data to settle before placing PFLG in the Busy state (t3). This PFLG transition to the Busy state automatically Clears PCTL (t4). The next handshake may be initiated when PFLG is placed in the Ready state by the peripheral (t5).



Full-Mode Input Handshake with RDY Clock Source

As with other Full-Mode Handshakes, the computer first checks to see if the peripheral is ready (t0). Since PCTL is Clear and PFLG is Ready, the computer may drive the I/O line High (not shown) and initiate the handshake by placing PCTL in the Set state (t1).

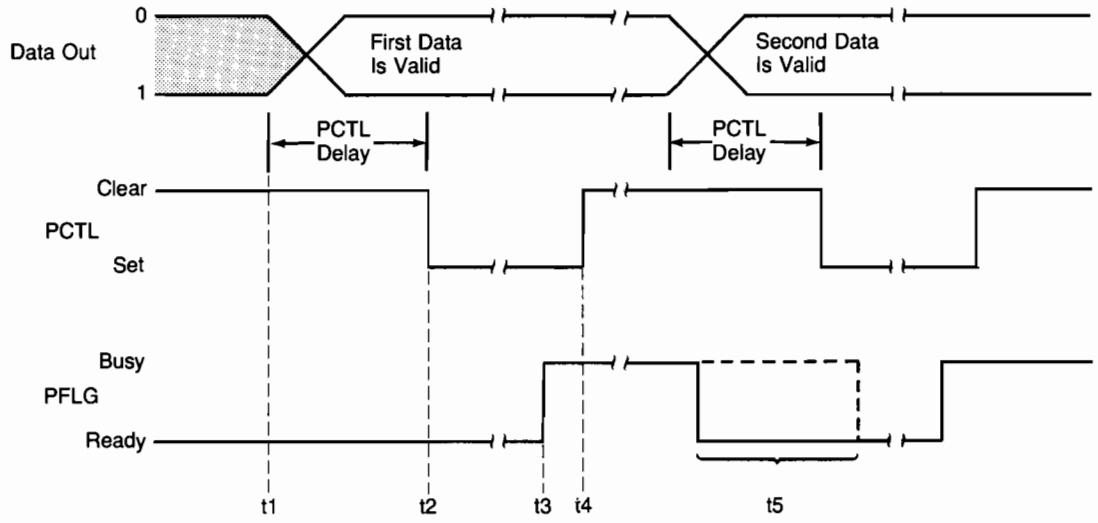
The peripheral may acknowledge by placing PFLG Busy (t2), which automatically Clears PCTL (t3). Unlike the previous example, this transition does not clock data into the interface Data-In registers. With the "RDY" clock source, the peripheral must place the data on the Data-In lines (t4), allowing enough time for the data to settle before placing PFLG in the Ready state (t5). The computer may then initiate a subsequent transfer.

Pulse-Mode Handshakes

The following drawings show the order of handshake-line events during Pulse-Mode Handshakes. Notice that the **main difference** between Full-Mode and Pulse-Mode Handshakes is that the **PFLG is not checked for Ready before the computer initiates Pulse-Mode Handshakes**; the computer may initiate a subsequent data transfer as soon as the PCTL line is Cleared by the Ready-to-Busy transition of PFLG.

Two cycles of data transfers are shown in these diagrams to illustrate that the computer need not wait for the PFLG = Ready indication with the Pulse-Mode Handshake. The first cycle shown in each diagram is a typical example of the first transfer of an I/O statement. The dashed PFLG line at the beginning of the second cycle shows that computer disregards whether or not PFLG is in the Ready state before the next transfer is initiated.

This absense of the PFLG check allows a **potentially higher data-transfer rate** than possible with the Full-Mode Handshake; however, in some cases, it also places additional timing restrictions on the peripheral's response time, as described in the text.

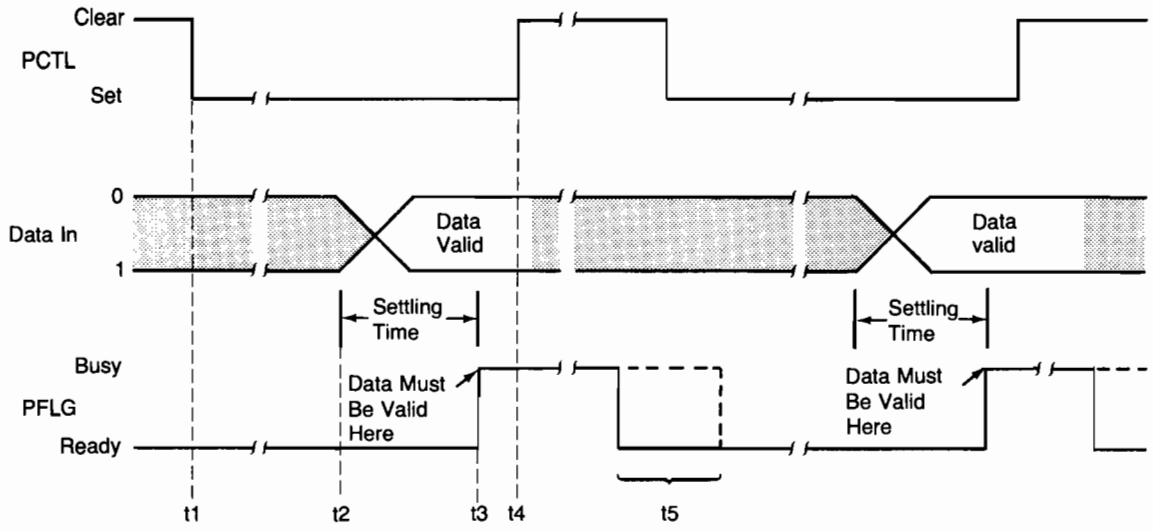


Busy Pulses With Pulse-Mode Output Handshake

The PFLG line is not checked for Ready before the computer drives the I/O line Low (not shown) and places data on the Data-Out lines (t1). A PCTL Delay time later, the interface initiates the transfer by placing PCTL in the Set state (t2).

The peripheral acknowledges by placing PFLG Busy (t3); this transition automatically Clears PCTL (t4). The dashed PFLG line shows that the computer may initiate another transfer any time after PCTL is Clear, possibly before the peripheral places PFLG in the Ready state (t5).

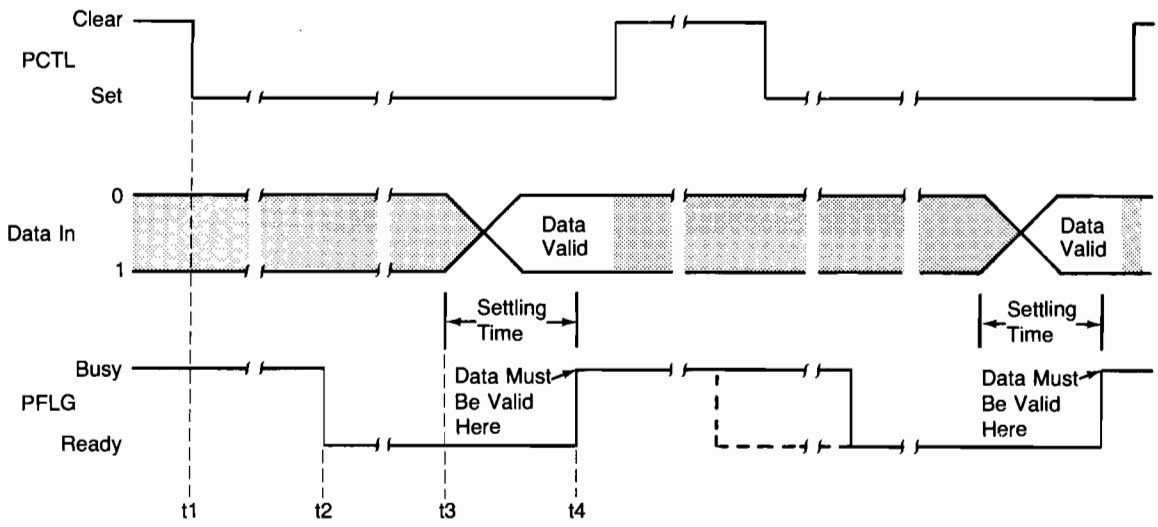
The Busy Pulse shown in the diagram is identical to the PFLG's response during the previous Full-Mode handshake; however, the Pulse-Mode Handshake works properly with this type of pulse **only** if the peripheral reads the data by the time PCTL is Clear (data should be read between t2 and t3). If the peripheral has not read the data by the time that PCTL is Clear, it might erroneously read the data for the second transfer, since the computer might have already changed the data and initiated the second transfer.



Busy Pulses With Pulse-Mode Input Handshakes (BSY Clock Source)

The computer does not have to check for PFLG to be Ready before placing I/O in the High state (not shown) and initiating the transfer by placing PCTL in the Set state (t1).

The peripheral must place data on the Data In lines (t2), allowing enough tome for the data to settle before placing PFLG in the Busy state (t3). This Ready-to-Busy transition of PFLG automatically Clears PCTL. The dashed PFLG signal shows that the next transfer may be initiated before PFLG indicates Ready.



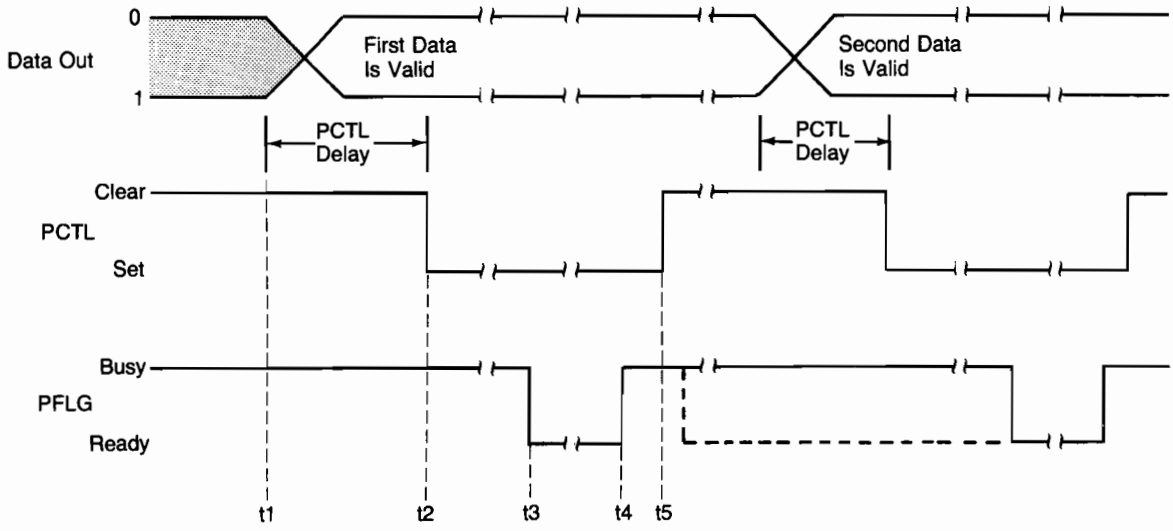
Busy Pulses With Pulse-Mode Input Handshakes (RDY Clock Source)

The computer does not have to check for PFLG to be Ready before placing I/O in the High state (not shown) and initiating the transfer by placing PCTL in the Set state (t1).

The peripheral must place data on the Data In lines (t2), allowing enough time for the data to settle before placing PFLG Busy (t3). This requirement **may seem contradictory**, since the clock source is the Busy-to-Ready transition of PFLG. However, with Pulse-Mode handshakes, the peripheral is assumed to be Ready whenever PCTL is Clear; consequently, the computer may read the data any time after PCTL is cleared by the Ready-to-Busy transition of PFLG. The PFLG transition to Busy Clears PCTL (t4), after which the peripheral may place PFLG Ready (t5).

Note

In order to use this type of pulse with the Pulse-Mode Handshake and RDY clock source, the peripheral must adhere to the stated timing restrictions.

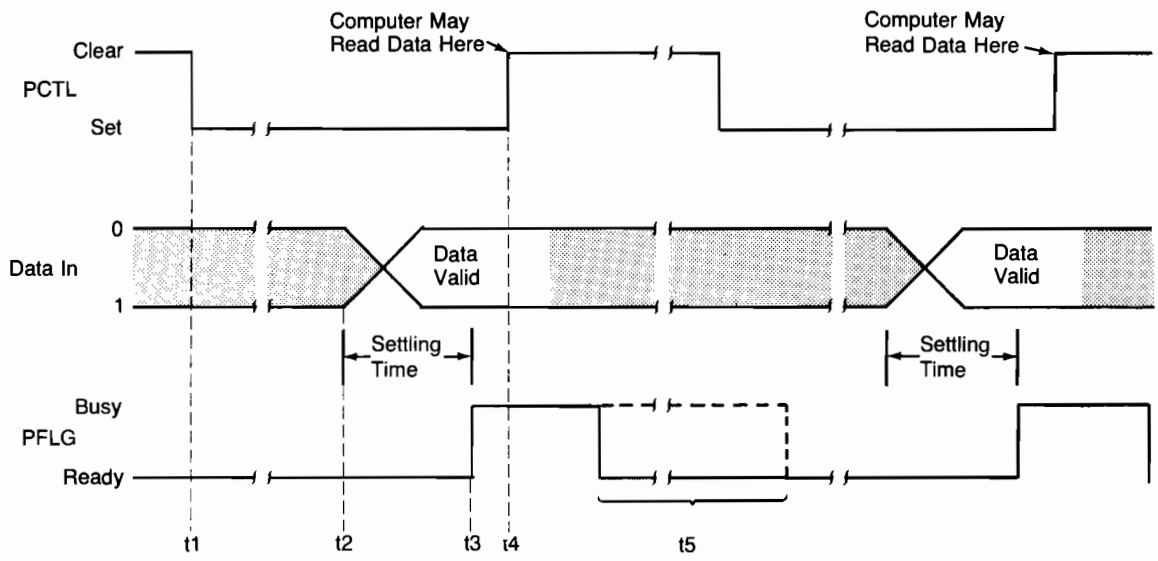


Ready Pulses With Pulse-Mode Output Handshakes

The PFLG line is not checked for Ready before the computer drives the I/O line Low (not shown) and places data on the Data Out lines (t1). A PCTL Delay time later the interface initiates the transfer by placing PCTL in the Set state (t2).

The peripheral later acknowledges by placing PFLG in the Ready state (t3). The handshake is completed by the peripheral placing PFLG in the Busy state (t4), which automatically Clears PCTL (t5).

If the peripheral uses the type of Ready pulses shown, either the Pulse-Mode handshake with default PFLG logic sense or Full-Mode handshake with inverted PFLG logic sense may be used. With this type of pulse, the data being output may be read by the peripheral as long as PCTL is Set.

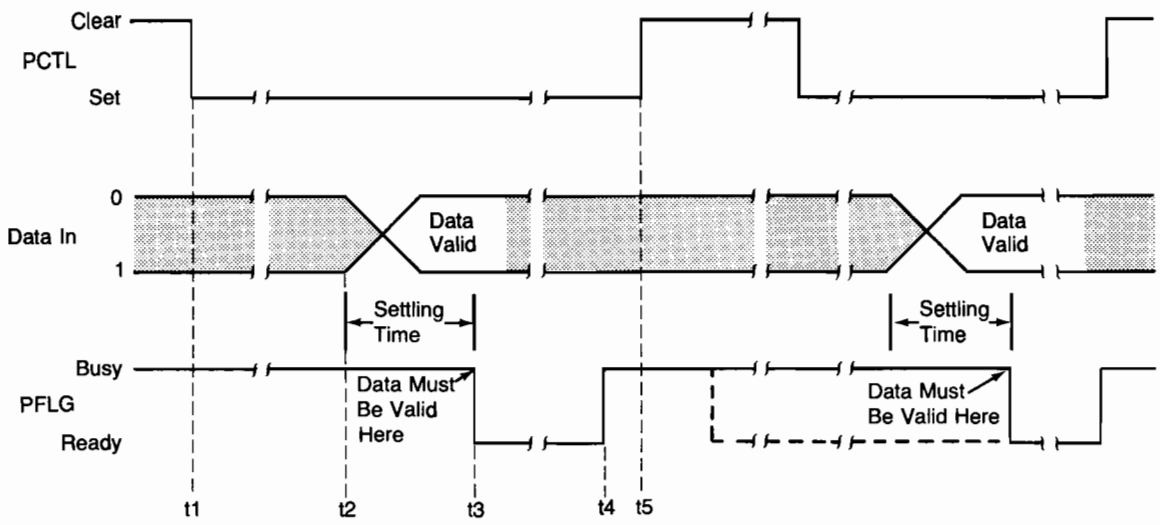


Ready Pulses With Pulse-Mode Input Handshakes (BSY Clock Source)

The computer does not have to check for PFLG to be Ready before placing I/O in the High state (not shown) and initiating the transfer by placing PCTL in the Set state (t1).

The peripheral acknowledges by placing PFLG in the Ready state (t2). The peripheral must place data on the Data In lines (t3), allowing enough time for the data to settle before placing PFLG in the Busy state (t4). With this type of pulse, events t2 and t3 may also occur in the reverse order.

The Ready-to-Busy transition of PFLG automatically Clears PCTL (t4). The dashed PFLG signal shows that the state of PFLG is not checked before the computer initiates a subsequent transfer.



Ready Pulses With Pulse-Mode Input Handshakes (RDY Clock Source)

The computer does not have to check for PFLG to be Ready before placing I/O in the High state (not shown) and initiating the transfer by placing PCTL in the Set state (t1).

The peripheral must place data on the Data In lines (t2), allowing enough time for the data to settle before placing PFLG Ready (t3). The peripheral places PFLG in the Busy state (t4), which automatically Clears PCTL (t5).

Interface Reset

The interface should always be reset before use to ensure that it is in a known state. All interfaces are automatically reset by the computer at certain times: when the computer is powered on, when the RESET key is pressed, and at other times including when the STOP or CLR I/O keys are pressed and when IOINITIALIZE and IOUNINITIALIZE are executed. The interface may be optionally reset at other times under control of Pascal programs. Two examples are as follows:

```
IORESET(12);


SC:=12;
IOCONTROL(Sc,1);
```

The following action is invoked whenever the GPIO Interface is reset:

- The Peripheral Reset line (PRESET) is pulsed Low for at least 15 microseconds.
- The PCTL line is placed in the Clear state.
- If the DOUT CLEAR jumper is installed, the Data Out lines are all cleared (set to logic 0).

The following lines are **unchanged** by a reset of the GPIO Interface:

- The CTL0 and CTL1 output lines.
- The I/O line.
- The Data Out lines, if the DOUT CLEAR jumper is not installed.

Outputs and Inputs through the GPIO

This section describes techniques for outputting and inputting data through the GPIO Interface. The mechanism by which data are communicated are the electrical signals on the data lines. The actual signals that appear on the data lines depend on three things: the data currently being transferred, how this data is being represented, and the logic sense of the data lines.

Brief explanations of ASCII and internal data representation are given in Chapter 4. This section gives simple examples of how several representations are implemented during outputs and inputs through the GPIO Interface.

ASCII and Internal Representations

When data are moved through the GPIO Interface, the **data are generally sent one byte at a time, with the most significant byte first**. However, there are **three exceptions**; data are represented by words when READWORD and WRITEWORD are used, and when TRANSFER-WORD is used and when numeric data are moved with reads of IOSTATUS register 3 and writes to IOCONTROL register 3. The following diagrams illustrate which data lines are used during byte and word transfers.

GPIO Interface	Peripheral Device
DO15 — DO8 or DI15 — DI8	Upper 8 bits are not used (all 0's during byte transfers).
DO7 — DO0 or DI7 — DI0	Only lower 8 bits are used.

Diagram of Byte Transfers

GPIO Interface	Peripheral Device
	Upper 8 bits are used only when:
DO15 — DO8 or DI15 — DI8	1. Writing to IOCONTROL register 3 (reading from IOSTATUS register 3). 2. When READWORD, WRITEWORD, and TRANSFER_WORD are used.
DO7 — DO0 or DI7 — DI0	Lower 8 bits are used for ALL data transfers.

Diagram of Word Transfers

Example - Output Data Bytes

The following diagram shows the actual logic signals that appear on the least significant data byte (DO7 thru DO0) as the result of the corresponding output procedure; the most significant byte is always zeros with byte transfers. The actual logic levels depend on how the data lines are configured (i.e., as Low-true or High-true).

```
WRITESTRINGLN(12,'ASCII');
```

Signal Line DO7 ..... DO0	ASCII Char.
0 1 0 0 0 0 0 1	A
0 1 0 1 0 0 1 1	S
0 1 0 0 0 0 1 1	C
0 1 0 0 1 0 0 1	I
0 1 0 0 1 0 0 1	I
0 0 0 0 1 1 0 1	$_{CR}$
0 0 0 0 1 0 1 0	$_{LF}$

```
WRITECHAR(12,'B');
```

Signal Line DO7 ..... DO0	ASCII Char.
0 1 0 0 0 0 1 0	B

Example - Input Data Bytes

The following diagrams show the variable values that result from the logic signals being present during the corresponding input procedures on the least significant data byte (DI7 thru DI0); the most significant byte is always ignored with byte transfers. The actual logic levels required depend on how the data lines are configured (i.e., as Low-true or High-true).

```
READCHAR(12,c),
WRITELN('Value entered=',ORD(c));

  Value entered= 65
```

Signal Line DI7 ..... DI0	ASCII Char.
0 1 0 0 0 0 0 1	A

```
READSTRING(12,Str);
WRITELN ('String entered=',Str);

  String entered= ruok?
```

Signal Line DI7 ..... DI0	ASCII Char.
0 1 1 1 0 0 1 0	r
0 1 1 1 0 1 0 1	u
0 1 1 0 1 1 1 1	o
0 1 1 0 1 0 1 1	k
0 0 1 1 1 1 1 1	?
0 0 0 0 1 0 1 0	$_{LF}$

Example - Output Data Words

The following diagrams show the actual signals that appear on the Data Out lines as a result of the corresponding Pascal procedures and numeric values. All numeric values are first rounded to an INTEGER value before being placed on the Data Out lines. The actual logic level that appears on each line depends on how the lines have been configured (i.e., as High-true or Low-true).

```
Word:=3*256+3;
WRITEWORD(12,word);
```

	Signal Lines			
	DO15 ..... DO8		DO7 ..... DO0	
	0 0 0 0	0 0 1 1	0 0 0 0	0 0 1 1

```
Output_16_bits:=-1;
IOCONTROL(12,3,Output_16_bits);
```

	Signal Lines			
	DO15 ..... DO8		DO7 ..... DO0	
	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1

It is important to note that no output handshake is executed when the IOCONTROL procedure is executed; only the states of the Data Out lines and the I/O line are affected. Handshake sequence, if desired, must be performed by Pascal procedures in the program.

Example - Input Data Words

The following diagrams show the variable values that result from entering the logic signals on the Data In lines. Note that all sixteen-bit values entered are interpreted as INTEGER values.

	Signal Lines			
	DI15 ..... DI8		DI7 ..... DI0	
	0 0 0 0	0 0 0 1	1 1 1 1	1 1 1 1

```
READWORD(12,Input_16_bits);
WRITELN('INTEGER entered=',;Input_16_Bits);

INTEGER entered= 511
```

	Signal Lines			
	DI15 ..... DI8		DI7 ..... DI0	
	1 1 1 1	1 1 1 0	0 0 0 0	0 0 0 0

```
X:=IOSTATUS(12,3);
WRITELN('INTEGER entered=',X);

INTEGER entered= -512
```

It is important to note that no enter handshake is performed when the IOSTATUS function is executed. The only actions taken are the I/O line being placed in the High state and the Data In registers being read. If an input handshake is required, it must be performed by the Pascal program.

Using the Special-Purpose Lines

Four special-purpose signal lines are available for a variety of uses. Two of these lines are available for output (CTL0 and CTL1), and the other two are used as inputs (STI0 and STI1).

Driving the Control Output Lines

Setting bits 0 and 1 of GPIO IOCONTROL register 2 places a logic low on CTL0 and CTL1, respectively. The definition of this IOCONTROL register is shown in the following diagram.

Control Register 2 **Peripheral Control**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used						Set CTL1 (1 = Low; 0 = High)	Set CTL0 (1 = Low; 0 = High)
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

```
CH0:=0;
CH1:=1;
IOCONTROL(12,2,CH1*2+CH0);
```

As indicated in the diagram, setting a bit in the register places the corresponding line Low, while clearing the bit places a logic High on the line. The logic polarity of these signals cannot be changed. The signal remains on these lines until another value is written into the IOCONTROL register, and Reset has no effect on the state of either line.

Interrogating the Status Input Lines

The state of both status input lines STI0 and STI1 are determined by reading bits 0 and 1 of IOSTATUS register 5, respectively. A logic "1" in a bit position indicates that the corresponding line is at logic Low, and a "0" indicates the opposite logic state. This logic polarity cannot be changed. The definition of GPIO IOSTATUS register 5 is shown below.

Status Register 5 **Peripheral Status**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	PSTS Ok	EIR Line Low	STI1 Line Low	STI0 Line Low
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

```
P_status:=IOSTATUS(12,5);
Sti0:=BIT_SET(P_status,0);
Sti1:=BIT_SET(P_status,1);
```

Reading this register returns a numeric value that reflects the logic states of these lines **at the instant the computer reads the interface lines**; the state of these lines are not latched by any internal or external event.

GPIO Status and Control Registers

Status Register 0 Card identification = 3

Control Register 0 Reset interface if non-zero

Status Register 1 **Interrupt and DMA Status**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts Are Enabled	An Interrupt Is Currently Requested	Interrupt Level Switches (Hardware Priority)		Burst-Mode DMA	Word-Mode DMA	DMA Channel 1 Enabled	DMA Channel 0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 1 Set PCTL if non-zero

Status Register2 Not implemented

Control Register 2 **Peripheral Control**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used						Set CTL1 (1 = Low; 0 = High)	Set CTL0 (1 = Low; 0 = High)
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 3 Data In (16 bits)

Control Register 3 Data Out (16 bits)

Status Register 4 1 = Ready; 0 = Busy

Status Register 5 **Peripheral Status**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	PSTS Ok	EIR Line Low	STI1 Line Low	STI0 Line Low
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Summary of GPIO IOREAD_BYTE and IOWRITE_BYTE Registers

This section describes the GPIO Interface's IOREAD_BYTE and IOWRITE_BYTE registers. Keep in mind that these registers should be used **only** when you know the exact consequences of their use, as using some of the registers improperly may result in improper interface behavior. If the desired operation can be performed with IOSTATUS or IOCONTROL, you should not use IOREAD_BYTE or IOWRITE_BYTE.

GPIO IOREAD_BYTE Registers

Register 0—Interface Ready
Register 1—Card Identification
Register 2—Undefined
Register 3—Interrupt Status
Register 4—MSB of Data In
Register 5—LSB of Data In
Register 6—Undefined
Register 7—Peripheral Status

IOREAD_Byte Register 0 **Interface Ready**

A 1 indicates that the interface is Ready for subsequent data transfers, and 0 indicates Not Ready.

IOREAD_BYTE Register 1 **Card Identification**

This register always contains 3, the identification for GPIO interfaces.

IOREAD_BYTE Register 3 — Interrupt Status

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts Are Enabled	An Interrupt Is Currently Requested	Interrupt Level Switches (Hardware Priority)		Burst-Mode DMA	Word-Mode DMA	DMA Channel 1 Enabled	DMA Channel 0 Enabled
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

IOREAD_BYTE Register 4 — MSB of Data In

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI15	DI14	DI13	DI12	DI11	DI0	DI9	DI8
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

IOREAD_BYTE Register 5 — LSB of Data In

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

IOREAD_Byte Register 7 — Peripheral Status

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	PSTS Ok	EIR Line Low	STI1 Line Low	STI0 Line Low
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

GPIO IOWRITE_BYTE Registers

Register 0 — Set PCTL
Register 1 — Reset Interface
Register 2 — Interrupt Mask
Register 3 — Interrupt and DMA Enable
Register 4 — MSB of Data Out
Register 5 — LSB of Data Out
Register 6 — Undefined
Register 7 — Set Control Output Lines

IOWRITE_BYTE Register 0 — Set PCTL

Writing any non-zero numeric value to this register places PCTL in the Set state; writing zero causes no action.

IOWRITE_BYTE Register 1 — Reset Interface

Writing any non-zero numeric value to this register resets the interface.

IOWRITE_BYTE Register 2 — Interrupt Mask

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used						Enable Interface Ready Interrupts	Enable EIR Interrupts
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

IOWRITE_BYTE Register 3 — Interrupt and DMA Enable

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable Interrupts	Not Used			Enable Burst-Mode DMA	Enable Word-Mode DMA	Enable DMA Channel 1	Enable DMA Channel 0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

IOWRITE_BYTE Register 4 — MSB of Data Out

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DO15	DO14	DO13	DO12	DO11	DO10	DO9	DO8
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

IOWRITE_BYTE Register 5 — LSB of Data Out

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

IOWRITE_BYTE Register 7 — Set Control Output Lines

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used						Set CTL1 (1 = Low; 0 = High)	Set CTL0 (1 = Low; 0 = High)
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Notes

Chapter 15
RS-232 Serial Interface

Introduction

The HP 98626 Serial Interface is an RS-232C[1] compatible interface used for simple asynchronous ("async" for short) I/O applications such as driving line printers, terminals, or other peripherals. If your applications require more advanced capabilities, use the HP 98628 Datacomm Interface instead.

The Serial Interface uses a UART (Universal Asynchronous Receiver and Transmitter) integrated circuit to generate the required signals. Because the Serial Interface does not have a processor onboard, the computer must provide most control functions. Consequently, there is more interaction between the card and computer than when you use a more intelligent interface.

The RS-232C interface standard establishes electrical and mechanical interface requirements, but does not define the exact function of all the signals that are used by various manufacturers of data communications equipment and serial I/O devices. Consequently, when you plug your Serial Interface into an RS-232 connector, there is no guarantee the devices can communicate unless you have configured optional parameters to match the requirements of the other device.

The terms "asynchronous data communication" and "serial I/O" refer to a technique for transferring data between two devices one bit at a time where characters are not synchronized with preceding or subsequent characters. Each character is sent as a complete entity without relationship to other events. Characters may be sent in close succession, or they may be sent sporadically as data becomes available. Start and stop bits are used to identify the beginning and end of each character, with the character data placed between them.

1 RS-232C is a data communication standard established and published by the Electronic Industries Association (EIA). Copies of the standard are available from the association at 2001 Eye Street N. W., Washington D. C. 20006. Its equivalent for European applications is CCITT V.24.

Details of Serial I/O

The transfer of data over a serial line is a trivial operation when the host and terminal devices are designed to work together. However, some applications require some configuration before the communication can be performed smoothly. You must determine the operating parameters of the terminal device and then set up the host device for compatible operation.

The Serial Interface includes three default configuration switch clusters in addition to the select code and interrupt level switches. These three switch clusters include Modem Line, Baud Rate and Line Control switches. The operating parameters can be set using these switches or by program control which overrides most switches.

To determine operating parameters, you need to know the answer for each of the following questions about the peripheral device.

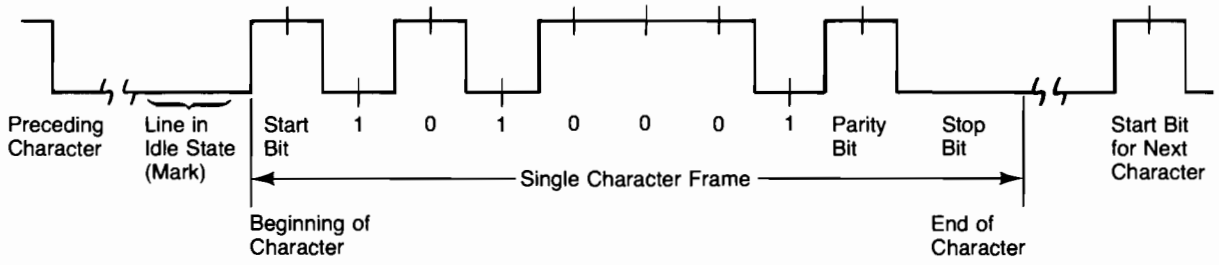
- What baud rate (line speed) is expected by the peripheral?
- Which of the following signal and control lines are actively used during communication with the peripheral?

—Data Set Ready (DSR)
—Clear to Send (CTS)
—Data Carrier Detect (DCD)
—Ring Indicator (RI)

In addition, you must know the expected format for an individual frame of character data. Each character frame consists of the following elements:

- **Start Bit**—The start bit signals the receiver that a new character is being sent. All other bits in a given frame are synchronized to the start bit.
- **Character Data Bits**—The next bits are the binary code of the character being transmitted, consisting of 5, 6, 7, or 8 bits; depending on the application.
- **Parity Bit**—The parity bit is optional, included only when parity is enabled.
- **Stop Bit(s)**—One or more stop bits identify the end of each character. The serial interface has no provision for inserting time gaps between characters.

Here is a simple diagram showing the structure of an asynchronous character and its relationship to other characters in the data stream:



Baud Rate

The rate at which data bits are transferred between the interface and the peripheral is called the baud rate. The interface card must be set to transmit and receive at the same rate as the peripheral, or data cannot be successfully transferred. The Baud Rate Select switches can be set to any one of the following values.

Baud Rate Switch Settings

Baud Rate	Switch Settings 3	2	1	0	Baud Rate	Switch Settings 3	2	1	0
50	0	0	0	0	1200	1	0	0	0
75	0	0	0	1	1800	1	0	0	1
110	0	0	1	0	2400 *	1	0	1	0
134.5	0	0	1	1	3600	1	0	1	1
150	0	1	0	0	4800	1	1	0	0
200	0	1	0	1	7200	1	1	0	1
300	0	1	1	0	9600	1	1	1	0
600	0	1	1	1	19200	1	1	1	1

* factory switch setting

Signal and Control Lines

A modem is used for serial communications between the computer and a remote device. The interface uses the following lines to indicate its status to the modem.

- Data-Terminal-Ready (DTR)—Indicates that the interface is ready for communications.
- Request-To-Send (RTS)—Indicates that the interface wants to send data.

The modem indicates its status to the interface through the following lines:

- Data-Set-Ready (DSR)—Indicates that the modem (data set) is ready.
- Clear-To-Send (CTS)—Indicates that the interface can transmit data over the communications link.
- Data-Carrier-Detect (DCD)—Indicates that the remote device has requested data.
- Ring-Indicator (RI)—Indicates that the modem is receiving an incoming call.

The Modem Line Disconnect switches are used to connect or disconnect the modem lines from the interface cable. When a given switch is in the CONNECT position, the corresponding modem line is connected from the peripheral device to the interface circuitry. When it is in the disconnected position, the modem line is disconnected, and the interface receiver input for that line is held HIGH (true). Any modem lines that are not actively used while communicating with the peripheral should be disconnected to minimize errors due to electrical noise in the cable. **Modem Line disconnect switch settings cannot be altered under program control.** To reconfigure the switches, the interface must be removed from the computer, and the settings changed by hand. These modem lines are monitored by the interface only if control register 13 is set. The default for control register 13 is 0. The modem line disconnect switches are not available with the HP Model 16's internal serial interface.

Software Handshake, Parity and Character Format

The Line Control switches are used to preset the software handshake, character format, and parity options. Functions are as follows:

Line Control Switch Settings

Software Handshake (Switches 6,7)	Parity (Switches 5,4,3)	Stop Bits (Switch 2)	Character Length (Switches 1,0)
00 ENQ/ACK *	000 no parity *	0 1 stop bit	00 5 bits/char
01 Xon/Xoff	001 ODD parity	1 2 stop bits *	01 6 bits/char
10 Reserved	011 EVEN parity	1 1.5 stop bits	10 7 bits/char
11 None	101 always ONE	if 5 bits/char	
11 8 bits/char *			
	111 always ZERO		

* factory switch settings

Software Handshake

Software handshakes are used by two communicating devices in order to prevent overflowing buffers. Special characters are used to implement the handshake. Two types of software handshakes are implemented.

- Enquire/Acknowledge—the host of this handshake sends an Enquire character after sending a specified number of characters (usually 80 characters), and then waits until it receives an Acknowledge character from the terminal. The terminal sends the Acknowlege character when it is ready to receive the specified number of characters.
- Xon/Xoff—the terminal sends an Xoff character when its receiving buffer is close to overflowing and then sends an Xon character when the buffer can again receive characters.

The Enquire/Acknowledge handshake implemented on the Serial Interface is the terminal-only version. The interface responds with an Acknowledge character (ASCII character 6) after it has received an Enquire character (ASCII character 5).

The Xon/Xoff handshake is the "host and terminal" version. The interface responds to an Xoff character by stopping all transmission. It resumes transmission when it receives a Xon character. It also sends a Xoff character (ASCII character 19) when it is running out of receiver buffer space, and sends an Xon character (ASCII character 17) after the buffer data has been processed.

Parity

The parity bit is used to detect errors as incoming characters are received. If the parity bit does not match the expected sense, the character is assumed to be incorrectly received. The action taken when an error is detected depends upon the interface and/or the application program.

Parity sense is determined by system requirements. The parity bit may be included or omitted from each character by enabling or disabling the parity function. When the parity bit is enabled, four options are available.

- ODD—Parity bit is set if there is an even number of bits set in the data character. The receiver performs parity checks on incoming characters.
- EVEN—Parity bit is set if there is an odd number of bits set in the data character. The receiver performs parity checks on incoming characters.
- ONE—Parity bit is set for all characters. Parity is checked by the receiver on all incoming characters.
- ZERO—Parity bit is cleared, but present for all characters. Parity is checked by the receiver on all characters.

Programming Techniques

Overview of Serial Interface Programming

Your computer uses several I/O Library facilities for data communication with various computers, terminals and peripheral devices. Serial Interface programs will include part or all of the following elements:

- Input procedures (including buffer-transfers)
- Output procedures (including buffer-transfers)
- IOSTATUS functions
- IOCONTROL procedures
- High level control procedures

The following steps represent a normal sequence of operations in a Serial I/O program.

1. Initialize the particular interface with an IORESET or initialize the whole I/O system by doing an IOINITIALIZE.
2. Set the operating parameters, this includes hardware characteristics, hardware handshake, and software handshake. This step can be skipped if the interface defaults are adequate.
3. Activate the Serial Interface by an IOCONTROL to Control Register 12. This activates the receiving buffer.
4. Do input and output using the I/O library procedures and functions. This is where all the data is transferred between the computer and the peripheral.
5. Deactivate the interface with an IOCONTROL to Control Register 12.
6. Cleanup the card by a IORESET or cleanup the whole I/O system by doing an IOUNINITIALIZE. This step disables the receiving buffer on the interface.

Initializing the Connection

Before you can successfully transfer information to a device, you must match the operating characteristics of the interface to the corresponding characteristics of the peripheral device. This includes matching signal lines and their functions as well as matching the character format for both devices. You can override some of the interface configuration switch settings by using the IOCONTROL procedure. This not only enables you to guarantee certain parameters, but also provides a means for changing selected parameters in the course of a running program. Control Register definitions for the Serial Interface are listed at the end of this chapter.

Interface Reset

Whenever an interface is connected to a modem that may still be connected to a telecommunications link from a previous session, it is good programming practice to reset the interface to force the modem to disconnect, unless the status of the link and remote connection are known. When the interface is connected to a line printer or similar peripheral, resetting the interface is usually unnecessary unless an error condition requires it.

The Serial Interface can be reset by an IORESET, IOINITIALIZE, IOUNITIALIZE or by use of an IOCONTROL to Control Register 0 with a non-zero value. The interface is restored to its power-up condition, except that the timeout value is not altered with the IORESET and IOCONTROL procedures.

Resetting the Serial Interface puts it in a non-active state. To activate the card use:

```
IOCONTROL( isc, 12, 1 )
```

But before the interface is activated, the operating parameters should be set.

Selecting the Baud Rate

In order to successfully transfer information between the interface card and a peripheral, the interface and peripheral must be set to the same baud rate. In addition to the procedure SET_BAUD_RATE, Control Register 3 will allow the user to change the baud rate. The following baud rates are recommended:

50	150	1200	4800
75	200	1800	7200
110	300	2400	9600
134	600	3600	19200

For example, to select a baud rate of 3600, either of these statements can be used:

```
IOCONTROL ( isc,3,3600 )
```

or

```
SET_BAUD_RATE ( isc, 3600 )
```

Use of values other than those shown may result in incorrect operation.

To verify the current baud rate setting, use the IOSTATUS function addressed to Status Register 3. All rates are in baud (bits/second).

Setting Character Format, Parity and Software Handshake

Control Register 4 overrides the Line Control switches that control software handshake, parity, and character format. To determine the value sent to the register, add the appropriate values selected from the following table:

Line Control IOCONTROL Register

Software Handshake (Bits 6,7)	Parity (Bits 5,4,3)	Stop Bits (Bit 2)	Character Length (Bits 1,0)
00 ENQ/ACK	xx0 no parity	0 1 stop bit	00 5 bits/char
00 Xon/Xoff	001 odd parity	1 2 stop bits	01 6 bits/char
01 Reserved	011 even parity	1 1.5 stop bits	10 7 bits/char
11 None	101 always One	if 5 bits/char	11 8 bits/char
	111 always Zero		

For example, use IOCONTROL to configure a character format of 8 bits per character, two stop bits, EVEN parity, and no software handshake:

```
IOCONTROL( isc, 4, BINARY('11011111'))
```

or

```
IOCONTROL( isc, 4, 223 )
```

To configure a 5-bit character length with 1 stop bit, no parity bit, and Enquire/Acknowledge software handshake use:

```
IOCONTROL( isc, 4, 0 )
```

The Serial_4 procedures SET_PARITY, SET_STOP_BITS, and SET_CHAR_LENGTH can be used to individually set these parameters. But to change the software handshake, you must do an IOCONTROL to register 4.

Modem Handshake

Two types of connections can be selected for the serial interface: direct connection and modem connection. The difference between the two types of connection is that with the modem connection, the modem lines DSR and DCD have to be high when a character is received and the lines DSR and CTS have to be high when a character is transmitted. To change modem checking, you must do an IOCONTROL to Control Register 13. For example:

```
IOCONTROL( isc, 13, 1 )  { turns on modem handshake }
IOCONTROL( isc, 13, 0 )  { direct connection }
```

Transferring Data

When the interface is properly configured, either by use of default switches or IOCONTROL statements, you are ready to begin data transfers.

Data Output

When a non-"buffer-transfer" output operation is done (example WRITECHAR), the interface waits until the previous character is sent and then puts the next character in the buffer. If your application requires that the character is sent before continuing with the program, bits 5 and 6 of Status Register 10 can be checked. The following procedure waits until all characters are transmitted:

```
procedure wait_sent( isc : type_isc );
{
 This procedure waits until the transmit buffer is empty.
 It works for the 98626 and 98628 cards.
 The modules IODECLARATIONS, GENERAL_O, and IOCOMASM needs
 to be imported.
}
var busy : boolean;
begin
 repeat
  if isc_table[isc].card_id = hp98626 then
    busy := binand( iostatus(isc,10),HEX('60')) <> HEX('60'))
  else  { assume the card is hp98628 }
    busy := iostatus(isc,38) = 0;
 until not busy;
end;
```

In the program the output sequence should be:

```
writechar( isc, 'a' );
wait_sent( isc );
```

Data Input

When a non-"buffer-transfer" input operation is done (example READSTRING), the interface waits for each character until the number of characters required is satisfied. For some applications, knowing if there is a character in the buffer is important. Bit 0 of Status Register 10 gives this information. The following function returns TRUE if there is at least one character in the receive buffer:

```
function have_char( isc : type_isc ) : boolean;
{
 This function returns true if there is a character in the
 receive buffer.  If not it returns false.
 It works for the 98626 and 98628 cards.
 The modules IODECLARATIONS, GENERAL_0, and IOCOMASM need
 to be imported.
}
begin
 if isc_table[isc].card_id = hp98626 then
  have_char := odd( iostatus( isc, 10 ))
 else { assume it is hp98628 card }
  have_char := odd( iostatus( isc, 5 ));
end;
```

The program input sequence would be:

```
if have_char( isc ) then readchar( isc, character );
```

Error Detection and Handling

The Serial Interface can detect and report several different classes of errors. The handling of errors by the interface differs depending on the severity of the error. For an unrecoverable error, an ESCAPE error is given. In case of an ESCAPE error, you can evaluate the error in the RECOVER section of your program. An I/O procedure ESCAPE error gives an ESCAPECODE of -26. To identify the error more closely, you can use the IOERROR_MESSAGE procedure with the IOE_RESULT variable as the parameter. For example:

```
if ESCAPECODE = -26  then
 begin
  writeln (IOERROR_MESSAGE(IOE_RESULT));
  ESCAPE(ESCAPECODE);
 end;
```

The TRY/RECOVER mechanism, the ESCAPECODE variable and the ESCAPE procedure are available by using $SYSPROG ON$. The IOERROR_MESSAGE procedure and the IOE_RESULT variable are available when you IMPORT the IODECLARATIONS module.

The errors which can happen are listed below.

- **Parity Error**—The parity bit on an incoming character does not match the parity expected by the receiver. This condition is most commonly caused by line noise. The interface handles this error by changing the character into a special character. This special character is defined by Control Register 19 and the default character is an underscore ("_"). The interface also sets bit 2 of Status Register 10.
- **Framing Error**—Start and stop bit(s) do not match the timing expectations of the receiver. This can occur when line noise causes the receiver to miss the start bit or obscures the stop bits. This error is handled similar to a parity error: the received character is translated into the special character defined by Register 19. The interface also sets bit 3 of Status Register 10.
- **Break received**—A BREAK was sent to the interface by the peripheral device. The Serial Interface does not interpret this condition as an error. The interface sets bit 4 of Status Register 10. Since BREAK is detected as a special type of framing error, bit 3 of Status Register 10 is also set. However, no special character is inserted into the receive buffer.
- **Overrun error**—Incoming data was not consumed fast enough so that one or more data characters were lost. This error can occur in two different ways: the software receive buffer overflowed, and the hardware receive buffer overflowed. In the first case, the program running cannot keep up with the receiver buffer at the current baud rate. Either reduce the baud rate, use software handshake, or change the program so that characters are read consistently. In the second case the error implies that interrupts were disabled so that the characters could not be processed. In both cases, an ESCAPE is generated and an IOE_RESULT of 314 results. In the second case, bit 1 of Status Register 10 is also set.
- **Timeout error**—Timeout errors occur when a character is not read or written within the timeout period specified. An ESCAPE is generated and an IOE_RESULT of 17 results. A timeout can occur when writing a character if DSR or CTS is low for the duration of the timeout. A timeout can occur when reading a character if no valid character was received during the timeout period.
- **CTS False Too Long**—This error occurs when a software handshake character cannot be sent because either DSR or CTS is low. The interface gives an ESCAPE error with an IOE_RESULT of 316.
- **Range Errors**—These errors occur when parameters passed to I/O library procedures and functions are out of range. For example, the Serial Interface does not support DMA; a call to TRANSFER with the transfer type being OVERLAP_DMA will result in an ESCAPE error with an IOE_RESULT of 7. These errors do not indicate a communications problem, rather they indicate a programming problem.

The ESCAPE errors "Overrun" and "CTS False Too Long" can happen even when there is no direct read or write to the interface. These errors will be saved by the interface and will be given at the next read or write operation to the interface. To avoid these ESCAPE errors, you can check Status Register 14. This register will return the IOE_RESULT of any pending errors. It will also clear the pending error so that the error can be handled without going into a RECOVER block.

As mentioned above, Status Register 10 has four bits which indicate if certain error conditions have occurred on the card. The four bits (1 through 4) are read-destructive bits. That is, if the register is read, the error bits are reset to zero.

When an ESCAPE error occurs (other than range type errors), it means there is a fairly serious problem. You should reset the interface if you decide to continue with the program. However an IORESET is sometimes undesirable since it resets all hardware parameters and modem connections are broken. To alleviate this problem, a soft reset is provided. A call to IOCONTROL with Register 14 and a non-zero value as parameters resets the interface without changing the hardware parameters or modem connections. It also clears the receive buffer.

Special Applications

This section provides advanced programming information for applications requiring special techniques.

Sending BREAK Messages

A BREAK is a special character transmission that usually indicates a change in operating conditions. Interpretation of break messages varies with the application. To send a break message, send a non-zero value to control Register 1.

```
IOCONTROL( isc,1,1 )     {Send a BREAK to peripheral}
```

Redefining Handshake and Special characters

Control registers 15 through 18 can be used to redefine the software handshake characters. The values passed to these registers should be the ordinal value of the character. The following example changes the Xon handshake character to DC2.

```
IOCONTROL( isc, 15, 20 )
```

Status registers 15 through 19 gives the ordinal value of the current handshake character. The following assigns to a character the current Acknowledge character.

```
ch := CHR(IOSTATUS(isc, 18))
```

As mentioned previously, Control Register 19 redefines the character into which parity error and framing error are converted. The following example sets this character to be the ASCII character DEL.

```
IOCONTROL( isc, 19, 127 )
```

Status Register 19 returns the current special character.

Using the Modem Line Control Registers

Modem line handshaking is performed automatically by the Serial Interface. The lines set by the interface are DTR and RTS. The lines checked by the interface are DSR, DCD, and CTS. Lines are set by the Serial Interface regardless of the modem handshake selection. Modem lines are checked only if the modem handshake is turned on. Your can change the values of the modem lines by writing to Control Register 5 or 7. The operations which involve modem lines are described below.

- Reset—both DTR and DSR are set to low.
- Activate—DTR is set to high.
- Deactivate—both DTR and DSR are set to low.
- Output—RTS is set to high. If the modem handshake is on, the interface will wait until DSR and CTS to become high before putting the characters in the transmit buffer.
- Input—If the modem handshake is on, all characters received when DSR or DCD is low are discarded (not put into the buffer).
- TRANSFER_END—When this procedure is called with direction "from_memory", at the end of the transfer RTS will be set low.

The following table summarizes the modem lines affected.

How Operations Affect Modem Lines

	DTR	RTS	DSR	CTS	DCD
reset	0	0	—	—	—
activate	1	—	—	—	—
deactivate	0	0	—	—	—
input	—	—	X	—	X
output	—	1	X	X	—
transfer_end	—	0	—	—	—

—	the modem line was not used.
0	the modem line was set to low.
1	the modem line was set to high.
X	the modem line was checked.

Control Register 5 controls various functions related to modem operation. Bits 0 thru 3 control modem lines, and bit 4 enables a self-test loopback configuration.

Modem Handshake Lines (RTS and DTR)

As explained earlier in this chapter, Request-To-Send and Data-Terminal-Ready lines are set or cleared by certain Serial Interface operations. For example, RTS is set high by the first write operation. Your application might require RTS to be high before the first write operation. The following example sets both RTS and DTR high at the same time.

```
IOCONTROL( isc, 5, 3 ); { set both RTS and DTR high }
IOCONTROL( isc,12, 1 ); { activate the receive buffer }
```

The above example also clears the loopback bit, and it clears the modem lines DRS and SRTS. To change only those two bits would require:

```
IOCONTROL(isc, 5, BINIOR(IOSTATUS(isc, 5), BINARY('00000011')))
{Sets RTS and DTR without disturbing other bits of register 5}
```

Programming the DRS and SRTS Modem Lines

Bits 3 and 2 of Control Register 5 control the present state of the Data Rate Select (DRS) and Secondary-Request-To-Send (SRTS) lines, respectively. When either bit is set, the corresponding modem line is activated. When the bit is cleared, so is the modem line.

Configuring the Interface for Self-test Operations

Self-test programs can be written for the Serial Interface. Prior to testing the interface, it must be properly configured. Using bit 4 of Control Register 5, you can rearrange the interconnections between input and output lines on the interface, enabling the interface to feed outbound data to the inbound circuitry.

When LOOPBACK is enabled (bit 4 is set), the UART output is set to its MARK state and sent to the Transmitted Data (TxD) line. The output of the transmitter shift register is then connected to the input of the receiver shift register, causing outbound data to be looped back to the receiver. In addition, the following modem control lines are connected to the indicated modem status lines.

Loopback Connections

Modem Control Line		to	Modem Status Line	
DTR	Data Terminal Ready		CTS	Clear-to-send
RTS	Request-to-send		DSR	Data Set Ready
DRS	Data Rate Select		DCD	Data Carrier Detect
SRTS	Secondary RTS		RI	Ring Indicator

When loopback is active, receiver and transmitter interrupts are fully operational. Modem control interrupts are then generated by the modem control outputs instead of the modem status inputs. Refer to Serial Interface hardware documentation for information about card hardware operation.

IOREAD_BYTE and IOWRITE_BYTE Register Operations

For those cases where you need to write special interface driver routines, the interface card hardware registers can be accessed by use of IOREAD_BYTE and IOWRITE_BYTE procedures. These capabilities are intended for use by experienced programmers who understand the inherent programming complexities that accompany this versatility. Warning: operations through hardware registers might interfere with the Serial Interface drivers.

Some registers are read/write; that is, both IOREAD_BYTE and IOWRITE_BYTE operations can be performed on a given register. Writing places a new value in the register; a read operation returns the current value. All registers have 8 bits available, and accept values from 0 thru 255 unless noted otherwise. When the value of a given bit is 1, the bit is set. Otherwise it is zero (cleared or inactive).

Some hardware registers are similar in structure and function to Status and Control Registers. However, their interaction with the Pascal operating system is considerably different. To prevent incorrect program operation, do not intermix the use of Status/Control registers and hardware registers in a given program.

Status and Control Registers

Most Control Registers accept values in the range from 0 thru 255. Some registers accept only specified values as indicated, or higher values for baud rate settings. Values less than zero are not accepted. Higher-order bits not needed by the interface are discarded if the specified value exceeds the valid range.

Reset value is the default value used by the interface after a reset or power-up until the value is overridden by a IOCONTROL procedure.

Status 0—Card Identification

Value returned: 2 (if 130 is returned, the Remote jumper wire has been removed from the interface card).

Control 0—Card Reset

Any value, 1 thru 255, resets the card. Immediate execution. Data transfers in process are aborted and any buffered data is destroyed.

Status 1—Interrupt Status

Bit 7 set: Interface hardware interrupt to CPU enabled.
Bit 6 set: Card is requesting interrupt service.
Bits 5&4: 00 Interrupt Level 3
01 Interrupt Level 4
10 Interrupt Level 5
11 Interrupt Level 6
Bits 3 thru 0 not used.

Control 1—Transmit BREAK

Any non-zero value sends a 400 millisecond BREAK on the serial line.

Status 2—Interface Activity Status

Bit 5 set: Software handshake character pending. The peripheral is the host and it should not be sending more characters since it is waiting for either an ENQUIRE character (ENQ/ACK handshake) or a Xon character (Xon/Xoff handshake).
Bit 4 set: Waiting for handshake character. The desktop is acting as a host and it is not transmitting because it has received an Xoff character and it is waiting for an Xon character.
Bit 1 set: Interrupts are enabled for this interface.
Bit 0 set: Transfer in progress. Either an input or an output transfer is in progress.
Bits 2, 3, 6, and 7 are not used.

Status 3—Current Baud Rate
Returns current baud rate.

Control 3 -- Set New Baud Rate
The recommended baud rates are:

50	150	1200	4800
75	200	1800	7200
110	300	2400	9600
134	600	3600	19200

Status 4—Current Character Format
See Control Register 4 for function of individual bits.

Control 4—Set New Character Format

Software Handshake (Bits 6,7)	Parity (Bits 5,4,3)	Stop Bits (Bit 2)	Character Length (Bits 1,0)
00 ENQ/ACK	xx0 no parity	0 1 stop bit	00 5 bits/char
01 Xon/Xoff	001 odd parity	1 2 stop bits	01 6 bits/char
10 Reserved	011 even parity	1 1.5 if	10 7 bits/char
11 None	101 always One	5 bits/char	11 8 bits/char
	111 always Zero		

Status 5—Current Status of Modem Control Lines
Returns CURRENT line state values. See Control Register 5 for function of each bit.

Control 5—Set Modem Control Line States
Bit 4 set: Enables loopback mode for diagnostic tests.
Bit 3 set: Set Secondary Request-to-Send line to active state.
Bit 2 set: Set Data Rate Select line to active state.
Bit 1 set: Set Request-To-Send line to active state.
Bit 0 set: Set Data-Terminal-Ready line to active state.

Status 6—Data In
Reads character from receive buffer. Results are undefined if no character is present in the receive buffer.

Control 6—Data Out
Sends character to transmitter holding register. This transmits a character without affect modem lines. Be sure that the transmit holding register is empty before this operation.

Status 7—Optional Receiver/Driver Status
Returns current value of optional circuit drivers or receivers as follows:
Bit 3: Optional Circuit Driver 3 (OCD3).
Bit 2: Optional Circuit Driver 4 (OCD4).
Bit 1: Optional Circuit Receiver 2 (OCR2).
Bit 0: Optional Circuit Receiver 3 (OCR3).
Other bits are not used (always 0).

Control 7—Set New Optional Driver States

Sets (bit = 1) or clears (bit = 0) optional circuit drivers as follows:

Bit 3: Optional Circuit Driver 3 (OCD3),
Bit 2: Optional Circuit Driver 2 (OCD2).
Other bits are not used.

Status 10—UART Status

Bit set indicates UART status or detected error as follows:

Bit 7: Not used.
Bit 6: Transmit Shift Register empty.
Bit 5: Transmit Holding Register empty.
Bit 4: Break received.
Bit 3: Framing error detected.
Bit 2: Parity error detected.
Bit 1: Receive Buffer Overrun error.
Bit 0: Receiver Buffer full.

Note: bits 1 through 4 are read destructive, they will be cleared each time this register is read with an IOSTATUS.

Status 11—Modem Status

Bit set indicates that the specified modem line or condition is active.

Bit 7: Data Carrier Detect (DCD) modem line active.
Bit 6: Ring Indicator (RI) modem line active.
Bit 5: Data Set Ready (DSR) modem line active.
Bit 4: Clear-to-Send (CTS) modem line active.
Bit 3: Change in DCD line state detected.
Bit 2: RI modem line changed from true to false.
Bit 1: Change in DSR line state detected.
Bit 0: Change in CTS line state detected.

Note: Bits 0 through 3 are read destructive; they will be cleared each time this register is read with an IOSTATUS.

Status 12—Interface activity

Returned value:

0—The interface is deactivated.
1—The interface is active.

Control 12—Set interface active

Value:

0—Deactivate the interface.
1—Activate the interface, sets DTR and does a soft reset.

Status 13—Modem handshake status

Returned value:

0—modem line handshaking is disabled.
1—modem line handshaking is enabled.

Control 13—Set modem handshake
Value
0—disable checking of modem lines.
1—enable checking of modem lines.

Status 14—Error pending
Returns the IOE_RESULT of any escape errors pending on the interface. A value of 0 is returned if no errors are pending.

Control 14—Soft reset
Any value, 1 through 255 resets the interface without affecting the modem lines or the hardware parameters. Receive buffer is reset with this command.

Status 15—Current Xon handshake character
Returns the ordinal value of the current Xon handshake character.

Control 15—Redefine Xon handshake character
Sets the Xon handshake character to have ordinal value equal to the input value. Default is DC1 (ASCII character 17).

Status 16—Current Xoff handshake character
Returns the ordinal value of the current Xoff handshake character.

Control 16—Redefine Xoff handshake character
Sets the Xoff handshake character to have ordinal value equal to the input value. Default is DC3 (ASCII character 19).

Status 17—Current Enquire handshake character
Returns the ordinal value of the current Enquire handshake character.

Control 17—Redefine Enquire handshake character
Sets the ENQUIRE handshake character to have ordinal value equal to the input value. Default is ENQ (ASCII character 5).

Status 18—Current Acknowledge handshake character
Returns the ordinal value of the current Acknowledge handshake character.

Control 18—Redefine Acknowledge handshake character
Sets the Acknowledge handshake character to have ordinal value equal to the input value. Default is ACK (ASCII character 6).

Status 19—Current framing/parity error character
Returns the ordinal value of the special character into which framing errors and parity errors would be converted.

Control 19—Redefine framing/parity error handshake character
Sets the special character used to represent framing errors and parity errors to have an ordinal value equal to the input value. Default is an underscore ("_") (ASCII character 95).

Serial Interface Hardware Registers

Interface Card Registers

IOREAD_BYTE and IOWRITE_BYTE registers 1, 3, 5, and 7 access interface registers. Their functions are as follows:

Register 1—Interface Reset and ID

IOREAD_BYTE to Register 1 returns the interface ID value — 2 for the HP 98626 Serial Interface. IOWRITE_BYTE to Register 1 with any value resets the interface as when using an IOCONTROL statement to Control Register 0.

Register 3—Interrupt Control

Only the upper four bits of Register 3 are used. Bits 5 and 4 return the setting of the Interrupt Level switches on the interface. Their values are as follows:

00	Interrupt Level 3	10	Interrupt Level 5
01	Interrupt Level 4	11	Interrupt Level 6

Bit 6 is set when an interrupt request is originated by the UART. No machine interrupt can occur unless bit 7, Interrupt Enable is set by an IOWRITE_BYTE statement. Only bit 7 is affected by IOWRITE_BYTE statements. During IOREAD_BYTE, bit 7 returns the current enable value; bits 6 thru 4 return interrupt request and level information.

Register 5—Optional Circuit and Baud Rate Control

IOWRITE_BYTE to bits 7 and 6 control the state of optional circuit drivers 3 and 4, respectively. IOREAD_BYTE returns current values of the respective drivers, plus the following:

Bit 5—Optional Circuit Receiver 2 state.
Bit 4—Optional Circuit Receiver 3 state.
Bits 3-0—Current Baud Rate switch setting (not necessarily the current UART baud rate). These switches can be interpreted in any way you choose. The current interpretation given to them by the serial interface drivers are as follows:

Setting	Baud Rate	Setting	Baud Rate
0000	50	1000	1200
0001	75	1001	1800
0010	110	1010	2400
0011	134.5	1011	3600
0100	150	1100	4800
0101	200	1101	7200
0110	300	1110	9600
0111	600	1111	19200

Note that IOWRITE_BYTE to this register can NOT be used to set the baud rate. Use Register 23, bit 7 and Registers 17 and 19 instead.

Register 7—Line Control Switch Monitor

IOREAD_BYTE to this register enables you to input the present settings of the Line Control switches that preset default character format and parity. Bit functions are included in the table earlier in this chapter under Using Interface Defaults to simplify programming. Bits 7 thru 0 correspond to switches 7 thru 0, respectively. IOWRITE_BYTE operations to this register are meaningless.

UART Registers

Addresses 17 through 29 access UART registers. They are used to directly control certain UART functions. The function of Registers 17 and 19 are determined by the state of bit 7 of Register 23.

Register 17—Receive Buffer/Transmitter Holding Register

When bit 7 of Register 23 is clear (0), this register accesses the single-character receiver buffer by use of IOREAD_BYTE. The IOWRITE_BYTE procedure places a character in the transmitter holding register.

The receiver and transmitter are doubly buffered. When the transmitter shift register becomes empty, a character is transferred from the holding register to the shift register. You can then place a new character in the holding register while the preceding character is being transmitted. Incoming characters are transferred to the receiver buffer when the receiver shift register becomes full. You can then input the character (IOREAD_BYTE) while the next character is being constructed in the shift register.

Registers 17 and 19—Baud Rate Divisor Latch

When bit 7 of Register 23 is set, Registers 17 and 19 access the 16-bit divisor latch used by the UART to set the baud rate. Register 17 forms the lower byte; Register 19 the upper. The baud rate is determined by the following relationship:

Baud Rate = 153 600/Baud Rate Divisor

To access the Baud Rate Divisor latch, set bit 7 of Register 23. This disables access to the normal functions of Registers 17 and 19, but preserves access to the other registers. When the proper value has been placed in the latch, be sure to clear bit 7 of Register 23 to return to normal operation.

Register 19—Interrupt Enable Register

When bit 7 of Register 23 is clear (0), this register enables the UART to interrupt when specified conditions occur. Only bits 0 thru 3 are used. IOWRITE_BYTE establishes a new value for each bit; IOREAD_BYTE returns the current register value. Interrupt enable conditions are as follows:

Bit 3—Enable Modem Status Change Interrupts. When set, enables an interrupt whenever a modem status line changes state as indicated by Register 29, bits 0 thru 3.

Bit 2—Enable Receiver Line Status Interrupts. When set, enables interrupts by errors, or received BREAKs as indicated by Register 27, bits 1 thru 4.

Bit 1—Enable Transmitter Holding Register Empty Interrupt. When set, allows interrupts when bit 5 of Register 27 is also set.

Bit 0—Enable Receiver Buffer Full Interrupts. When set, enables interrupts when bit 0 of Register 27 is also set.

Register 21—Interrupt Identification Register

This register identifies the cause of the highest-priority, currently-pending interrupt. Only bits 2, 1, and 0 are used. Bit 0, if set, indicates no interrupt pending. Otherwise an interrupt is pending as defined by bits 2 and 1. Causes of pending interrupts in order of priority are as follows:

11—Receiver Line Status interrupt (highest priority) is caused when bit 2 of Register 19 is set and a framing, parity, or overrun error, or a BREAK is detected by the receiver (indicated by bits 1 thru 4 of Register 27). The interrupt is cleared by reading Register 27.

10—Receive Buffer Register Full interrupt is generated when bit 0 of Register 19 is set and the Data Ready bit (bit 0) of Register 27 is active. To clear the interrupt, read the receiver buffer, or write a zero to bit 0 of Register 27.

01—Transmitter Holding Register Empty interrupt occurs when bit 1 of Register 19 is set and bit 5 of Register 27 is set. The interrupt is cleared by writing data into the transmitter holding register (Register 17 with bit 7 of Register 23 clear) with a IOWRITE_BYTE statement, or by reading this register (Interrupt Identification).

00—Modem Line Status Change interrupt occurs when bit 3 of Register 19 is set and a modem line change is indicated by one or more of bits 0 thru 3 of Register 29. To clear the interrupt, read Register 29 which clears the status change bits.

Register 23—Character Format Control Register

This register is functionally equivalent to Control and Status Register 4 except for bits 6 and 7. IOWRITE_BYTE sets a new character format; IOREAD_BYTE returns the current character format setting.

Bit 7—Divisor Latch Access Bit. When set, enables you to access the divisor latches of the Baud Rate generator during read/write operations to registers 17 and 19.

Bit 6—Set BREAK. When set, holds the serial line in a BREAK state (always zero), independent of other transmitter activity. This bit must be cleared to disable the break and resume normal activity.

Bits 5,4—Parity Sense. Determined by both bits 5 and 4. When bit 5 is set, parity is always ONE or ZERO. If bit 5 is not set, parity is ODD or EVEN as defined by bit 4. The combinations of bits 5 and 4 are as follows:

00	ODD parity	10	Always ONE
01	EVEN parity	11	Always ZERO

Bit 3—Parity Enable. When set, sends a parity bit with each outbound character, and checks all incoming characters for parity errors. Parity is defined by bits 4 and 5.

Bit 2—Stop Bit(s). Defined by a combination of bit 2 and bits 1 & 0.

Bit 2	Character Length	Stop Bits
0	5, 6, 7, or 8	1
1	5	1.5
1	6, 7, or 8	2

Bits 1,0—Character Length. Defined as follows:

Bits 1&0	Character Length
00	5 bits
01	6 bits
10	7 bits
11	8 bits

Register 25—Modem Control Register

This is a READ/WRITE register. IOREAD_BYTE returns current control register value. IOWRITE_BYTE sets a new value in the register. This register is equivalent to interface Control Register 5.

Bit 4—Loopback. When set, enables a loopback feature for diagnostic testing. Serial line is set to MARK state, UART receiver is disconnected, and transmitter output shift register is connected to receiver input shift register. Modem line outputs and inputs are connected as follows: DTR to CTS, RTS to DSR, DRS to DCD, and SRTS to RI. Interrupts are enabled, with interrupts caused by modem control outputs instead of inputs from modem.

Bit 3—Secondary Request-to-Send. Controls the OCD2 driver output. 1 = Active, 0 = Disabled.

Bit 2—Data Rate Select. Controls the OCD1 driver output. 1 = Active, 0 = Disabled.

Bit 1—Request-to-Send. Controls the RTS modem control line state. When bit 1 = 1, RTS is always active. When bit 1 = 0, RTS is toggled by the OUTPUT statement as described earlier in this chapter.

Bit 0—Data Terminal Ready. Holds the DTR modem control line active when the bit is set. If not set, DTR is controlled by the OUTPUT or ENTER statement as described earlier.

Bits 7, 6, and 5 are not used.

Register 27—Line Status Register

Bit 7—Not used.

Bit 6—Transmitter Shift Register Empty. Indicates no data present in transmitter shift register.

Bit 5—Transmitter Holding Register Empty. Indicates no data present in transmitter holding register. The bit is cleared whenever a new character is placed in the register.

Bit 4—Break Indicator. Indicates that the received data input remained in the spacing (line idle) state for longer than the transmission time of a full character frame. This bit is cleared when the line Status register is read.

Bit 3—Framing Error. Indicates that a character was received with improper framing; that is, the start and stop bits did not conform with expected timing boundaries.

Bit 2—Parity Error. Indicates that the received character did not have the expected parity sense. This bit is cleared when the register is read.

Bit 1—Overrun Error. Indicates that a character was destroyed because it was not read from the receiver buffer before the next character arrived. This bit is cleared by reading the line Status register.

Bit 0—Data Ready. Indicates that a character has been placed in the receiver buffer register. This bit is cleared by reading the receiver buffer register, or by writing a zero to this bit of the line Status register.

Register 29—Modem Status Register

Bit 7—Data Carrier Detect. When set, indicates DCD modem line is active.
Bit 6—Ring Indicator. If set, indicates that the RI modem line is active.
Bit 5—Data Set Ready. If set, indicates that the DSR modem line is active.
Bit 4—Clear-to-send. If set, indicates that CTS is active.
Bit 3—Change in Carrier Detect. When set, indicates that the DCD modem line has changed state since the last time the modem status register was read.
Bit 2—Trailing Edge of Ring Indicator. Set when the RI modem line changes from active to inactive state.
Bit 1—Delayed Data Set Ready. Set when the DSR line has changed state since the last time the modem status register was read.
Bit 0—Change in Clear-to-send. If set, indicates that the CTS modem line has changed state since the last time the register was read.

Cable Options and Signal Functions

The HP 98626A Serial Interface is available with RS-232C DTE and DCE cable configurations. The DTE cable option consists of a male RS-232C connector and cable designed to function as Data Terminal Equipment (DTE) when used with the serial interface. The cable and connector are wired so that signal paths are correctly routed when the cable is connected to a peripheral device wired as Data Communication Equipment (DCE), such as a modem. The cables are designed so that you can write programs that work for both DCE and DTE connections without requiring modifications to accommodate equipment changes.

The DCE cable option includes a female connector and cable wired so that the interface and cable behave like normal DCE. This means that signals are routed correctly when the female cable connector is connected to a male DTE connector.

Line printers and other peripheral devices that use RS-232C interfacing are frequently wired as DTE with a female RS-232C chassis connector. This means that if you use a male (DTE) cable option to connect to the female DTE device connector, no communication can take place because the signal paths are incompatible. To eliminate the problem, use an adapter cable to convert the female RS-232C chassis connector to a cable connector that is compatible with the male or female interface cable connector. The HP 13242 adapter cable is available in various configurations to fit most common applications. Consult cable documentation to determine which adapter cable to use.

The DTE Cable

The signals and functions supported by the DTE cable are shown in the signal identification table which follows. The table includes RS-232C signal identification codes, CCITT V.24 equivalents, the pin number on the interface card rear panel connector, the RS-232C connector pin number, the signal mnemonic used in this manual, whether the signal is an input or output signal, and its function.

RS-232C DTE (male) Cable Signal Identification Tables

Signal RS-232C	Signal V.24	Interface Pin #	RS-232C Pin #	Mnemonic	I/O	Function
AA	101	24	1	–	–	Safety Ground
BA	103	12	2		Out	Transmitted Data
BB	104	42	3		In	Received Data
CA	105	13	4	RTS	Out	Request to Send
CB	108	44	5	CTS	In	Clear to Send
CC	107	45	6	DSR	In	Data Set Ready
AB	102	48	7	–	–	Signal Ground
CF	109	46	8	DCD	In	Data Carrier Detect
SCF (OCR2)	122	47	12	SDCD	In	Secondary DCD
DB	114	41	15		In	DCE Transmit Timing
DD	115	43	17		In	DCE Receive Timing
SCA (OCD2)	120	15	19	SRTS	Out	Secondary RTS
CD	108.1	14	20	DTR	Out	Data Terminal Ready
CE (OCR1)	125	9	22	RI	In	Ring Indicator
CH (OCD1)	111	40	23	DRS	Out	Data Rate Select
DA	113	7	24		Out	Terminal Transmit Timing

Optional Circuit Driver/Receiver Functions

Not all signals from the interface card are included in the cable wiring. RS-232C provides for four optional circuit drivers and two receivers. Only two drivers and two receivers are supported by the DCE and DTE cable options. They are as follows:

Drivers		Receivers	
Name	Function	Name	Function
OCD1	Data Rate Select	OCR1	Ring Indicator
OCD2	Secondary Request-to-send	OCR2	Secondary Data Carrier Detect
OCD3	Not used		
OCD4	Not used		

If your application requires use of OCD3 or OCD4, you must provide your own interface cable to fit the situation.

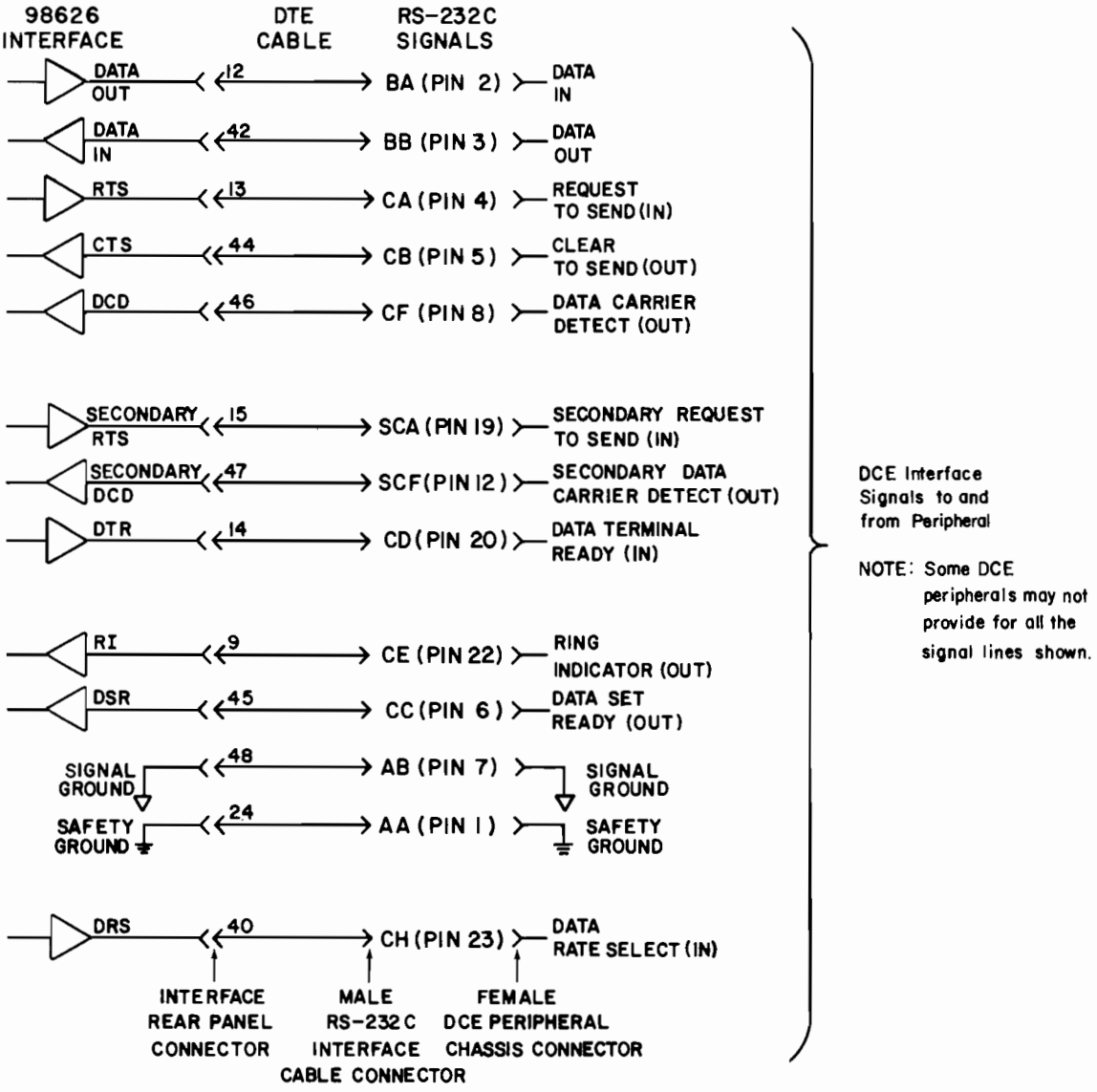
The DCE Cable

The DCE cable option is designed to adapt a DTE cable and serial or data communications interface to an identical interface on another desktop computer. It is also used with the serial interface to simulate DCE operation when driving a peripheral wired for DTE operation. The DCE cable is equipped with a female connector. Since most DTE peripherals are also equipped with female connectors (pin numbering is the same as the standard male DTE connector), an adapter (such as the HP 13242M) is used to connect the two female connectors as explained earlier.

Note

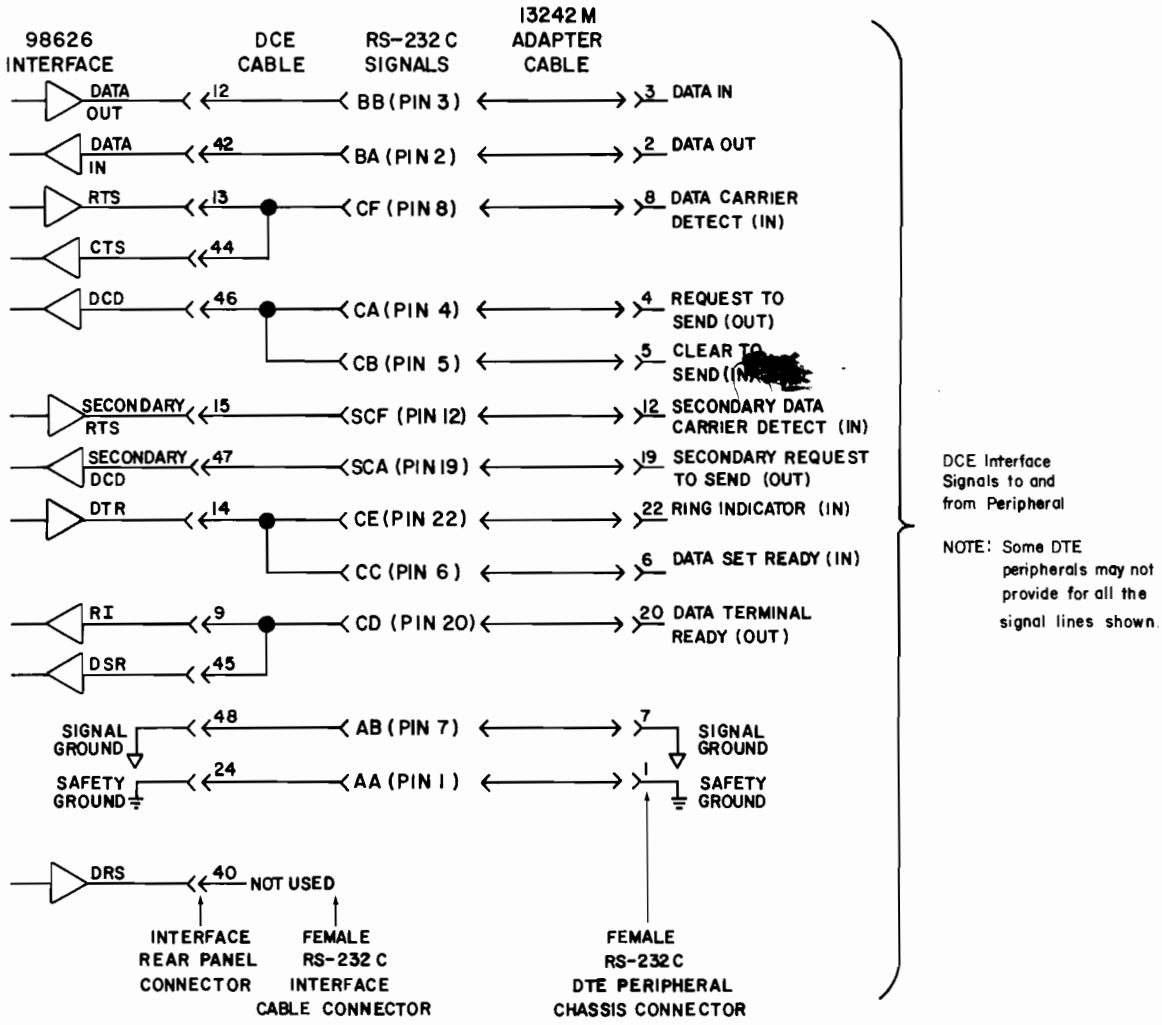
Not all RS-232C devices are wired the same. To ensure proper operation, you must know whether the peripheral device is wired as DTE or DCE. The interface cable option and associated adapter cable, if needed, must be configured to properly mate with the female DTE chassis connector.

The following schematic diagram shows the input and output signals for the Serial Interface and how they are connected to a DCE peripheral.



DTE Cable Diagram

This diagram shows an HP 13242M adapter cable connected to a DCE interface cable and a DTE peripheral. Note that RTS is connected to CTS in the DCE cable. If your peripheral uses RTS/CTS handshaking, a different adapter cable must be used with the appropriate DTE or DCE interface cable option.



DCE Cable Diagram

I/O Procedures

HPIB Status/Control

ABORT_HPIB	Ceases all HP-IB activity and attempts to place the HP-IB in a known state.
ACTIVE_CONTROLLER	TRUE if the specified interface is currently active controller.
CLEAR	Attempts to send a form of the clear message to the specified device(s).
CLEAR_HPIB	Clears the specified HP-IB line.
END_SET	Indicates whether or not EOI was set on the last byte read.
HPIB_LINE	Returns the current state of the specified line. Not all lines are accessable at all times.
LISTEN	Sends the specified listen address on the bus.
LISTENER	TRUE if the specified interface is currently addressed as a listener.
LOCAL	Places the device(s) in local mode.
LOCAL_LOCKOUT	Sends LLO (the local lockout message) on the bus.
LOCKED_OUT	TRUE if the specified interface is currently in the local lockout state.
MY_ADDRESS	Returns the HP-IB address of the specified HP-IB interface.
PASS_CONTROL	Passes control from the specified interface to another device on the bus.
PPOLL	Sets the ATN and EOI bus lines on the specified interface and then reads the data bus lines.
PPOLL_CONFIGURE	Programs the logical sense and data bus line on which the selected device responds to a parallel poll.
PPOLL_UNCONFIGURE	Causes the specified device(s) to disable the parallel poll response.
REMOTE	Sends the messages to place the bus device(s) into the remote state.
REMOTED	Indicates if the REM line is being asserted.
REQUESTED	TRUE if any device is currently asserting the SRQ line.
REQUEST_SERVICE	Sets up the SPOLL response byte in the specified interface.
SECONDARY	Sends a secondary command byte over the bus.
SEND_COMMAND	Sends a single byte over the HP-IB interface with ATN true.
SET_HPIB	Sets the specified HP-IB control line.
SPOLL	Performs a serial poll to the selected device.
SYSTEM_CONTROLLER	TRUE if the specified interface is the system controller.
TALK	Sends a talk address over the bus.
TALKER	TRUE if the specified interface is currently addressed as a talker.
TRIGGER	Sends a trigger command to the specified device(s).
UNLISTEN	Sends an unlisten command on the bus.
UNTALK	Sends an untalk command on the bus.

Serial Control

ABORT_SERIAL	Attempts to return a serial interface to a known state.
CLEAR_SERIAL	Clears the specified line on a serial interface card.
SEND_BREAK	Sends a break to the selected serial interface.
SERIAL_LINE	TRUE if the specified line on the serial interface is asserted.
SET_BAUD_RATE	Sets the serial interface to the specified baud rate.
SET_CHAR_LENGTH	Specifies the character length, in bits, for serial communications.
SET_PARITY	Determines what parity mode the serial interface will use.
SET_SERIAL	Sets the specified modem line on the connector.
SET_STOP_BITS	Sets the number of stop bits on the serial interface.

General Status/Control

IOCONTROL	Sends control information to the selected interface.
IOERROR_MESSAGE	Returns a string containing an English textual description of an error produced by the I/O procedure library.
IOINITIALIZE	Initializes all interfaces.
IOREAD_BYTE	Reads the byte contained in specified register (physical address) on the selected interface.
IOREAD_WORD	Reads the word contained in the specified register (physical address) on the selected interface.
IORESET	Resets the specified interface to its intial (power on) state.
IOSTATUS	Returns the contents of an interface status register.
IOUNINITIALIZE	Uninitializes all interfaces.
IOWRITE_BYTE	Writes the supplied value (representing one byte) to the specified register (physical address) on the selected interface.
IOWRITE_WORD	Writes the supplied value (representing 16 bits) to the specified register on the selected interface.
SET_TIMEOUT	Sets up a timeout for all read and write operations except transfer.

General Input

READCHAR	Reads a single byte from the specified interface.
READWORD	Reads 2 bytes from byte oriented interfaces or a single 16 bit quantity from word-oriented interfaces.
READNUMBER	Performs a free field numeric entry from the specified device.
READNUMBERLN	Reads in a free field number and then searches for a line feed.
READSTRING	Reads characters into the specified string.
READSTRING_UNTIL	Reads characters from the selected device into the specified string until the prescribed terminator is encountered.
READUNTIL	Reads characters until the match character occurs.
SKIPFOR	Reads the specified number of characters from the selected device.

General Output

WRITECHAR	Sends a single byte as data over the interface path.
WRITENUMBER	Outputs a free field number to the specified device.
WRITENUMBERLN	Outputs the number, a carriage return and a linefeed.
WRITESTRING	Sends the specified string to the specified device.
WRITESTRINGLN	Outputs the string, a carriage return and a line feed.
WRITEWORD	Writes 2 bytes to a byte-oriented interface or a 16-bit quantity to a word-oriented interface.

Buffer Control

BUFFER_DATA	Returns the number of characters available in the buffer.
BUFFER_RESET	Sets the empty and fill pointers to the empty state.
BUFFER_SPACE	Returns the available space left in the buffer.
BUFFER	Create a buffer area of the specified number of bytes.

Buffer I/O

READBUFFER	Reads a single byte from the buffer space and updates the empty pointer in the buf_info record.
READBUFFER_STRING	Reads the specified number of characters from the buffer and puts them into the string variable.
WRITEBUFFER	Writes a single byte into the buffer space and update the fill pointer in the buf_info record.
WRITEBUFFER_STRING	Takes the specified string and places it in the buffer and updates the fill pointer.

Transfer Control

ABORT_TRANSFER	Stop any transfer that is currently active in the buffer.
BUFFER_ACTIVE	Returns a TRUE if there is a transfer occurring on the buffer.
ISC_ACTIVE	Returns a TRUE if there is a transfer occurring on the interface.
TRANSFER	Transfers the specified number of bytes to or from the buffer space using the specified transfer type.
TRANSFER_END	Transfers data to or from the buffer.
TRANSFER_UNTIL	Transfers bytes into the buffer until the buffer is full or the termination character was received.
TRANSFER_WORD	Transfers the specified number of words into the buffer.

Binary Logic Operations

BINAND	Returns the bit-by-bit logical AND of its arguments.
BINCMP	Returns the bit-by-bit logical complement of the argument.
BINEOR	Returns the bit-by-bit logical exclusive-OR of the argument.
BINIOR	Returns the bit-by-bit logical inclusive-OR of its arguments.
BIT_SET	TRUE if the specified bit position of the argument is equal to 1.

Procedure Library Summary

Graphics Procedures

Graphics Control

CLEAR_DISPLAY	Clears the graphics display.
DISPLAY_INIT	Enables a device as the logical graphics display.
DISPLAY_TERM	Disables the enabled graphics display device.
GRAPHICSERROR	Returns a graphics error number.
GRAPHICS_INIT	Initializes the graphics system.
GRAPHICS_TERM	Terminates the graphics system.
INPUT_ESC	Invokes a device dependent escape function to inquire from the graphics display device.
INQ_WS	Returns information about the graphics system.
OUTPUT_ESC	Performs a device dependent escape function on the graphics display device.

Graphics Output Primitives

GTEXT	Outputs graphical text to the graphics display.
INT_LINE	Draws a line from the starting position to the world coordinate specified.
INT_MOVE	Sets the starting position to the world coordinate position specified.
LINE	Draws a line from the starting position to the world coordinate specified.
MOVE	Sets the starting position to the world coordinate specified.

Primitive Attributes

SET_COLOR	Sets the color attribute for output primitives.
SET_CHAR_SIZE	Sets the character size attribute for graphical text.
SET_LINE_STYLE	Sets the line style attribute for lines and text.
SET_TEXT_ROT	Specifies the text direction.

Viewing Transformation

SET_ASPECT	Redefines the aspect ratio of the virtual coordinate system.
SET_DISPLAY_LIM	Redefines the logical display limits of the graphics display.
SET_VIEWPORT	Sets the boundaries of the viewport in the virtual coordinate system.
SET_WINDOW	Defines the boundaries of the window.

Graphics Input

AWAIT_LOCATOR	Waits until activation of the locator button and then reads from the enabled locator device.
LOCATOR_INIT	Enables the locator device for input.
LOCATOR_TERM	Disables the enabled locator device.
SAMPLE_LOCATOR	Samples the locator device.
SET_ECHO_POS	Defines the locator echo position on the graphics display.
SET_LOCATOR_LIM	Redefines the logical locator limits of the graphics locator.

LIF Procedures

LIFASCIIGET	Sequentially reads ASCII or BINARY file records.
LIFASCIIPUT	Sequentially writes ASCII or BINARY file records.
LIFCLOSE	Performs the final operations on a file, and removes the file block from the heap if it was created by LIFOPEN.
LIFCREATE	Creates a directory entry for the file described.
LIFDISPOSEFIB	De-allocates a file information block from the heap and clears the file designator pointer.
LIFEOF	TRUE if the designated file is at end-of-file, or if the file is closed.
LIFGET	Reads data from files of types other than ASCII and BINARY.
LIFGETFLD	Reads some of the attribute fields of a LIF file.
LIFNEWFIB	Allocates a file information block from the heap, initializes it, and sets the file designator pointer.
LIFOPEN	Initializes a file information block for use by file access functions.
LIFPURGE	Removes the LIF directory entry for the named file.
LIFPUT	Writes data to files of types other than ASCII and BINARY.
LIFSETFLD	Reads some of the attribute fields of a LIF file.

Procedure Library Language Reference

Introduction

The Pascal Programming Language was designed as a teaching language, and as such was intended to be machine independent. This has good and bad points. Being machine independent makes the language more easily tranportable, but also ensures that it is difficult, if not impossible, to access any innovative hardware features provided by a specific computer system.

To allow easy access to the graphics and I/O features of your Pascal system, a set of procedures and functions are provided in SYSTEM.LIBRARY. This language reference describes the syntax and semantics for the procedures and functions provided to access I/O and graphics, along with the LIF procedure library, for reading and writing LIF files, for interchange between HP computers.

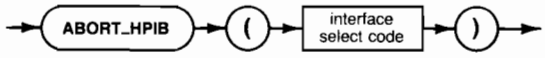
The small block of text labeled IMPORT, immediately below the title of each entry, lists the module which must be declared in an IMPORT statement in order to access the feature.

ABORT_HPIB

IMPORT: hpib_2
iodeclarations

This **procedure** ceases all HP-IB activity and attempts to place the HP-IB in a known state. If the controlling interface is System Controller, but not Active Controller, it is made Active Controller.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

Semantics

The actual action taken depends upon whether the computer is currently active or system controller. The various actions taken are listed in the table below:

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	IFC (duration ≥100μsec) REN $\overline{\text{ATN}}$	Error	ATN MTA UNL $\overline{\text{ATN}}$	Error
Not Active Controller	IFC (duration ≥100 μsec)* REN $\overline{\text{ATN}}$		No Action	

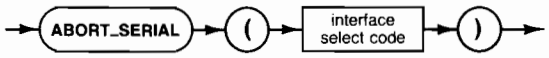
* The IFC message allows a non-active controller (which is the system controller) to become the active controller.

ABORT_SERIAL

IMPORT: serial_3
iodeclarations

This **procedure** attempts to return a serial interface to a known state. Any current active transfers are halted.

Syntax



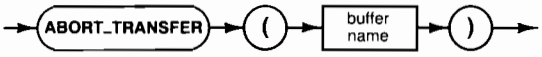
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

ABORT_TRANSFER

IMPORT: general_4
iodeclarations

This **procedure** will stop any transfer that is currently active in the buffer.

Syntax



Item	Description/Default	Range Restrictions
buffer name	Variable of TYPE *buf_info_type*.	See Chapter 11

Semantics

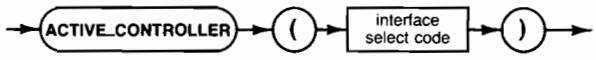
The termination of the transfer is accomplished by reseting the interface currently associated with the specified buffer name. **This returns the interface to power on default configuration, and all configuring information is lost.**

ACTIVE_CONTROLLER

IMPORT: hpib_1
iodeclarations

This BOOLEAN **function** returns TRUE if the specified interface is currently active controller.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

ADDR_TO_LISTEN

IMPORT: hpib_1
iodeclarations

Note

This function is provided for use by the internal I/O Procedure Library drivers, only. Unexpected and possible undesirable results may occur if it is used.

The following sequence of statements will address a device to listen:

```
.
.
TALK (7,24);
UNLISTEN (7);
LISTEN( 7, MY_ADDRESS(7));
.
.
```

ADDR_TO_TALK

IMPORT: hpib_1
iodeclarations

Note

This function is provided for use by the internal I/O Procedure Library drivers, only. Unexpected and possible undesirable results may occur if it is used.

The following sequence of statements will address a device to talk:

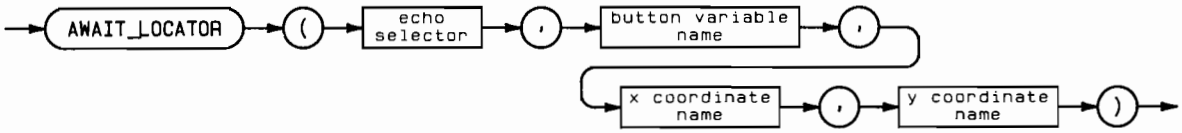
```
.
.
UNLISTEN (7);
LISTEN (7,24);
TALK (7, MY_ADDRESS(7));
.
.
```

AWAIT_LOCATOR

IMPORT: dgl_lib

This **procedure** waits until activation of the locator button and then reads from the enabled locator device. Various echo methods can be selected.

Syntax



Item	Description/Default	Range Restrictions
echo selector	Expression of TYPE INTEGER	MININT to MAXINT
button variable name	Variable of TYPE INTEGER	–
x coordinate name	Variable of TYPE REAL	–
y coordinate name	Variable of TYPE REAL	–

Procedure Heading

```
PROCEDURE AWAIT_LOCATOR (     Echo   : INTEGER;
                          VAR Button : INTEGER;
                          VAR WX, WY : REAL    );
```

Semantics

AWAIT_LOCATOR waits until the locator button is activated and then returns the value of the selected button and the world coordinates of the locator. While the button press is awaited, the locator position can be tracked on the graphic display device. If an invalid button is pressed, the button value will be returned as 0; otherwise it will contain the value of the button pressed. On locators that use a keyboard for the button device (e.g. HP 9826 / HP 9836), the ordinal value of the key pressed is returned.

The **echo selector** selects the type of echo used. Possible values are:

0 - No echo.
1 - Echo on the locator device.
2 - Small cursor
3 - Full cross hair cursor
4 - Rubber band line
5 - Horizontal rubber band line
6 - Vertical rubber band line
7 - Snap horizontal / vertical rubber band line
8 - Rubber band box
9 and above - Device dependent echo on the locator device.

Locator input can be echoed on either a graphics display device or a locator device. The meaning of the various echoes on various devices used as locators and displays is discussed below.

The **button value** is the INTEGER value of the button used to terminate the locator input.

The **x** and **y position** represent the world coordinate point returned from the enabled locator.

AWAIT_LOCATOR implicitly makes the picture current before sending any commands to the locator device. The locator should be enabled (LOCATOR_INIT) before calling AWAIT_LOCATOR. The locator is terminated by the procedure LOCATOR_TERM.

Range and Limit Considerations
If the echo selector is out of range, the call to AWAIT_LOCATOR is completed using an echo selector of 1 and no error is reported. Echoes 2 through 8 require a graphics display to be enabled. If a display is not enabled, the call will be completed with echo 1 and GRAPHICSERROR will return 4.

If the point entered is outside of the current logical locator limits, the transformed point will still be returned in world coordinates.

Starting Position Effects
The location of the starting position is device dependent after this procedure with echo 0 or echo 1. For soft-copy devices it is typically unchanged; however, for plotters the pen position (starting position) will remain at the last position it was moved to by the operator. This is done to reduce pen movement back to the current position after each AWAIT_LOCATOR invocation.

Echo Types
Several different types of echoing can be performed. Some echoes are performed on the locator device while others use the graphics display device. When the echo selector is in the range 2 thru 8, the graphics display device will be used in echoing. All of the echoes on the graphics display start at a point on the graphics display called the locator echo position (see SET_ECHO_POS). For some of these echoes the locator echo position is also used as a fixed reference point. For example, the fixed end of the rubber band line will be at the locator echo position. The echoes available are:

2. Small cursor
   Track the position of the locator on the graphics display device. The initial position of the cursor is at the locator echo position. The point returned is the locator position.
3. Full cross hair cursor
   Designate the position of the locator on the graphics display device with two intersecting lines. One line is horizontal with a length equal to the width of the logical display surface. The other line is vertical with a length equal to the height of the logical display surface. The initial point of intersection is at the current locator echo position. The point returned is the locator position.
4. Rubber band line
   Designate the endpoints of a line. One end is fixed at the locator echo position; the other is designated by the current locator position. The locator position can be told from the locator echo position by the presence of a small cursor (echo 2) at end representing the locator echo position. The point returned is the locator position.

5. Horizontal rubber band line
   Designate a horizontal line. One endpoint of the line is fixed at the locator echo position; the other endpoint has the world Y-coordinate of the locator echo position and the world X-coordinate of the current locator position. The locator position can be told from the locator echo position by the presence of a small cursor (echo 2) at end representing the locator echo position. The point returned will have the X-coordinate of the locator position and the Y-coordinate of the locator echo position.
6. Vertical rubber band line
   Designate a vertical line. One endpoint of the line is fixed at the locator echo position; the other endpoint will have the world X-coordinate of the locator echo position and the world Y-coordinate of the current locator position. The locator position can be told from the locator echo position by the presence of a small cursor (echo 2) at end representing the locator echo position. The point returned will have the X-coordinate of the locator echo position and the Y-coordinate of the locator position.
7. Snap horizontal / vertical rubber band line
   Designate a horizontal / vertical line. One endpoint of the line is fixed at the locator echo position. The other endpoint will be either a horizontal (see echo 5) or vertical (see echo 6) rubber band line, depending on which one produces the longer line. If both lines are of equal length, a horizontal line will be used. The locator position can be told from the locator echo position by the presence of a small cursor (echo 2) at end representing the locator echo position. The point returned is the endpoint of the echoed line.
8. Rubber band box
   Designate a rectangle. The diagonal of the rectangle is the line from the locator echo position to the current locator position. The locator position can be told from the locator echo position by the presence of a small cursor (echo 2) at end representing the locator echo position. The point returned will be the locator position.

Echo selectors of 1 and greater than or equal to 9 produce a device dependent echo on the locator device. Most locator devices support at least one form of echoing. Possible ones include beeping, displaying the value entered, or blinking a light each time a point is entered. If the specified echo is not supported on the enabled locator device, echo 1 will be used.

Echoes on Raster Displays
Raster displays support all the echoes described under "Echo Types."

Echoes on HPGL Plotters
Hard copy plotting devices (such as the 9872 or the 7580) cannot perform all the echoes listed above. The closest approximation possible is used for simulating them. The actual echo performed may also depend on whether the plotter is also being used as the locator. The echoes available on plotters are:

2. Small cursor

   Initially the plotter's pen will be moved to the locator echo position. The pen will then reflect the current locator position (i.e., track) until the locator operation is terminated.
3. Full cross hair cursor
   Simulated by ECHO #2.
4. Rubber band line

   Simulated by ECHO #2.

5. Horizontal rubber band line
   If the plotter is **not** the current locator device, the plotter's pen will initially be moved to the current locator echo position. The pen will then reflect the X coordinate of the current locator position and the Y coordinate of the current locator echo position.

   If the plotter is used as the locator, this echo is simulated by echo 2 except the current locator X coordinate and the locator echo position Y coordinate are returned.
6. Vertical rubber band line
   If the plotter is **not** the current locator device, the plotter's pen position will initially be moved to the current locator echo position. The pen will then reflect the X coordinate of the current locator echo position and the Y coordinate of the current locator position.

   If the plotter is used as the locator, this echo is simulated by echo 2 except the locator echo position X coordinate and the current locator Y coordinate are returned.
7. Snap horizontal / vertical rubber band line
   Designate a horizontal / vertical line. One endpoint of the line is fixed at the locator echo position. The other endpoint will be either a horizontal (see echo 5) or vertical (see echo 6) rubber band line, depending on which one produces the longer line. If both lines are of equal length, a horizontal line will be used. The locator position can be told from the locator echo position by the presence of a small cursor (echo 2) at end representing the locator echo position. The point returned is the endpoint of the echoed line.
8. Rubber band box
   Simulated by echo 2. The point returned will be the locator position.

Tablet Locators
For HPGL graphics tablets the operator positions the stylus to the desired position and depresses it. The button value returned is always one. For an echo selector of 1 the tablet beeper is sounded when the stylus is depressed. An echo selector greater than or equal to 9 uses the same echo as an echo selector of 1.

The Knob as Locator
When the knob is specified as the locator (LOCATOR_INIT with device selector of 2) the keyboard keys have the following meanings:

Arrow keys	Move the cursor in the direction indicated.
Knob	Move the cursor right and left.
Knob with shift key pressed	Move the cursor up and down.
Number keys $1 \rightarrow 9$	Change the amount the cursor is moved per arrow keypress or knob rotation. 1 provides the least movement and 9 provides the most.

All other keys act as the locator buttons. The ordinal value of the locator button (key) struck is returned in BUTTON.

For an echo selector of 1 the position of the locator is indicated by a small crosshair cursor on the graphics display.

The initial position of the cursor is located at the current starting position of the graphics display. This is the point obtained by the last invocation of await_locator, or the lower left hand corner of the locator limits if no point has been received since LOCATOR_INIT was executed. For back to back AWAIT_LOCATOR calls this would mean the second AWAIT_LOCATOR would begin were the first AWAIT_LOCATOR left the cursor. Echo selectors greater than or equal to 9 have the same effect as an echo selector of 1.

Locator input can be echoed on either a graphics display device or a locator device. Echoes 2 thru 8 are explained above under "Echoes on Raster Displays" and "Echoes on HPGL Plotters". For an echo selector of 0 or 1 the pen tracks the locator position. Echo selectors greater than or equal to 9 have the same effect as an echo selector of 1.

HPGL Plotters as Locators

The AWAIT_LOCATOR function enables a digitizing mode in the device. For HPGL plotters the operator then positions the pen to the desired position with the cursor buttons or joy stick and then presses the enter key. The pen state (0 for 'up', and 1 for 'down') is returned in the button parameter.

Following locator input (echo on the locator device), the pen position will remain at the last position it was moved to by the operator. This means that the starting position for the next graphics primitive will be wherever the pen was left.

Locator input can be echoed on either a graphics display device or a locator device. Echoes 2 thru 8 are explained above under "Echoes on Raster Displays" and "Echoes on HPGL Plotters". For an echo selector of 0 or 1 the pen tracks the locator position. Echo selectors greater than or equal to 9 have the same effect as an echo selector of 1.

Error Conditions

The graphics system must be initialized and the locator device must be enabled or the call will be ignored. If the echo selector is between 1 and 9 and the graphics display is not enabled, the call will be completed with an echo selector of 1. If any of the preceding errors are encountered, an ESCAPE (-27) is generated, and GRAPHICSERROR will return a non-zero value.

BINAND

IMPORT: iocomasm

This INTEGER **function** returns the bit-by-bit logical-and of its arguments.

Syntax

Item	Description/Default	Range Restrictions
argument	Expression of TYPE INTEGER.	MININT thru MAXINT

Semantics

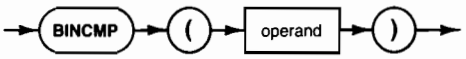
The arguments for this function are represented as 32-bit two's complement integers. Each bit in an argument is logically anded with the corresponding bit in the other argument. The results of all the ands are used to construct the integer which is returned.

BINCMP

IMPORT: iocomasm

This INTEGER **function** returns the bit-by-bit logical complement of the argument.

Syntax



Item	Description/Default	Range Restrictions
argument	Expression of TYPE INTEGER.	MININT thru MAXINT

Semantics

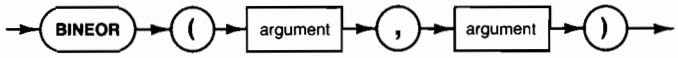
The argument for this function is represented as a 32-bit two's complement integer. Each bit in the argument is logically complemented, and the resulting integer is returned.

BINEOR

IMPORT: iocomasm

This INTEGER **function** returns the bit-by-bit logical exclusive-or of the two arguments.

Syntax



Item	Description/Default	Range Restrictions
argument	Expression of TYPE INTEGER.	MININT thru MAXINT

Semantics

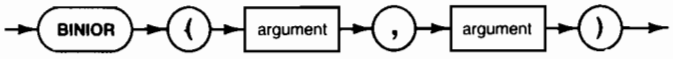
The arguments for this function are represented as 32-bit two's complement integers. Each bit in an argument is exclusively-ored with the corresponding bit in the other argument. The results of all the exclusive-ors are used to construct the integer which is returned.

BINIOR

IMPORT: iocomasm

This INTEGER **function** returns the bit-by-bit logical inclusive-or of its arguments.

Syntax



Item	Description/Default	Range Restrictions
argument	Expression of TYPE INTEGER.	MININT thru MAXINT

Semantics

The arguments for this function are represented as 32-bit two's complement integers. Each bit in an argument is inclusively-ored with the corresponding bit in the other argument. The results of all the inclusive-ors are used to construct the integer which is returned.

BIT_SET

IMPORT: iocomasm

This BOOLEAN **function** is TRUE if the specified bit position of the argument is equal to 1.

Syntax

Item	Description/Default	Range Restrictions	Recommended Range
argument	Expression of TYPE INTEGER.	MININT thru MAXINT	
bit position	Expression of TYPE INTEGER.	MININT thru MAXINT	0 thru 31

Semantics

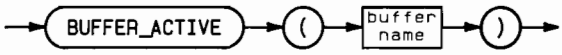
The argument for this function is represented as a 32-bit two's complement integer. Bit 0 is the least-significant bit and bit 31 is the most-significant bit.

BUFFER_ACTIVE

IMPORT: general_4
iodeclarations

This BOOLEAN function is TRUE if there is a transfer active on the specified buffer.

Syntax



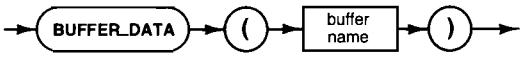
Item	Description/Default	Range Restrictions
buffer name	variable of TYPE buf_info_type	See Chapter 11

BUFFER_DATA

IMPORT: general_4
iodeclarations

This INTEGER **function** returns the number of characters available in the buffer.

Syntax



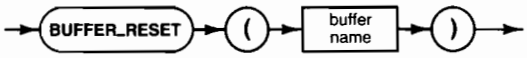
Item	Description/Default	Range Restrictions
buffer name	Variable of TYPE *buf_info_type*.	See Chapter 11

BUFFER_RESET

IMPORT: general_4
iodeclarations

This **procedure** will set the empty and fill pointers to the empty state.

Syntax



Item	Description/Default	Range Restrictions
buffer name	Variable of TYPE *buf_info_type*.	See Chapter 11

Semantics

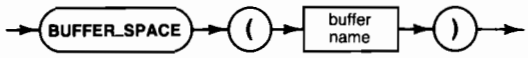
The actual buffer data will not be modified - only the pointers to it. A buffer will only be reset if there are no transfers currently active on the specified buffer.

BUFFER_SPACE

IMPORT: general_4
iodeclarations

This INTEGER **function** returns the available space left in the buffer.

Syntax



Item	Description/Default	Range Restrictions
buffer name	Variable of TYPE *buf_info_type*.	See Chapter 11

Semantics

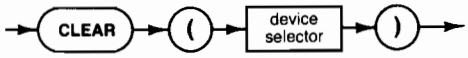
This function not only returns the current available space in the buffer, it also attempts to keep data at the front of the buffer. The buffer is reset if there is no data remaining in the buffer.

CLEAR

IMPORT: hpib_2
iodeclarations

This **procedure** attempts to send a form of the clear message to the specified HP-IB device(s).

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary

Semantics

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN DCL	ATN MTA UNL LAG SDC	ATN DCL	ATN MTA UNL LAG SDC
Not Active Controller	Error			

CLEAR_DISPLAY

IMPORT: dgl_lib

This **procedure** clears the graphics display.

Syntax

Semantics

The graphics system provides the capability to clear the graphics display of all output primitives at any time in an application program. This procedure has different connotations for each graphics display surface. On CRT devices such as the HP 9826, the screen will be erased. Plotters with page advance will advance the paper. On devices such as fixed page plotters, a call to CLEAR_DISPLAY will be ignored.

The starting position is not effected by this procedure.

Error conditions:

1 The graphics system is not initialized.
ACTION: Call ignored.

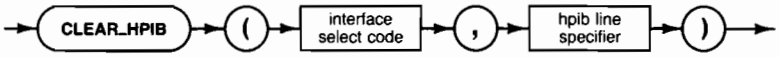
2 The graphics display is not enabled.
ACTION: Call ignored.

CLEAR_HPIB

IMPORT: hpib_0
iodeclarations

This **procedure** will clear the specified HP-IB line. Not all lines are accessable at all times.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
hpib line specifier	Expression of enumerated TYPE *hpib_line*.	atn_line dav_line ndac_line nrfd_line eoi_line srq_line ifc_line ren_line	

Semantics

All possible hpib_line types are not legal when using this procedure.

Handshake lines (DAV, NDAC, NRFD) are never accessible, and an error is generated if an attempt is made to clear them.

The interface clear line (IFC) is automatically cleared after being set, and no action occurs if an attempt is made to clear it through CLEAR_HPIB.

The Service Request line (SRQ) is not accessible through CLEAR_HPIB, and should be accessed through REQUEST_SERVICE. Attempting to clear the service line directly through CLEAR_HPIB generates an error.

The remote enable line (REN) can be cleared only if the selected interface is currently System Controller. Otherwise, an error is generated.

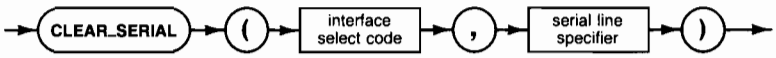
The attention line (ATN) can be cleared only if the selected interface is currently Active Controller. Otherwise, an error is generated.

CLEAR_SERIAL

IMPORT: serial_0
iodeclarations

This **procedure** will clear the specified line on a serial interface card.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
serial line specifier	Expression of enumerated TYPE *type_serial_line*.	rts_line cts_line dcd_line dsr_line drs_line ri_line dtr_line	

Semantics

The values of the enumerated TYPE type_serial_line have the following definitions :

name	RS-232 line
rts	ready to send
cts	clear to send
dcd	data carrier detect
dsr	data set ready
drs	data rate select
dtr	data terminal ready
ri	ring indicator

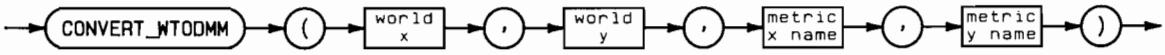
The access to the various lines is determined by the use of an Option 1 or Option 2 connector on the selected interface.

CONVERT_WTODMM

IMPORT: dgl_lib

This **procedure** converts from world coordinates to millimetres on the graphics display.

Syntax



Item	Description/Default	Range Restrictions
world x	Expression of TYPE REAL	–
world y	Expression of TYPE REAL	–
metric x name	Variable of TYPE REAL	–
metric y name	Variable of TYPE REAL	–

Procedure Heading

```
PROCEDURE CONVERT_WTODMM (      WX, WY    : REAL;
                          VAR MmX, MmY : REAL );
```

Semantics

This procedure returns a coordinate pair (**metric X,metric Y**) representing the **world X** and **Y** coordinates. The metric X and Y values are the number of millimetres along the X and Y axis from the supplied world coordinate point to the origin of the metric coordinate system on the device. The location of this origin is device dependent.

For raster devices, the metric origin is the lower-left dot. For HPGL plotters, it is the lower-left corner of pen movement.

Since the origin of the world coordinate system need not correspond to the origin of the physical graphics display, converting the point (0.0,0.0) in the world coordinate system may not result in the value (0.0,0.0) offset from the physical display device's origin.

CONVERT_WTODMM will take any world coordinate point, inside or outside the current window, and convert it to a point offset from the physical display device's origin.

Error conditions:

The graphics system must be initialized and the graphics display must be enabled or the call will be ignored, an ESCAPE (– 27) will be generated, and GRAPHICSERROR will return a non-zero value.

CONVERT_WTOLMM

IMPORT: dgl_lib

This **procedure** converts from world coordinates to millimetres on the locator surface.

Syntax

Item	Description/Default	Range Restrictions
world x	expression of TYPE REAL	–
world y	expression of TYPE REAL	–
metric x name	variable of TYPE REAL	–
metric y name	variable of TYPE REAL	–

Procedure Heading

```
PROCEDURE CONVERT_WTOLMM (      WX, WY    : REAL;
                          VAR MmX, MmY : REAL );
```

Semantics

This procedure returns a coordinate pair (**metric x**,**metric y**) representing the **world X** and **Y** coordinates. The metric x and y values are the number of millimetres along the X and Y axis from the supplied world coordinate point to the origin of the metric coordinate system on the device. The location of this origin is device dependent.

For raster devices, the metric origin is the lower-left dot. For HPGL plotters, it is the lower-left corner of pen movement.

Since the origin of the world coordinate system need not correspond to the origin of the physical locator device, converting the point (0.0,0.0) in the world coordinate system does not necessarily result in the value (0.0,0.0) offset from the physical locator device's origin.

CONVERT_WTOLMM will take any world coordinate point, inside or outside the current window, and convert it to a point offset from the physical locator origin.

Error Conditions

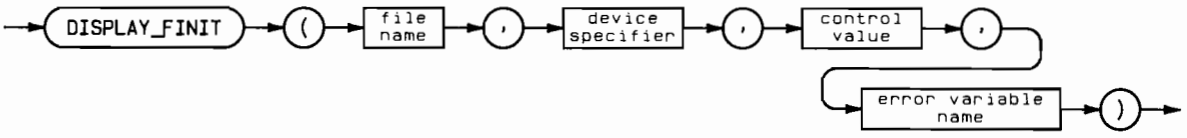
The graphics system must be initialized, the graphics device must be enabled, and the locator must be initialized or the call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

DISPLAY_FINIT

IMPORT: dgl_lib

This **procedure** enables the output of the graphics library to be sent to a file.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
file name	Expression of TYPE *Gstring255*; can be a STRING of any length up to 255 characters.	Must be a valid file name (see "The File System")	–
device specifier	Expression of TYPE *Gstring255*; can be a STRING of any length up to 255 characters. First six characters are significant.	9872A, 9872B, 9872C, 9872S, 9872T, 7470A, 7580A, 7585A	–
control value	Expression of TYPE INTEGER	MININT thru MAXINT	see below
error variable name	Variable of TYPE INTEGER	–	–

Procedure Heading

```
PROCEDURE DISPLAY_FINIT (      File_Name   : Gstring255;
                                 Device_Name: Gstring255;
                                 Control     : INTEGER;
                           var Ierr          : INTEGER );
```

Semantics

DISPLAY_FINIT allows output from the graphics library to be sent to a file. This file can then be sent a graphics display device by use of the operating system's file system (e.g. FILER, or SRM spooler). The contents of the file are device dependent, and MUST be sent only to devices of the type indicated in device name when the file was created.

The **file name** specifies the name of the file to send device dependent commands to.

The **device specifier** tells the graphics system the type of device that the file will be sent to. Only some types of devices may be use this command. For example raster devices (i.e. the internal display) may not use this command. For the currently supported devices, see the range restrictions under Syntax, above.

The **control value** is used to control characteristics of the graphics display device and should be set according to the display device the file is intended for. See "Control Values," below, for the meaning of the control value.

The **error variable name** will contain a value indicating whether the graphics display device was successfully initialized.

Value	Meaning
0	The graphics display device was successfully initialized.
1	The graphics display device (indicated by *device name*) is not supported by the graphics library.
2	Unable to open the file specified. File error is returned in Escapecode, and Ioresult (see the Pascal Language System User's manual).

DISPLAY_FINIT enables a file as the logical graphics display. The file can be of any type, although the current spooling mechanisms can only handle TEXT and ASCII files. The file need not exist before this procedure is called. If this procedure is successful the file will be closed with 'LOCK' when DISPLAY_TERM is executed.

Note

This procedure uses space allocated with the NEW procedure. The application program should call this procedure before any space is allocated for the application program (at least before the MARK procedure is used). If the application program is compiled with the $HEAP_DISPOSE ON$ compiler option, the space will be returned to the system during the next call to the DISPLAY_TERM procedure.

This procedure initializes and enables the graphics display for graphics output. Before the device is initialized the device status is 0, the device address is 0, and the device name is the default name. The default name is ' ' (six ASCII blanks).

When the device is enabled the device status is set to 1 (enabled) and the internal device specifier used by the graphics library is set to the file name provided by the user. The device name is set to the supplied device name. This information is available by calling INQ_WS with operation selectors of 11050 and 12050.

Initialization includes the following operations:

- The graphics display surface is cleared (e.g., CRT erased, plotter page advanced) if Bit 7 of CONTROL is not set.
- The starting position is set to a device dependent location.
- The logical display limits are set to the default limits for the device.
- The aspect ratio of the virtual coordinate system is applied to the logical display limits to define the limits of the virtual coordinate system.
- All primitive attributes are set to the default values.
- The locator echo position is set to its default value.

Only one graphics output device can be initialized at a time. If a graphics display device is currently enabled, the enabled device will be terminated (via DISPLAY_TERM) and the call will continue.

A call to MOVE or INT_MOVE should be made after this call to update the starting position and in so doing, place the physical pen or beam at a known location on the graphics display device.

The Control Value

The control value is used to control characteristics of the graphics display device. Bits should be set according to the following bit map. All unused bits should be set to 0.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits	Meaning
0 thru 6	Currently unused. Should be set to 0.
7	If this bit is set (BIT 7 = 1), it will inhibit clearing of the graphics display as part of the DISPLAY_FINIT procedure. Some devices have the ability to **not** clear the graphics display, or **not** to perform a page advance during device initialization. This bit is ignored on devices that do not support the feature.
8 thru 15	Not used by DISPLAY_FINIT.

HPGL Plotter Initialization

When an HPGL device is initialized the following device dependent actions are performed, in addition to the general initialization process:

- Pen velocity, force, and acceleration are set to the default for that device.
- ASCII character set is set to 'ANSI ASCII'.
- Paper cutter is enabled (HP-9872S / HP-9872T).
- Advance page option is enabled (HP-9872S / HP-9872T).
- Paper is advanced one full page (HP-9872S / HP-9872T) (unless DISPLAY_INIT CONTROL bit 7 is set).
- The automatic pen options are set (HP 7580 / HP 7585).

The default initial dimensions for the HPGL plotters supported by the graphics library are:

Plotter	Wide mm	High mm	Wide points	High points	Aspect	Resolution points/mm
9872	400	285	16000	11400	.7125	40.0
7580	809.5	524.25	32380	20970	.6476	40.0
7585[1]	809.5	524.25	32380	20970	.6476	40.0
7470	257.5	190	10300	7600	.7378	40.0

Any device not in this list is **not** supported.

1 Only "D" size paper can be used in the 7585 when it is used with this call.

The default logical display surface is set equal to the maximum physical limits of the device. The view-surface is always justified in the lower left corner of the current logical display surface (corner nearest the turret for the HP 7580 and HP 7585 plotters). The physical origin of the graphics display is at the lower left boundary of pen movement.

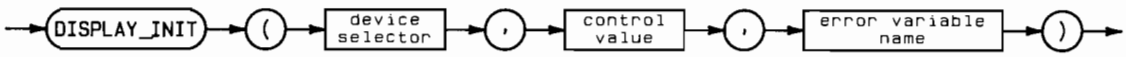
Error Conditions
If the graphics system is not initialized, the call is ignored, an ESCAPE (−27) is generated, and GRAPHICSERROR returns a non-zero value.

DISPLAY_INIT

IMPORT: dgl_lib

This **procedure** enables a device as the logical graphics display.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE INTEGER	MININT to MAXINT	
control value	Expression of TYPE INTEGER	MININT to MAXINT	–
error variable name	Variable of TYPE INTEGER	–	–

Procedure Heading

```
PROCEDURE DISPLAY_INIT (     Dev_Adr : INTEGER;
                             Control : INTEGER;
                         VAR IErr    : INTEGER );
```

Semantics

DISPLAY_INIT enables a device as the logical graphics display. It initializes and enables the graphics display device for graphics output.

Before the device is initialized the device status is 0, the device address is 0, and the device name is the default name. The default name is ' ' (six ASCII blanks).

When the device is enabled the device status is set to 1 (enabled) and the internal device specifier used by the graphics library is set equal to the device selector provided by the user. The device name is set to the device being used. This information is available by calling INQ_WS with operation selectors 11050 and 12050.

The **device selector** specifies the physical address of the graphics output device.

- address = 3 Internal graphics CRT (HP Series 200)
- 8 ≤= device selector ≤= 31 Interface Card Select Code (HP 98627A default = 28)
- 100 ≤= device selector ≤= 3199 composite HPIB/device address

The **control value** is used to control device dependent characteristics of the graphics display device.

The **error variable name** will contain a value indicating whether the graphics display device was successfully initialized.

Value	Meaning
0	The graphics display device was successfully initialized.
2	Unrecognized device specified. Unable to communicate with a device at the specified address, non-existent interface card or non-graphics system supported interface card.

If an error is encountered, the call will be ignored.

The graphics library attempts to directly identify the type of device by using its device selector in some way. The meanings for device address are listed above.

At the time that the graphics library is initialized, all devices which are to be used must be connected, powered on, ready, and accessible via the supplied device selector. Invalid device selectors or unresponsive devices result in that device not being initialized and an error being returned.

Only one graphics output device maybe initialized at a time. If a graphics display device is currently enabled, the enabled device will be terminated (via DISPLAY_TERM) and the call will continue.

A call to MOVE or INT_MOVE should be made after this call to update the starting position and in so doing, place the physical pen or beam at a known location on the graphics display device.

The Control Value

Used to control characteristics of the graphics display device. Bits should be set according to the following bit map. All unused bits should be set to 0.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Bits	Meaning
0 thru 6	Currently unused. Should be set to 0.
7	If this bit is set (BIT 7 = 1), it will inhibit clearing of the graphics display as part of the DISPLAY_FINIT procedure. Some devices have the ability to **not** clear the graphics display, or **not** to perform a page advance during device initialization. This bit is ignored on devices that do not support the feature.
8 thru 15	Bits 8 though 15 are used by some devices to control device dependent features of those devices.

Bits 8,9, and 10 of DISPLAY_INIT's CONTROL parameter determine the type of display for the HP 98627A card and the default dimensions assumed by the graphics system.

CONTROL	Bits 10 9 8	Description	
256	001	US STD	(512 x 390, 60 hz refresh)
512	010	EURO STD	(512 x 390, 50 hz refresh)
768	011	US TV	(512 x 474, 15.75 Khz horizontal refresh, interlaced)
1024	100	EURO TV	(512 x 512, 50 hz vertical refresh, interlaced)
1280	101	HI RES	(512 x 512, 60 hz)
1536	110	Internal	(HP) use only

Out of range values are treated as if CONTROL = 256.

Note

This procedure uses space allocated with the NEW procedure. The application program should call this procedure before any space is allocated for the application program (at least before the MARK procedure is used). If the application program is compiled with the $HEAP_DISPOSE ON$ compiler option, the space will be returned to the system during the next call to the DISPLAY_TERM procedure.

General Initialization Operations

Initialization includes the following operations:

- The graphics display surface is cleared (e.g., CRT erased, plotter page advanced) unless Bit 7 of the control value is set.
- The starting position is set to a device dependent location. (This is undefined for HPGL plotters.)
- The logical display limits are set to the default limits for the device.
- The aspect ratio of the virtual coordinate system is applied to the logical display limits to define the limits of the virtual coordinate system.
- All primitive attributes are set to the default values.
- The locator echo position is set to its default value.
- If the display and locator are the same physical device, the logical locator limits are set to the limits of the view surface.

Raster Display Initialization

When a raster display is initialized the following device dependent actions are performed, in addition to the general initialization process:

- The starting position is in the lower left corner of the display.
- Graphics memory is cleared if bit 7 of the control word is 0.
- Initialize the color table to default values. If the device has retroactive color definition (Model 36C) and the color table has been changed from the default colors, the colors of an image will change even if bit 7 is set to 1.
- The graphics display is turned on.
- The view surface is centered within the logical display limits.

- The drawing mode (see OUTPUT_ESC) is set to dominate.
- The DISPLAY_INIT CONTROL parameter is used as specified above.

The following table describes the internal raster displays for Series 200 computers:

	Wide mm	High mm	Wide points	High points	Memory Planes	Color Map
9816	168	126	400	300	1	no
9826	120	90	400	300	1	no
9836	210	160	512	390	1	no
9836C	217	163	512	390	4	yes

The HP 98627A is a 3 plane non-color mapped color interface card which connects to an external RGB monitor. Bits 8,9, and 10 of DISPLAY_INIT's CONTROL parameter determine the type of display for the HP 98627A card and the default dimensions assumed by the graphics system.

CONTROL	Bits 10 9 8	Description	
256	001	US STD	(512 x 390, 60 hz refresh)
512	010	EURO STD	(512 x 390, 50 hz refresh)
768	011	US TV	(512 x 474, 15.75 Khz horizontal refresh, interlaced)
1024	100	EURO TV	(512 x 512, 50 hz vertical refresh, interlaced)
1280	101	HI RES	(512 x 512, 60 hz)
1536	110	Internal	(HP) use only

Out of range values are treated as if CONTROL = 256.

The physical size of the HP 98627A display (needed by the SET_DISPLAY_LIM procedure) may be given to the graphics system by an escape function (OPCODE = 250). The physical limits assumed until the escape function is given are:

CONTROL =	256	153.3mm wide and 116.7mm high.
	512	153.3mm wide and 116.7mm high.
	768	153.3mm wide and 142.2mm high.
	1280	153.3mm wide and 153.3mm high.

The default logical display surface of the graphics display device is the maximum physical limits of the screen. The physical origin is the lower left corner of the display.

The view surface is always centered within the current logical display surface.

HPGL Plotter Initialization

When an HPGL device is initialized the following device dependent actions are performed, in addition to the general initialization process:

- Pen velocity, force, and acceleration are set to the default for that device.
- ASCII character set is set to 'ANSI ASCII'.
- Paper cutter is enabled (HP-9872S / HP-9872T).
- Advance page option is enabled (HP-9872S / HP-9872T).
- Paper is advanced one full page (HP-9872S / HP-9872T) (unless DISPLAY_INIT CONTROL bit 7 is set).
- The automatic pen options are set (HP 7580 / HP 7585).

The default initial dimensions for the HPGL plotters supported by the graphics library are:

Plotter	Wide mm	High mm	Wide points	High points	Aspect	Resolution points/mm
9872	400	285	16000	11400	.7125	40.0
7580	809.5	524.25	32380	20970	.6476	40.0
7585	1100	890	44000	35670	.809	40.0
7470	257.5	190	10300	7600	.7378	40.0

The maximum physical limits of the graphics display for an HPGL device not listed above are determined by the default settings of P1 and P2. The default settings of P1 and P2 are the values they have after an HPGL 'IN' command. Refer to the specific device manual for additional details.

The default logical display surface is set equal to the area defined by P1 and P2 at the time DISPLAY_INIT is invoked. The view-surface is always justified in the lower left corner of the current logical display surface (corner nearest the turret for the HP 7580 and HP 7585 plotters). The physical origin of the graphics display is at the lower left boundary of pen movement.

Note

If the paper is changed in an HP 7580 or HP 7585 plotter while the graphics display is initialized, it should be the same size of paper that was in the plotter when DISPLAY_INIT was called. If a different size of paper is required, the device should be terminated (DISPLAY_TERM) and re-initialized after the new paper has been placed in the plotter.

Error Conditions

The graphics system must be initialized or the call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

DISPLAY_TERM

IMPORT: dgl_lib

This **procedure** disables the enabled graphics display device.

Syntax

Procedure Heading

```
PROCEDURE DISPLAY_TERM;
```

Semantics

DISPLAY_TERM terminates the device enabled as the graphics display. DISPLAY_TERM completes all remaining display operations and disables the logical graphics display. It makes the picture current and releases all resources being used by the device. The device name is set to the default name ' ' (six ASCII blanks), the device status is set to 0 (not enabled) and the device address is set to 0. DISPLAY_TERM does not clear the graphics display.

The graphics display device should be disabled before the termination of the application program. DISPLAY_TERM is the complementary routine to DISPLAY_INIT.

Error Conditions

The graphics system should be initialized and the display should be enabled or the call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

DMA_RELEASE

IMPORT: iocomasm
iodeclarations

Note

This function is provided for use by the internal I/O Procedure Library drivers, only. Unexpected and possible undesirable results may occur if it is used.

DMA channel allocation and deallocation occur automatically in the I/O library.

DMA_REQUEST

IMPORT: iocomasm
iodeclarations

Note

This function is provided for use by the internal I/O Procedure Library drivers, only. Unexpected and possible undesirable results may occur if it is used.

DMA channel allocation and deallocation occur automatically in the I/O library.

END_SET

IMPORT: hpib_1
iodeclarations

This BOOLEAN **function** indicates whether or not EOI was set on the last byte read – this is **not** a current indication of the EOI line.

Syntax

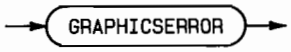
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

GRAPHICSERROR

IMPORT: dgl_lib

This **function** returns and integer error code and can be used to determine the cause of a graphics escape.

Syntax



Function Heading

```
FUNCTION GRAPHICSERROR: INTEGER;
```

Semantics

When an error occurs that uses the escape function, escape-code −27 is used. After the escape is trapped and it has been determined that the graphics library is the source of the error (the escape code equal to −27), GRAPHICSERROR can be used to determine the cause of the error. The function returns the value of the last error generated and then clears the value of the return error. A user who is trapping errors and wishes to keep the value of the error must save it in some variable.

The following list of returned values and the error they represent can be used to interpret the value returned by GRAPHICSERROR.

Value	Meaning
0	No errors since the last call to GRAPHICSERROR or since the last call to GRAPHICS_INIT.
1	The graphics system is not initialized. ACTION: CAll ignored.
2	The graphics display is not enabled. ACTION: Call ignored.
3	The locator device is not enabled. ACTION: Call ignored.
4	Echo value requires a graphics display to be enabled. ACTION: Call completes with echo value = 1.
5	The graphics system is already initialized. ACTION: Call ignored.
6	Illegal aspect ratio specified. X-SIZE and Y-SIZE must be greater than 0. ACTION: Call ignored.
7	Illegal parameters specified. ACTION: Call ignored.
8	The parameters specified are outside the physical display limits. ACTION: Call ignored.
9	The parameters specified are outside the limits of the window. ACTION: Call ignored.
10	The logical locator and the logical display are the same physical device. The logical locator limits cannot be defined explicitly, they must correspond to the logical view surface limits. ACTION: Call ignored.

11	The parameters specified are outside the current virtual coordinate system boundary. ACTION: Call ignored.
13	The parameters specified are outside the physical locator limits. ACTION: Call ignored.
14	Color table contents cannot be inquired or changed. ACTION: Call ignored.
18	The number of points specified for a polygon or polyline operation is less than or equal to zero. ACTION: Call ignored.

GRAPHICS_INIT

IMPORT: dgl_lib

This **procedure** initializes the graphics system.

Syntax

Procedure Heading

```
PROCEDURE GRAPHICS_INIT;
```

Semantics

GRAPHICS_INIT initializes the graphics system. It must be the first graphics system call made by the application program. Any procedure call other than GRAPHICS_INIT will be ignored. GRAPHICS_INIT performs the following operations:

- Get dynamic storage space for the graphics library.
- Sets the aspect ratio to 1.
- Sets the virtual coordinate and viewport limits to range from 0 to 1.0 in the X and Y directions.
- Sets the world coordinate limits to range from −1.0 to 1.0 in the X and Y directions.
- Sets the starting position to (0.0,0.0) in world coordinate system units.
- Sets all attributes equal to their default values.

GRAPHICS_INIT does not enable any logical devices. The graphics system is terminated with a call to GRAPHICS_TERM. Calling GRAPHICS_INIT while the graphics system is initialized will result in an implicit call to GRAPHICS_TERM, before the system is reinitialized.

Note

Space is allocated for the graphics system using the standard Pascal procedure, NEW. The application program should call this procedure before any space is allocated for the application program. If memory allocated at graphics_init is to be returned at graphics_term, the compiler option $HEAP_DISPOSE ON$ must be used.

GRAPHICS_TERM

IMPORT: dgl_lib

This **procedure** terminates the graphics system.

Syntax

Procedure Heading

```
PROCEDURE GRAPHICS_TERM;
```

Semantics

GRAPHICS_TERM terminates the graphics system. Termination includes terminating both the graphics display and the locator devices. GRAPHICS_TERM does not clear the graphics display.

GRAPHICS_TERM should be called as the last graphics system call in the application program.

GRAPHICS_TERM releases dynamic memory allocated during GRAPHICS_INIT. In order that this memory actually be returned the compiler option $HEAP_DISPOSE ON$ must be used.

Error Conditions

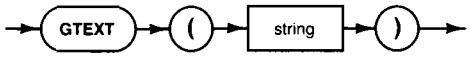
If the graphics system is not initialized, the call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

GTEXT

IMPORT: dgl_types dgl_lib

This **procedure** draws characters on the graphics display.

Syntax



Item	Description/Default	Range Restrictions
string	Expression of TYPE *Gstring255*. Can be a string of any length up to 255 characters	length <= 255 characters

Procedure Heading

```
PROCEDURE GTEXT ( String : Gstring255 );
```

Semantics

The **string** contains the characters to be output.

GTEXT produces characters on the graphics display. A series of vectors representing the characters in the string is produced by the graphics system.

When the text string is output, the starting position will represent the lower left-hand corner of the first character in STRING. Text is normally output from left to right and is printed vertically with no slant.

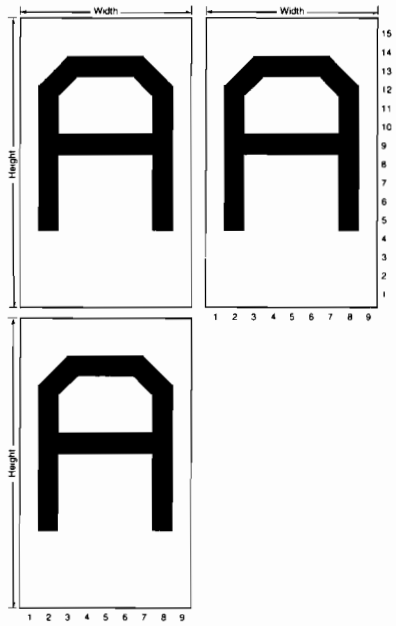
After completion of this call, the starting position is left in a device dependent location such that successive calls to GTEXT will produce a continuous line of text (i.e., `GTEXT('H'); GTEXT('I');` is equivalent to `GTEXT('HI');`).

The attributes of color, line-style, line-width, text rotation, and character size apply to text primitives. However, the text will appear with these attributes only if the graphics device is capable of applying them to text.

Characters

The character sets provided by the graphics system are the same ones used by the CRT in alpha mode, namely the standard character set plus either the Roman extension character set (for all non-Katakana machines) or the Katakana character set (for Katakana machines).

Characters are defined within a cell that has an aspect ratio of 9/15. The character cells are adjacent, both horizontally and vertically, as shown here.



Control Codes

The following control codes are supported by GTEXT:

Control Character	Program Access	Keyboard Access	Action
backspace	CHR(8)	CTRL-H	Move one character cell to the left along the text direction vector (defined by SET_CHAR_SIZE).
linefeed	CHR(10)	CTRL-J	Move down the height of one character cell.
carriage return	CHR(13)	CTRL-M	Move back the length of the text just completed.

Any other control characters are ignored.

The current position is maintained to the resolution of the display device. A text size less-than-or-equal-to the resolution of the display device will result in all the characters in a GTEXT call, or a series of GTEXT calls, being written to the same point on the device.

The current position returned by an INQ_WS is **not** updated by calls to GTEXT. If you want to know the current position after a GTEXT, you must do a MOVE, or some other call which updates the current position.

Error Conditions

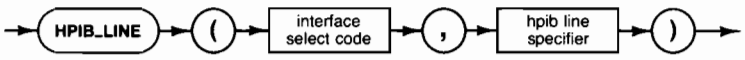
If the graphics system is not initialized or a display is not enabled, the call will be ignored, an ESCAPE (– 27) will be generated, and GRAPHICSERROR will return a non-zero value.

HPIB_LINE

IMPORT: hpib_0
iodeclarations

This BOOLEAN **function** will return the current state of the specified line. Not all lines are accessible at all times.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE type_isc. This is an INTEGER subrange.	0 thru 31	7 thru 31
hpib line specifier	Expression of enumerated TYPE *hpib_line*.	atn_line dav_line ndac_line nrfd_line eoi_line srq_line ifc_line ren_line	

Semantics

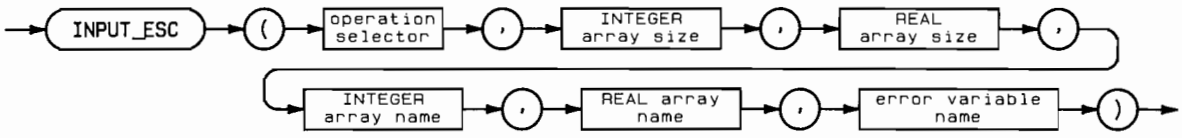
The lines are only accessible when the hardware buffer is pointing into the interface. For example, REN can only be examined when the selected interface is **not** system controller. No error is generated when an in-accessible line is examined.

INPUT_ESC

IMPORT: dgl_lib

This **procedure** allows the user to obtain device dependent information from the graphics system.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
operation selector	Expression of TYPE INTEGER	MININT to MAXINT	–
INTEGER array size	Expression of TYPE INTEGER	MININT to MAXINT	>0
REAL array size	Expression of TYPE INTEGER	MININT to MAXINT	>0
INTEGER array name	Variable of TYPE ANYVAR should be array of INTEGERs	–	–
REAL array name	Variable of TYPE ANYVAR should be array of REAL	–	–
error variable name	Variable of TYPE INTEGER	–	–

Procedure Heading

```
PROCEDURE INPUT_ESC (       Opcode : INTEGER;
                      Isize : INTEGER;
                      Rsize : INTEGER;
                      ANYVAR Ilist : Gint_list;
                      ANYVAR Rlist : Greal_list;
                      VAR Ierr  : INTEGER    );
```

Semantics

The **operation selector** determines the device dependent inquiry escape function being invoked.

The **INTEGER array size** is the number of INTEGER parameters to be returned in the INTEGER array by the escape function. The correct value for this can be found in the hundred's place of the operation selector (see the table below).

The **REAL array size** is the number of REAL parameters to be returned in the REAL array by the escape function. The correct value for this can be found in the thousand's place of the operation selector (see the table below).

The **INTEGER array** is the array in which zero or more INTEGER parameters are returned by the escape function.

The **REAL array** is the array in which zero or more REAL parameters are returned by the escape function.

The **error variable** will contain a code indicating whether the input escape function was performed.

Value	Meaning
0	Inquiry escape function successfully completed.
1	Inquiry operation (operation selector) not supported by the graphics display device.
2	INTEGER array size is not equal to the number of INTEGER parameters to be returned.
3	REAL array size is not equal to the number of REAL parameters to be returned.

If the error variable contains a non-zero value, the call has been ignored.

INPUT_ESC allows application programs to access special device features on a graphics display device. The type of information returned from the graphics display device is determined by the value of operation selector. Possible inquiry escape functions may return the status or the options supported by a particular graphics display device.

Inquiry escape functions only apply to the graphics display device. INPUT_ESC implicitly makes the picture current before the escape function is performed.

HPGL Plotter Operation Selectors

The following inguiry is supported:

Operation Selector	Meaning
2050	Inquire about current turret. INTEGER array [1] = −1 >> Turret mounted, but its type is unknown INTEGER array [1] = 0 >> No turret mounted INTEGER array [1] = 1 >> Fiber tip pens INTEGER array [1] = 2 >> Roller ball pens INTEGER array [1] = 3 >> Capillary pens INTEGER array [2] = 0 >> No turret mounted or turret has no pens INTEGER array [2] = n >> Sum of these values: 1: Pen in stall #1 2: Pen in stall #2 4: Pen in stall #3 8: Pen in stall #4 16: Pen in stall #5 32: Pen in stall #6 64: Pen in stall #7 128: Pen in stall #8

For example, if INTEGER array[2] = 3, pens would only be contained in stalls 1 and 2.

Operation selector 2050 is only supported on the HP 7580 and HP 7585 plotters.

Error Conditions

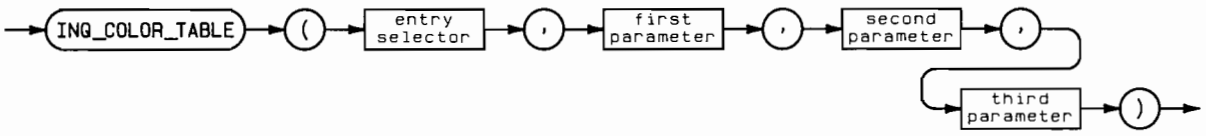
If the graphics system is not initialized or a display is not enabled, the call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

INQ_COLOR_TABLE

IMPORT: dgl_lib
dgl_inq

This **procedure** inquires the color modeling parameters for an index into the device-dependent color capability table.

Syntax



Item	Description/Default	Range Restrictions
entry selector	Expression of TYPE INTEGER	>0
first parameter name	Variable of TYPE REAL	–
second parameter name	Variable of TYPE REAL	–
third parameter name	Variable of TYPE REAL	–

Procedure Heading

```
PROCEDURE INQ_COLOR_TABLE (    Index : INTEGER;
                           VAR Colp1 : REAL;
                           VAR Colp2 : REAL;
                           VAR Colp3 : REAL    );
```

Semantics

This routine inquires the color modelling parameters for the specified location in a device-dependent color capability table.

The **entry selector** specifies the location in the color capability table. The parameters returned are for the specified location. The size of the color capability table is device dependent. For raster displays in Series 200 computers, 32 entries are available.

The **first parameter** represents red intensity if the RGB model has been selected with the SET COLOR statement, or hue if the HSL model has been selected.

The **second parameter** represents green intensity if the RGB model has been selected with the SET COLOR statement, or saturation if the HSL model has been selected.

The **third parameter** represents blue intensity if the RGB model has been selected, or luminosity if the HSL model has been selected.

A more detailed description of the color models and the meaning of their parameters can be found under the procedure definition of SET_COLOR_MODEL.

Note

The color table stores color specifications as RGB values. The conversion from RGB to HSL is a one-to-many transformation, and the following arbitrary assignments may be made during the conversion:

```
IF Luminosity = 0
   THEN  Hue = 0
         Saturation = 0

IF Saturation = 0
   THEN  Hue = 0
```

Error Conditions

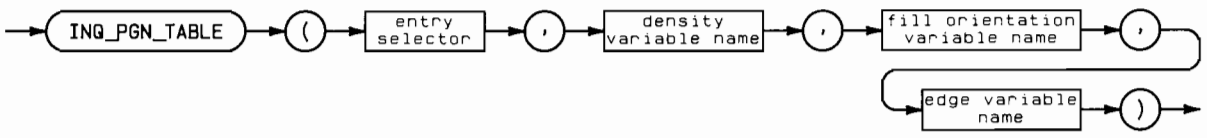
If the graphics system is not initialized, a display device is not enabled, the color table contents cannot be inquired, or the color table entry selector is out of range, the call is ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

INQ_PGN_TABLE

IMPORT: dgl_lib
dgl_inq

This **procedure** inquires the polygon style attributes for an entry in the polygon style table.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
entry selector	Expression of TYPE INTEGER	MININT thru MAXINT	Device dependent
density variable name	Variable of TYPE REAL	–	–
fill orientation variable name	Variable of TYPE REAL	–	–
edge variable name	Variable of TYPE INTEGER	–	–

Procedure Heading

```
PROCEDURE INQ_PGN_TABLE (     Index  : INTEGER;
                          VAR Densty : REAL;
                          VAR Orient : REAL;
                          VAR Edge   : INTEGER );
```

Semantics

The **entry selector** specifies the entry in the polygon style table the inquiry is directed at.

The **density variable** will contain a value between -1 and 1. This magnitude of this value is the ratio of filled area to non-filled area. Zero means the polygon interior is not filled. One represents a fully filled polygon interior. All non-zero values specify the density of continuous lines used to fill the interior. Negative values are used to specify crosshatching. Calculations for fill density are based on the thinnest line possible on the device and on continuous line-style. If the interior line-style is not continuous, the actual fill density may not match that found in the polygon style table.

The **fill orientation variable** will contain a value from -90 through 90. This value represents the angle (in degrees) between the lines used for filling the polygon and the horizontal axis of the display device. The interpretation of fill orientation is device-dependent. On devices that require software emulation of polygon styles, the angle specified will be adhered to as closely as possible, within the line-drawing capabilities of the device. For hardware generated polygon styles, the angle specified will be adhered to as closely as is possible given the hardware simulation of the requested density. If crosshatching is specified, the fill orientation specifies the angle of orientation of the first set of lines in the crosshatching, and the second set of lines is always perpendicular to this.

The **edge variable** will contain a 0 if the polygon edge is not to be displayed and a 1 if the polygon edge is to be displayed. If polygon edges are displayed, they adhere to the current line attributes of color, line-style, and line-width, in effect at the time of polygon display.

All current devices support 16 entries in the polygon table. The polygon styles defined in the default tables are defined to exploit the hardware capabilities of the devices they are defined for.

Error Conditions

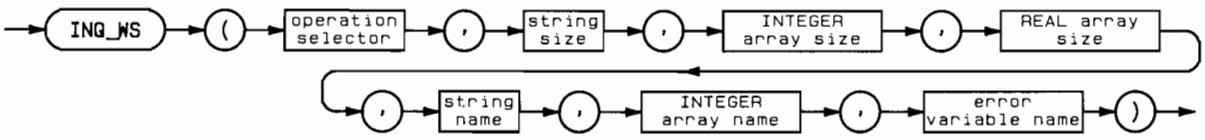
The graphics system must be initialized, a display must be enabled, and the entry selector must be in range or the call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

INQ_WS

IMPORT: dgl_lib
dgl_inq

This **procedure** allows the user to determine characteristics of the graphics system.

Syntax



Item	Description/Default	Range Restrictions
operation selector	Expression of TYPE INTEGER	see below
string size	Expression of TYPE INTEGER	see below
integer array size	Expression of TYPE INTEGER	see below
REAL array size	Expression of TYPE INTEGER	see below
string name	Variable of TYPE PACKED ARRAY OF CHAR	–
INTEGER array name	Variable of TYPE ARRAY OF INTEGER	–
REAL array name	Variable of TYPE ARRAY OF REAL	–
error variable name	Variable of TYPE INTEGER	–

Procedure Heading

```
PROCEDURE INQ_WS (          Opcode : INTEGER;
                            Ssize  : INTEGER;
                            Isize  : INTEGER;
                            Rsize  : INTEGER;
                  ANYVAR    Slist  : Gchar_list;
                  ANYVAR    Ilist  : Gint_list;
                  ANYVAR    Rlist  : Greal_list;
                  VAR       Ierr   : INTEGER);
```

Semantics

The **operation selector** is an integer from the list of operation selectors given below. It is used to specify the topic of the inquiry to the system.

The **string size** is used to specify the maximum number of characters that are to be returned in the string array by the function specified by the operation selector. If there is a 1 in the ten-thousand's place a string value will be returned. The number of characters in the string is returned in the first entry in the INTEGER arrray.

The **INTEGER array size** is the number of integer parameters that are returned in the integer array by the function specified by OPCODE. The thousand's digit of the operation selector is the number of elements the INTEGER array must contain.

The **REAL array size** is the number of REAL parameters that are returned in the REAL array by the function specified by OPCODE. The hundred's digit of the operation selector is the number of elements the REAL array must contain.

The **string array** is a PACKED ARRAY OF CHAR which will contain a string or strings that represents characteristics of the work station specified by the value of operation selector. The application program must ensure that string array is dimensioned to contain all of the values returned by the selected function.

The **INTEGER array** will contain integer values that represent characteristics of the work station specified by the value of OPCODE. The application program must ensure that the integer array is dimensioned to contain all of the values returned by the selected function.

The **REAL array** will contain REAL values that represent characteristics of the work station specified by the value of OPCODE. The application program must ensure that the REAL array is dimensioned to contain all of the values returned by the selected function.

The **error variable** will return an integer indicating whether the inquiry was successfully performed.

Value	Meaning
0	The inquiry was successfully performed.
1	The operation selector was invalid.
2	The INTEGER array size was not equal to the number INTEGER parameters requested by the operation selector.
3	The REAL array size was not equal to the number of REAL parameters requested by the operation selector.
4	The string array was not large enough to hold the string requested by the operation selector.

The procedure INQ_WS returns current information about the graphics system to the application program. The type of information desired is specified by a unique value of OPCODE. The thousands digit of the operation selector specifies the number of integer values returned in the integer array and the hundreds digit specifies the number of REAL values returned in the REAL array. A 1 in the ten-thousand's place indicates that a value will be returned in the string.

One use of INQ_WS is device optimization: the use of inquiry to enhance the application's utilization of the output device. An example of this is using color to distinguish between lines when a device supports colors, and using line-styles when color is not available. Another example is maximizing the aspect ratio used, based on the maximum aspect ratio of the display device.

Device dependent information returned by the procedure is undefined if the device being inquired from is not enabled (e.g., inquire number of colors supported, operation selector 1053, only returns valid information when the display is enabled).

If the graphics system is not initialized, the call will be ignored and GRAPHICSERROR will return a non-zero value.

Supported Operation Selectors

The operation selectors supported by the system and their meaning is listed below:

Operation Selector	Meaning
250	Current cell size used for text. REAL Array[1] = Character cell width in world coordinates REAL Array[2] = Character cell height in world coordinates
251	Marker size. REAL Array[1] = Marker width in world coordinates REAL Array[2] = Marker height in world coordinates
252	Resolution of graphics display REAL Array[1] = Resolution in X direction (points/mm) REAL Array[2] = Resolution in Y direction (points/mm)
253	Maximum dimensions of the graphics display. REAL Array[1] = Maximum size in X direction (MM) REAL Array[2] = Maximum size in Y direction (MM)
254	Aspect ratios REAL Array[1] = Current aspect ratio of the virtual coordinate system. REAL Array[2] = Aspect ratio of logical limits.
255	Resolution of locator device REAL Array[1] = Resolution in X direction (points/mm) REAL Array[2] = Resolution in Y direction (points/mm)
256	Maximum dimensions of the locator display. REAL Array[1] = Maximum size in X direction (MM) REAL Array[2] = Maximum size in Y direction (MM)
257	Current locator echo position REAL array[1] = X world coordinate position REAL array[2] = Y world coordinate position
258	Current virtual coordinate limits REAL array[1] = Maximum X virtual coordinate REAL array[2] = Maximum Y virtual coordinate
259	Starting position. The information returned may not be valid (not updated) following a text call, an escape function call, changes to the viewing transformation or after initialization of the graphics display device. REAL array[1] = X world coordinate position REAL array[2] = Y world coordinate position
450	Current window limits REAL array[1] = Minimum X world coordinate position REAL array[2] = Maximum X world coordinate position REAL array[3] = Minimum Y world coordinate position REAL array[4] = Maximum Y world coordinate position
451	Current viewport limits REAL array[1] = Minimum X virtual coordinate REAL array[2] = Maximum X virtual coordinate REAL array[3] = Minimum Y virtual coordinate REAL array[4] = Maximum Y virtual coordinate

Operation Selector	Meaning
1050	Does graphics display device support clipping at physical limits? INTEGER Array[1] = 0 - No INTEGER Array[1] = 1 - Yes, to the view-surface boundaries INTEGER Array[1] = 2 - Yes, but only to the physical limits of the display surface.
1051	Justification of the view surface within the logical display limits. INTEGER Array[1] = 0 - View-surface is centered within the logical display limits INTEGER Array[1] = 1 - View surface is positioned in the lower left corner of the logical display limits.
1052	Can the graphics display draw in the background color? Drawing in the background color can be used to 'erase' previously drawn primitives. INTEGER Array[1] = 0 - No INTEGER Array[1] = 1 - Yes
1053	The total number of non-dithered colors supported on the graphics display. The number returned does not include the background color. (Compare operation selectors 1053, 1054, and 1075.) INTEGER Array[1] = number of distinct colors supported.
1054	Number of distinct non-dithered colors which can appear on the graphics display at one time. The number returned does not include the background color. INTEGER Array[1] = number of distinct colors which can appear on the display device at one time.
1056	Number of line-styles supported on the graphics display. INTEGER Array[1] = number of hardware line-styles supported.
1057	Number of line-widths supported on the graphics display. INTEGER Array[1] = number of line-widths supported.
1059	Number of markers supported on the graphics display. INTEGER Array[1] = # of distinct markers supported.
1060	Current value of color attribute. INTEGER Array[1] = Current value of color attribute.
1062	Current value of line-style attribute INTEGER Array[1] = Current value of line-style attribute.
1063	Current value of line-width attribute. INTEGER Array[1] = Current value.
1064	Current timing mode. INTEGER Array[1] = 0 - Immediate visibility INTEGER Array[1] = 1 - System buffering
1065	Number of entries in the polygon style table. INTEGER Array[1] = # styles.
1066	Current polygon interior color index. INTEGER Array[1] = Index

Operation Selector	Meaning
1067	Current polygon style index. INTEGER Array[1] = Index
1068	Maximum number of polygon vertices that a display device can process. INTEGER Array[1] = 0 — No hardware support. = N (0<n<32767) — Number of vertices supported. = 32767 — The graphics display device uses all available memory to process polygons (the maximum number of vertices is determined by current free memory).
1069	Does the graphics device support immediate, retroactive change of polygon style for polygons already displayed? INTEGER Array[1] = 0 - No. INTEGER Array[1] = 1 - Yes.
1070	Does the graphics device support hardware (or low-level device handler) generation of polygons using INT_POLYGON_DD? INTEGER Array[1] = 0 - No INTEGER Array[1] = 1 - Yes
1071	Does the graphics device support immediate, retroactive change for primitives already displayed? INTEGER Array[1] = 0 - No INTEGER Array[1] = 1 - Yes
1072	Can the background color of the display be changed? INTEGER Array[1] = 0 - No INTEGER Array[1] = 1 - Yes
1073	Can entries in the color table be redefined using SET_COLOR_TABLE? INTEGER Array[1] = 0 - No INTEGER Array[1] = 1 - Yes
1074	Current color model in use. INTEGER Array[1] = 1 - RGB INTEGER Array[1] = 2 - HSL
1075	Number of entries in the color capability table. The number returned does not include the background color. INTEGER Array[1] = # entries
1076	Current polygon interior line-style. INTEGER Array[1] = Current interior line-style
11050	Graphics display device association. String = Name of device path. (Internal device specifier.) INTEGER Array[1] = Number of characters in the device path.
11052	Locator device association. String = Name of device path. (Internal device specifier.) INTEGER Array[1] = Number of characters in the device path.

Operation Selector	Meaning
12050	Graphics display device information. String = Name of graphics display device. INTEGER Array[1] = Number of characters in the device name. INTEGER Array[2] = Status = 0 Graphics display is not enabled. = 1 Graphics display is enabled.
13052	Graphics locator device information. String = Name of the locator device. INTEGER Array[1] = Number of characters in the device name. INTEGER Array[2] = Status = 0 Locator device is not enabled. = 1 Locator device is enabled. INTEGER Array[3] = Number of buttons on the locator device.

Error Conditions

If the graphics system is not initialized, the call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

INT_LINE

IMPORT: dgl_types
dgl_lib

This **procedure** draws a line from the starting position to the world coordinate specified.

Syntax

Item	Description/Default	Range Restrictions
x coordinate	Expression of TYPE *Gshortint*; This is subrange of INTEGER	−32 768 to 32 767
y coordinate	Expression of TYPE *Gshortint*; This is subrange of INTEGER	−32 768 to 32 767

Procedure Heading

```
PROCEDURE INT_LINE ( Iwx, Iwy : Gshortint );
```

Semantics

The **x** and **y coordinate** pair is the ending of the line to be drawn in the world coordinate system.

A line is drawn from the starting position to the world coordinate specified by the x and y coordinates. The starting position is updated to this point at the completion of this call.

The primitive attributes of line style (see SET_LINE_STYLE), line width (see SET_LINE_WIDTH), and color (see SET_COLOR) apply to lines drawn using INT_LINE.

This procedure is the same as the LINE procedure, with the exception that the parameters are of type *Gshortint* (−32 768..32 767). When used with some displays this procedure may perform about 3 times faster than the LINE procedure. For all other displays this procedure has about the same performance as the LINE procedure.

The INT_LINE procedure only has increased performance when the following conditions exist:

- The display must be a raster device.
- The window bounds within the range −32 768 to 32 767.
- The window must be less then 32 767 units wide and high.

INT operations are provided for efficient vector generation. Although their use can be mixed with other, non-integer operations, one dot roundoff errors may result with mixed use since different algorithms are used to implement each.

Drawing to the starting position generates the shortest line possible. Depending on the nature of the current line-style, nothing may appear on the graphics display surface. See SET_LINE_STYLE for a complete description of how line-style affects a particular point or vector.

INT_MOVE

IMPORT: dgl_types
dgl_lib

This **procedure** sets the starting position to the world coordinate position specified.

Syntax

Item	Description/Default	Range Restrictions
x coordinate	Expression of TYPE *Gshortint*; This is subrange of INTEGER	−32 768 to 32 767
y coordinate	Expression of TYPE *Gshortint*; This is subrange of INTEGER	−32 768 to 32 767

Procedure Heading

```
PROCEDURE INT_MOVE ( Iwx, Iwy : INTEGER );
```

Semantics

The **x** and **y coordinate** pair define the new starting position in world coordinates.

INT_MOVE specifies where the next graphical primitive will be output. It does this by setting the value of the starting position to the world coordinate system point specified by the x and y coordinate values and then moving the pen (or its logical equivalent) to that point.

The starting position corresponds to the location of the physical pen or beam in all but four instances: after a change in the viewing transformation, after initialization of a graphical display device, after the output of a text string, or after the output of an escape function. A call to MOVE or INT_MOVE should therefore be made after any one of the following calls to update the value of the starting position and in so doing, place the physical pen or beam at a known location: SET_ASPECT, DISPLAY_INIT, SET_DISPLAY_LIM, OUTPUT_ESC, TEXT, SET_VIEWPORT, and SET_WINDOW.

This procedure is the same as the MOVE procedure, with the exception that the parameters are of type *Gshortint* (−32 768..32 767). When used with the same display, this procedure can perform about 3 times faster than the MOVE procedure. For all other displays this procedure has about the same performance as the MOVE procedure.

The INT_MOVE procedure only has increased performance when the following conditions exist:

- The display must be a raster device.
- The window bounds within the range −32 768 to 32 767.
- The window must be less than 32767 units wide and high.

INT operations are provided for efficient vector generation. Although their use can be mixed with non-integer operations, one dot roundoff errors may result with mixed use since different algorithms are used to implement each.

Error Conditions

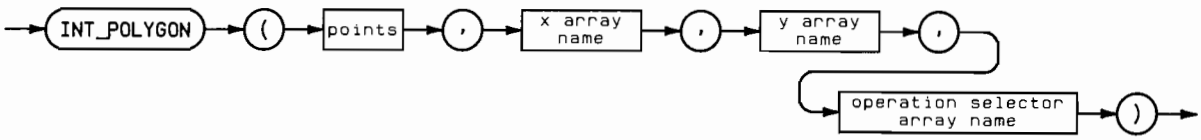
The graphics system must be initialized and a graphics display must be enabled or the call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

INT_POLYGON

IMPORT: dgl_types
dgl_lib
dgl_poly

This **procedure** displays a polygon-set starting and ending at the specified point adhering to the specified polygon style exactly as specified (i.e., device-independent results).

Syntax



Item	Description/Default	Range Restrictions
points	Expression of TYPE INTEGER	MININT thru MAXINT
x array name	Array of TYPE *Gshortint_list*. *Gshortint* is a sub-range of INTEGER.	−32 768 to 32 767
y array name	Array of TYPE *Gshortint_list*. *Gshortint* is a sub-range of INTEGER.	−32 768 to 32 767
operation selector array name	Array of TYPE *Gshortint_list*. *Gshortint* is a sub-range of INTEGER.	−32 768 to 32 767

Procedure Heading

```
PROCEDURE INT_POLYGON ( Npoint     : INTEGER;
                  ANYVAR   Xvec    : Gshortint_list;
                  ANYVAR   Yvec    : Gshortint_list;
                  ANYVAR   Opcodes : Gshortint_list);
```

Semantics

Points is the number of vertices in the polygon set.

The **x** and **y coordinate arrays** contain the world coordinate values for each vertex of the polygon-set. The vertices must be in order. The vertices for the first sub-polygon must be at the beginning of these arrays, followed by the vertices for the second sub-polygon, etc. So, the coordinate arrays must contain a total number of vertices that equals points.

The **operation selector array** contains a series of integer operation selectors defining which vertices start new polygons, and defining which edges should be displayed.

Value	Meaning
0	Don't display the line for the edge extending to this vertex from the previous vertex.
1	Display the line for the edge extending to this vertex from the previous vertex.
2	This vertex is the first vertex of a sub-polygon. Succeeding vertices are part of a sub-polygon until a new start-of-polygon operation selector (2) is encountered. (Or the end of the arrays is encountered.)

Note

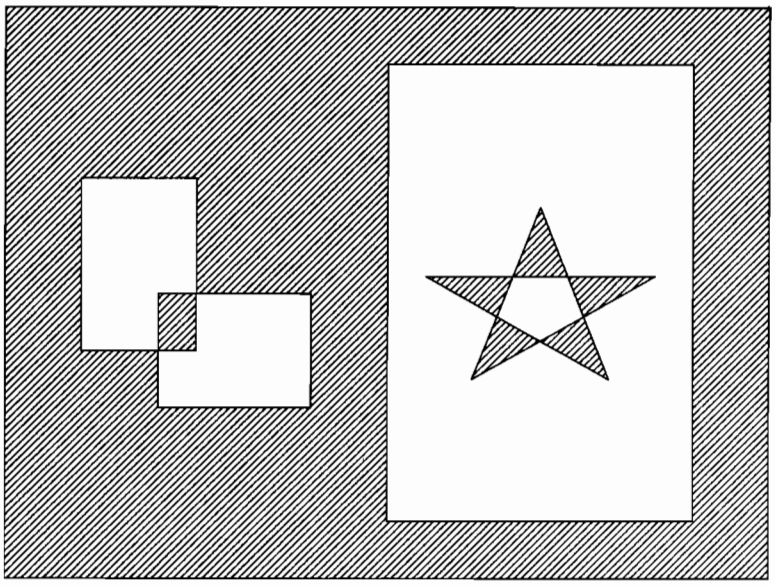
The first entry in the operation selector array **must** be 2, since it is the first vertex of a sub-polygon.

INT_POLYGON is used to output a polygon-set, specified in world coordinates, adhering exactly to the polygon style attributes that are currently specified. A polygon-set is a set of polygons (called "sub-polygons") that are treated graphically as one polygon. This is accomplished by "stacking" the sub-polygons. The subpolygons in a polygon-set may intersect or overlap each other.

The edge of a sub-polygon is defined as the line sequence that connects its vertices in the order specified. If the last vertex specified for a sub-polygon is not the same as the first, they are automatically connected.

When a polygon-set is displayed, the primitive attributes for polygons and lines define its appearance. In particular, the interior of the polygon-set will be filled according to the attributes of polygon style, polygon interior color and polygon interior line-style. If the edges are to be displayed as specified in the polygon style, the edges will adhere to the current line attributes of color, line-style and line-width. A dot will disappear on an edged polygon if the edge is done with a complementing line.

The filling of polygons also depends on how the sub-polygons "nest" within each other. An "even-odd" rule is used for determining which areas will be filled. Moving across the screen, count the edges of the polygon. Odd-numbered edges will turn the fill on and even-numbered edges will turn the fill off. The picture below will help clear up how the fills work.



Polygon Filling

Refer to SET_PGN_TABLE, SET_PGN_STYLE, SET_PGN_COLOR, SET_PGN_LS for a more detailed description of how attributes affect polygons.

As stated above, the values in the operation selector array define how the edges of the sub-polygons are displayed. The edge from the (I-1)th vertex to the Ith vertex will only be displayed if the Ith entry in the operation selector array equals 1. To display the edge from the last vertex to the first vertex of a sub-polygon, the first vertex must be explicitly respecified after all the other vertices of the sub-polygon, with an operation selector equal to 1. Otherwise the edge from the last vertex to the first will not be drawn. It will, however, automatically be connected for polygon filling.

If it is within the capabilities of the device, filling of the sub-polygon will be done to the sub-polygon edges regardless of whether the edges are displayed. If an entry in the operation selector array does not equal 0, 1, or 2, it will be treated as if it were equal to 0 and the edge will not be drawn.

When INT_POLYGON is used, the current position is updated to the end of the last sub-polygon specified in the polygon-set. The end of the last sub-polygon is defined to be the first (implicit last) vertex of the subpolygon. So, if there is only one vertex in a polygon-set this call degenerates to an update of the current position to the first coordinate set in the x and y point arrays (x coordinate array[1], y coordinate array[1]).

It is the application program's responsibility to ensure that the arrays are all dimensioned to at least the number of elements specified by points and that at least that many values are contained in each array.

Polygons are defined to be closed surfaces. When a sub-polygon extends beyond a clipping edge the closed nature of the sub-polygon is destroyed. As with other primitives, unpredictable results may occur if the sub-polygon extends beyond the clipping window.

This procedure is the same as the POLYGON procedure, with the exception that the parameters are of type *Gshortint* ($-32\,768..32\,767$). When used with some displays this procedure may perform about 3 times faster than the POLYGON procedure. For all other displays this procedure has about the same performance as the POLYGON procedure.

The INT_POLYGON procedure only has increased performance when the following conditions exist:

- The display must be a raster device.
- The window bounds are within the range $-32\,768$ through $32\,767$.
- The window must be less than $32\,767$ units wide and high.

INT_POLYGON is provided for efficient vector generation. Although its use can be mixed with MOVE, LINE, POLYLINE, and POLYGON, one dot roundoff errors may result with mixed use since different algorithms are used to implement each.

Error Conditions

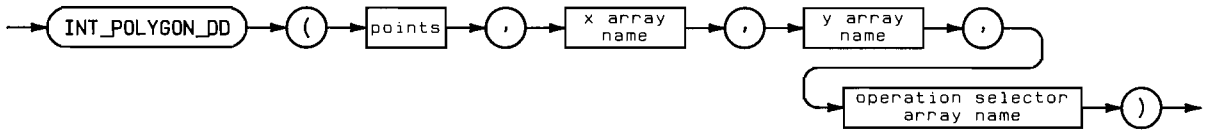
The graphics system must be initialized, a graphics display must be enabled, all parameters must be within specified limits and the number of points specified must be greater than 0 or the call will be ignored, an ESCAPE (-27) will be generated, and GRAPHICSERROR will return a non-zero value.

INT_POLYGON_DD

IMPORT: dgl_types
dgl_lib
dgl_poly

This **procedure** displays a polygon-set starting and ending at the specified point adhering to the specified polygon style in a device-dependent fashion.

Syntax



Item	Description/Default	Range Restrictions
points	Expression of TYPE INTEGER	MININT thru MAXINT
x array name	Array of TYPE *Gshortint_list*. *Gshortint* is a subrange of INTEGER.	−32 768 to 32 767
y array name	Array of TYPE *Gshortint_list*. *Gshortint* is a subrange of INTEGER.	−32 768 to 32 767
operation selector array name	Array of TYPE *Gshortint_list*. *Gshortint* is a subrange of INTEGER.	−32 768 to 32 767

Procedure Heading

```
PROCEDURE INT_POLYGON_DD ( Npoint    INTEGER;
                           ANYVAR   Xvec    : Gshortint_list;
                           ANYVAR   Yvec    : Gshortint_list;
                           ANYVAR   Opcodes : Gint_list      );
```

Semantics

Points is the number of vertices in the polygon set.

The **x** and **y coordinate arrays** contain the world coordinate values for each vertex of the polygon-set. The vertices must be in order. The vertices for the first sub-polygon must be at the beginning of these arrays, followed by the vertices for the second sub-polygon, etc. So, the coordinate arrays must contain a total number of vertices that equals points.

The **operation selector array** contains a series of integer operation selectors defining which vertices start new polygons, and defining which edges should be displayed.

Value	Meaning
0	Don't display the line for the edge extending to this vertex from the previous vertex.
1	Display the line for the edge extending to this vertex from the previous vertex.
2	This vertex is the first vertex of a sub-polygon. Succeeding vertices are part of a sub-polygon until a new start-of-polygon operation selector (2) is encountered. (Or the end of the arrays is encountered.)

Note

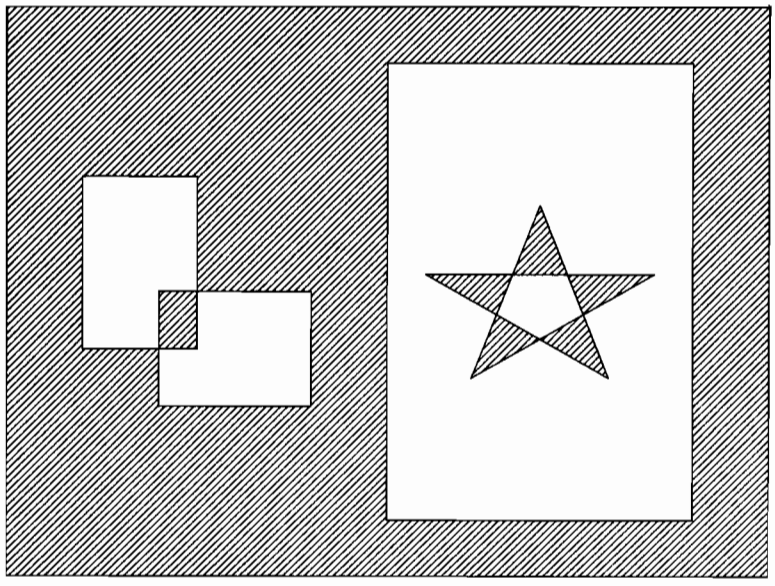
The first entry in the operation selector array **must** be 2, since it is the first vertex of a sub-polygon.

INT_POLYGON_DD is used to output a polygon-set, specified in world coordinates, adhering within the capabilities of the device to the polygon style attributes that are currently specified. A polygon-set is a set of polygons (called "sub-polygons") that are treated graphically as one polygon. The subpolygons in a polygon-set may intersect or overlap each other.

The edge of a sub-polygon is defined as the line sequence that connects its vertices in the order specified. If the last vertex specified for a sub-polygon is not the same as the first, they are automatically connected.

When a polygon-set is displayed, the primitive attributes for polygons and lines define its appearance. In particular, the interior of the polygon-set will be filled according to the attributes of polygon style, polygon interior color and polygon interior line-style. If the edges are to be displayed as specified in the polygon style, the edges will adhere to the current line attributes of color, line-style and line-width.

The filling of polygons also depends on how the sub-polygons "nest" within each other. An "even-odd" rule is used for determining which areas will be filled. Moving across the screen, count the edges of the polygon. Odd-numbered edges will turn the fill on and even-numbered edges will turn the fill off. The picture below will help clear up how the fills work.



Polygon Filling

Refer to SET_PGN_TABLE, SET_PGN_STYLE, SET_PGN_COLOR, SET_PGN_LS for a more detailed description of how attributes affect polygons.

As stated above, the values in the operation selector array define how the edges of the sub-polygons are displayed. The edge from the (I-1)th vertex to the Ith vertex will only be displayed if the Ith entry in the operation selector array equals 1. To display the edge from the last vertex to the first vertex of a sub-polygon, the first vertex must be explicitly respecified after all the other vertices of the sub-polygon, with an operation selector equal to 1. Otherwise the edge from the last vertex to the first will not be drawn. It will, however, automatically be connected for polygon filling.

If it is within the capabilities of the device, filling of the sub-polygon will be done to the sub-polygon edges regardless of whether the edges are displayed. If an entry in the operation selector array does not equal 0, 1, or 2, it will be treated as if it were equal to 0, i.e., the edge will not be drawn.

When INT_POLYGON_DD is used, the current position is updated to the end of the last sub-polygon specified in the polygon-set. The end of the last sub-polygon is defined to be the first (implicit last) vertex of the subpolygon. So, if there is only one vertex in a polygon-set this call degenerates to an update of the current position to the first coordinate set in the x and y point arrays (x coordinate array[1], y coordinate array[1]).

It is the application program's responsibility to ensure that the arrays are all dimensioned to at least the number of elements specified by points and that at least that many values are contained in each array.

Device capabilities vary widely. Not all devices are able to draw polygon edges as requested. If a device is not able to draw polygon edges as requested, they will be simulated in software. The simulation will always adhere to the edge value in SET_PGN_STYLE and the operation selector in INT_POLYGON_DD, but the line-style and color of the edge will depend on the capability of the device to produce lines with those attributes.

Polygon fill capabilities can vary widely between devices. A device may have no filling capabilities at all, may be able to perform only solid fill, or may be able to fill polygons with different fill densities and at different fill line orientations. INT_POLYGON_DD tries to match the device capabilities to the request. If the device cannot fill the request at all, then no simulation is done and the polygon will not be filled. For HPGL plotters, the fill is simulated. For raster devices, if the density is greater than 0.5, a solid fill is used, otherwise, the fill is simulated.

In the case where the polygon style specifies non-display of edged, this would result in no visible output although visible output had been specified. To provide some visible output in this case, INT_POLYGON_DD will outline the polygon using the color and line-style specified for the fill lines. However, only those edge segments specified as displayable by the operation selector array will be drawn. Therefore, if all edge segments are specified as non-displayed, there will still be no visible output.

Regardless of the capabilities of the device, INT_POLYGON_DD sets the starting position to the first vertex of the last member polygon specified in the call. If there is only one polygon specified, the starting position will therefore be set to the first vertex specified.

Polygons are defined to be closed surfaces. When a sub-polygon extends beyond a clipping edge the closed nature of the sub-polygon is destroyed. As with other primitives, unpredictable results may occur if the sub-polygon extends beyond the clipping window.

This procedure is the same as the procedure POLYGON_DEV_DEP, with the exception that the parameters are of type *Gshortint* (−32 768..32 767). When used with some displays this procedure may perform about 3 times faster than the POLYGON_DEV_DEP procedure. For all other displays this procedure has about the same performance as the POLYGON_DEV_DEP procedure.

The INT_POLYGON_DD procedure only has increased performance when the following conditions exist:

- The display is a raster device.
- The window bounds are within the range −32 768 through 32 767.
- The window is less then 32 767 units wide and high.

INT_POLYGON_DD is provided for efficient vector generation. Although its use can be mixed with MOVE, LINE, POLYLINE, and POLYGON_DEV_DEP, one dot roundoff errors may result with mixed use since different algorithms are used to implement each.

Error Conditions

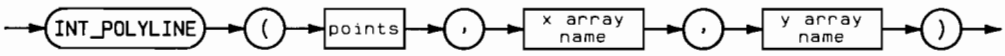
The graphics system must be initialized, a graphics display must be enabled, all parameters must be within specified limits and the number of points (Points) must be greater than 0 or the call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

INT_POLYLINE

IMPORT: dgl_types
dgl_lib

This **procedure** draws a connected line sequence starting at the specified point.

Syntax



Item	Description/Default	Range Restrictions
points	Expression of TYPE INTEGER	MININT thru MAXINT
x array name	Array of TYPE *Gshortint_list*. *Gshortint* is a sub-range of INTEGER.	–32 768 to 32 767
y array name	Array of TYPE *Gshortint_list*. *Gshortint* is a sub-range of INTEGER.	–32 768 to 32 767

Procedure Heading

```
PROCEDURE INT_POLYLINE (        Npts          : INTEGER;
                         ANYVAR Xvec, Yvec : Gshortint_list )
```

Semantics

Points is the number of vertices in the polygon set.

The **x** and **y coordinate arrays** contain the world coordinate values for each vertex of the polyline-set. The vertices must be in order. The vertices for the first sub-polyline must be at the beginning of these arrays, followed by the vertices for the second sub-polyline, etc. So, the coordinate arrays must contain a total number of vertices that equals points.

The procedure INT_POLYLINE provides the capability to draw a series of connected lines starting at the specified point. A complete object can be drawn by making one call to this procedure. This call first sets the starting position to be the first elements in the x and y coordinate arrays. The line sequence begins at this point and is drawn to the second element in each array, then to the third and continues until points-1 lines are drawn.

This procedure is equivalent to the following sequence of calls:

```
INT_MOVE (X_coordinate_array[1],Y_coordinate_array[1]);
INT_LINE (X_coordinate_array[2],Y_coordinate_array[2]);
INT_LINE (X_coordinate_array[3],Y_coordinate_array[3]);
              :
              :
INT_LINE (X_coordinate_array[Points],Y_coordinate_array[Points]);
```

The starting position is set to (X_coordinate_array[Points], Y_coordinate_array[Points]) at the completion of this call.

Specifying only one element, or Points equal to 1, causes a move to be made to the world coordinate point specified by the first entries in the two coordinate arrays.

It is the application program's responsibility to ensure that the arrays are all dimensioned to at least the number of elements specified by points and that at least that many values are contained in each array.

Depending on the nature of the current line-style nothing may appear on the graphics display. See SET_LINE_STYLE for a complete description of how line-style affects a particular point or vector.

The primitive attributes of color, line-style, and line-width apply to polylines.

This procedure is the same as the POLYLINE procedure, with the exception that the parameters are of type *Gshortint* (– 32 768..32 767). When used with some displays this procedure may perform about 3 times faster than the POLYLINE procedure. For all other displays this procedure has about the same performance as the POLYLINE procedure.

The INT_POLYLINE procedure only has increased performance when the following conditions exist:

- The display must be a raster device.
- The window bounds within the range – 32 768 to 32 767.
- The window must be less then 32 767 units wide and high.

INT_POLYLINE is provided for efficient vector generation. Although its use can be mixed with MOVE, LINE, and POLYLINE, one dot roundoff errors may result with mixed use since different algorithms are used to implement each.

Error Conditions

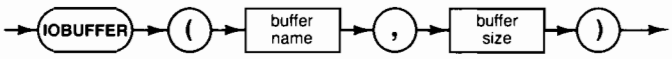
The graphics system must be initialized, a graphics display must be enabled, all parameters must be within specified limits and the number of points (points) must be greater than 0 or the call will be ignored, an ESCAPE (– 27) will be generated, and GRAPHICSERROR will return a non-zero value.

IOBUFFER

IMPORT: general_4
iodeclarations

This **procedure** will create a buffer area of the specified number of bytes. The buffer name variable contains the various empty and fill pointers necessary to use the buffer space.

Syntax



Item	Description/Default	Range Restrictions
buffer name	Variable of TYPE *buf_info_type*.	See Chapter 11
buffer size	Expression of TYPE INTEGER, specifies bytes.	MININT thru MAXINT

Semantics

Re-executing IOBUFFER on a buffer name will allocate new space in the system, not reclaim the old space, or put a transfer in the old space into a known state.

MARK and RELEASE interact with IOBUFFER, and it is possible to lose an io buffer by releasing it.

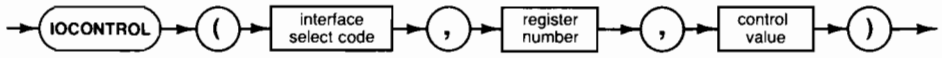
The buffer name should be in a VAR declaration at the outermost level of the program or module containing it.

IOCONTROL

IMPORT: general_0
iodeclarations

This **procedure** sends control information to the selected interface. Refer to the specific interface in the Status and Control Register Appendix in the Pascal System User's Manual.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
register number	Expression of TYPE *io_word*. This is an INTEGER subrange.	−32 768 thru 32 767	Interface dependent
control value	Expression of TYPE INTEGER.	MININT thru MAXINT	0 thru 65 535 (interface dependent)

Note

Unexpected and possibly undesirable side effects may result from attempting to use this procedure in combination with other parts of the I/O procedure library. Make sure you understand the full implications of using it before including it in a program.

IOERROR_MESSAGE

IMPORT: general_3
iodeclarations

This **function** returns a value of TYPE *iostring* (a string dimensioned to 255 characters) containing an English textual description of an error produced by the I/O procedure library.

Syntax

Item	Description/Default	Range Restrictions	Recommended Range
error number	Expression of TYPE INTEGER.	MININT thru MAXINT	0 thru 327

Semantics

Example:

```
PROGRAM Sample(Input, Output);
  .
  .
  .
BEGIN
   TRY
     .
     .
     .
   RECOVER BEGIN
      IF Escapecode = Ioescapecode THEN
         WRITELN (IOERROR_MESSAGE(Ioe_result),' on ',Ioe_isc);
         ESCAPE (Escapecode);
   END {Recover}
END. {Main Program}
```

IO_FIND_ISC

IMPORT: iodeclarations

Note

This function is provided for use by the internal I/O Procedure Library drivers, only. Unexpected and possible undesirable results may occur if it is used.

IO_ESCAPE

IMPORT: iodeclarations

Note

This function is provided for use by the internal I/O Procedure Library drivers, only. Unexpected and possible undesirable results may occur if it is used.

IOINITIALIZE

IMPORT: general_1

This **procedure** resets all interfaces.

Syntax

Semantics

A program should be bracketed by IOINITIALIZE and IOUNINITIALIZE.

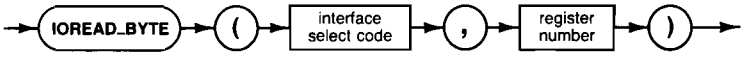
```
PROGRAM userprog ( ..... ) ;
   .
   .
BEGIN
  ioinitialize;
   .
   .
   .
  iouninitialize;
END.
```

IOREAD_BYTE

IMPORT: general_0
iodeclarations

This **function** reads the byte contained in specified register (physical address) on the selected interface. The function returns a value of TYPE *io_byte*. This is an INTEGER subrange, 0..255.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
register number	Expression of TYPE *io_word*. This is an INTEGER subrange.	–32 768 thru 32 767	Interface dependent

Semantics

Note

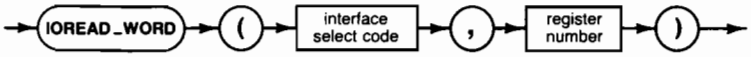
These are physical address registers, **not** the Status registers used by the IOSTATUS statement. See the Memory Map Appendix in the Pascal System Users Manual.

IOREAD_WORD

IMPORT: general_0
iodeclarations

This **function** reads the word contained in the specified register (physical address) on the selected interface. The function returns a value of TYPE *io_word*. This is an INTEGER subrange, −32 768..32 767.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
register number	Expression of TYPE *io_word*. This is an INTEGER subrange.	−32 768 thru 32 767	Interface dependent

Semantics

Note

These are physical address registers, **not** the Status registers used by the IOSTATUS statement. See the Memory Map Appendix in the Pascal System Users Manual.

IORESET

IMPORT: general_1
iodeclarations

This **procedure** will reset the specified interface to its intial (power on) state. Any currently active transfers will be terminated.

Syntax

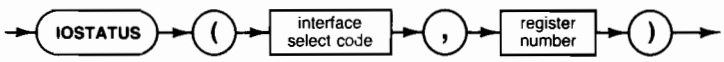
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

IOSTATUS

IMPORT: general_0
iodeclarations

This **function** returns the contents of an interface status register. The value returned is of TYPE *io_word*, an integer subrange (– 32 768 thru 32 767).

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru31	7 thru 31
register number	Expression of TYPE *io_word*. This is an INTEGER subrange.	– 32 768 thru 32 767	Interface dependent

Semantics

The register meaning depends on the interface. Refer to the specific interface in the Status and Control Register Appendix in the Pascal System Users Manual.

IO_SYSTEM_RESET

IMPORT: general_0
iodeclarations

Note

This function is provided for use by the internal I/O Procedure Library drivers, only. Unexpected and possible undesirable results may occur if it is used.

IOUNINITIALIZE

IMPORT: general_1
iodeclarations

This **procedure** resets all interfaces.

Syntax

Semantics

A program should be bracketed by IOINITIALIZE and IOUNINITIALIZE.

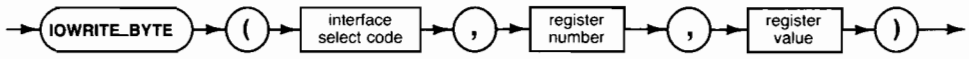
```
PROGRAM userprog ( ..... ) ;
     .
     .
BEGIN
  ioinitialize;
     .
     .

  iouninitialize;
END.
```

IOWRITE_BYTE

IMPORT: general_0
iodeclarations

This **procedure** writes the supplied value (representing one byte) to the specified register (physical address) on the selected interface. The actual action resulting from the operation depends on the interface and register selected.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
register number	Expression of TYPE *io_word*. This is an INTEGER subrange.	−32 768 thru 32 767	Interface dependent
register value	Expression of TYPE *io_byte*. This is an INTEGER subrange.	0 thru 255	Interface dependent

Semantics

Notes

These are physical address registers, **not** the Status registers used by the IOSTATUS statement. See the Memory Map Appendix in the Pascal System Users Manual.

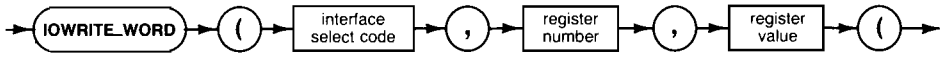
Unexpected and possibly undesirable side effects may result from attempting to use this procedure in combination with other parts of the I/O procedure library. Make sure you understand the full implications of using it before including it in a program.

IOWRITE_WORD

IMPORT: general_0
iodeclarations

This **procedure** writes the supplied value (representing 16 bits) to the specified register on the selected interface. The actual action resulting from the operation depends on the interface and register selected.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
register number	Expression of TYPE *io_word*. This is an INTEGER subrange.	−32 768 thru 32 767	Interface dependent
register value	Expression of TYPE *io_word*. This is an INTEGER subrange.	−32 768 thru 32 767	Interface dependent

Semantics

Notes

These are physical address registers, **not** the Status registers used by the IOSTATUS statement. See the Memory Map Appendix in the Pascal System Users Manual.

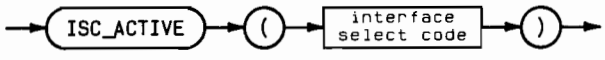
Unexpected and possibly undesirable side effects may result from attempting to use this procedure in combination with other parts of the I/O procedure library. Make sure you understand the full implications of using it before including it in a program.

ISC_ACTIVE

IMPORT: general_4
iodeclarations

This BOOLEAN function is TRUE if there is a transfer active on the specified interface.

Syntax



Item	Description/Default	Range Restrictions
interface select code	Expression of TYPE type_isc. This is an INTEGER subrange	7 thru 31

KERNEL_INITIALIZE

IMPORT: general_0

Note

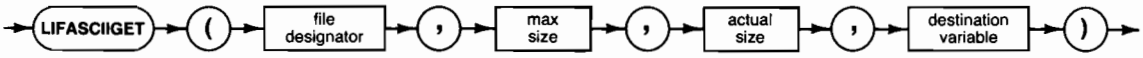
This function is provided for use by the internal I/O Procedure Library drivers, only. Unexpected and possible undesirable results may occur if it is used. It will probably blow up your program, and will definitely destroy any operation you are currently performing in the I/O Procedure Library.

LIFASCIIGET

IMPORT: liflib

This INTEGER **function** sequentially reads LIF ASCII or LIF BINARY file records.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
file designator	Variable of TYPE *liffile*.	See Glossary	–
max size	Expression of type INTEGER.	MININT thru MAXINT	1 thru 32767
actual size	Variable of type INTEGER.	–	–
destination variable	Variable of any type.	–	–

Semantics

Each call to the function will read at most one logical record from the designated file, even if the buffer could contain more data. Zero length records will cause the actual size to be set to zero.

The file designator is used to point to the information record. The max size is the maximum number of bytes to read. The actual size variable will contain the number of bytes actually read into the destination variable by the function call.

The destination variable will contain the information read from the designated file after the function call. This function approximates the operation performed by READ on Pascal files.

Returned Value

The value returned by LIFASCIIGET is an error code.

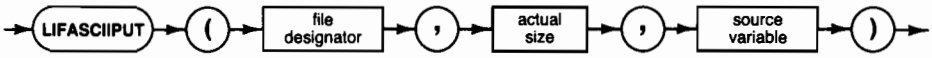
0 No error.
1 File is not open for reading or file type is not ASCII or BINARY.
2 File is not open.
3 File is at End Of File (actual size = 0).
4 Max size <= 0.
5 End-of-file encountered while reading the record. (The last record in the file was malformed.) Actual size will contain the count of actual bytes read.

LIFASCIIPUT

IMPORT: liflib

This INTEGER **function** sequentially writes LIF ASCII or LIF BINARY file records.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
file designator	Variable of TYPE *liffile*.	See Glossary	–
actual size	Expression of type INTEGER.	MININT thru MAXINT	0 thru 32767
source variable	Variable of any type.	–	–

Semantics

A record must contain 0 thru 32767 bytes.

The file designator is used to point to the information record.

The actual size specifies the number of bytes to be written to the designated file.

The source variable contains the information to be written to the designated file.

This function approximates the operation performed by WRITELN or Pascal files.

Returned Value

The value returned by LIFASCIIPUT is an error code:

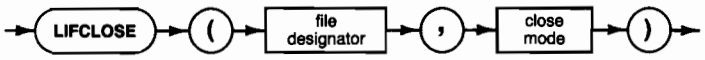
0 No error.
1 File is not open for writing or file type is not ASCII or BINARY.
2 File is not open.
3 File is at End Of File (no data written).
4 The actual size is less than 0 or greater than 32767.

LIFCLOSE

IMPORT: liflib

This INTEGER **function** performs the final operations on a file, and removes the file block from the heap if it was created by LIFOPEN.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
file designator	Variable of TYPE *liffile*.	See Glossary	–
close mode	Expression of enumerated TYPE *lifclosemode*.	LIFKEEP LIFREMOVE LIFMINSIZE	

Semantics

The file designator is used to point to the information record. The close mode determines the housekeeping to be performed on the LIF directory:

LIFKEEP	Close the file and retain the LIF directory entry.
LIFREMOVE	Close the file and remove the LIF directory entry.
LIFMINSIZE	For LIFW and LIFU, this closes the file and sets the allocated size to cover the last sector written. For LIFR, this is the same as LIFKEEP. The directory entries retained in all three cases.

This function approximates the operation performed by CLOSE on Pascal files.

Returned Value

The value returned by LIFCLOSE is an error code:

0 No error.
1 File is not open.
2 Unit number is not a workstation volume, or does not have a LIF header.
3 File is not found.

Note

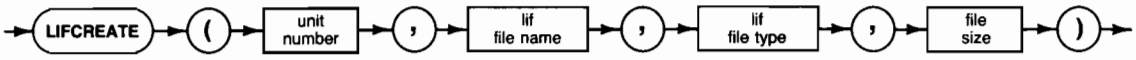
Errors 2 and 3 can occur only if the close mode is LIFREMOVE.

LIFCREATE

IMPORT: liflib

This INTEGER **function** creates a directory entry for the file described.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
unit number	Expression of TYPE INTEGER.	MININT thru MAXINT	3 thru 50, but not 6
lif file name	Expression of TYPE *lifname.*	See Glossary	–
lif file type	Expression of TYPE INTEGER.	MININT thru MAXINT	1, −2
file size	Expression of TYPE INTEGER, represents sectors.	MININT thru MAXINT	

Semantics

The unit number is the workstation file unit which contains the LIF volume.

The lif file name is the name of the file in the LIF directory. It must not exist before LIFCREATE is used.

The lif file type is the numeric code used in the LIF directory structure for identifying various types of files.

1	LIF ASCII
−2	LIF BINARY

Other file types are possible on the LIF directory, with any number in the range −32768 thru 32767 acceptable, excluding the values 0 and −1.

The file size determines the number of sectors to be reserved for the file. If the file size is less than or equal to 0, the maximum number of sectors available on the media is allocated.

Return Value

The value returned by LIFCREATE is an error code:

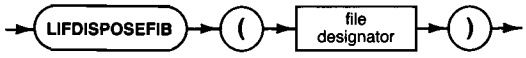
0 No error.
1 Unit Number is not a workstation volume, or volume does not have a LIF header.
2 The lif file type is invalid (0 or −1).
3 The file already exists.
4 Not enough room on the volume to create the file.

LIFDISPOSEFIB

IMPORT: liflib

This BOOLEAN **function** de-allocates a file information block from the heap and clears the file designator pointer.

Syntax



Item	Description/Default	Range Restrictions
file designator	Variable of TYPE *liffile*.	See Glossary

Semantics

The file designator is used to point to the information record.

This function is used after closing a file (using LIFCLOSE) which was created (using LIFOPEN) with a file block mode equal to LIFUSEFIB. This is done when MARK and RELEASE are being used for heap management. File blocks allocated with LIFNEWFIB are disposed of with LIFDISPOSEFIB.

Return Value

The value returned by LIFDISPOSEFIB is a BOOLEAN:

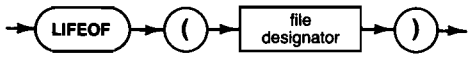
TRUE	The file block was disposed of.
FALSE	The file was still opened, and the file block was not disposed of.

LIFEOF

IMPORT: liflib

This BOOLEAN **function** is TRUE if the designated file is at end-of-file, or if the file is closed.

Syntax



Item	Description/Default	Range Restrictions
file designator	Expression of TYPE *liffile*.	See Glossary

Semantics

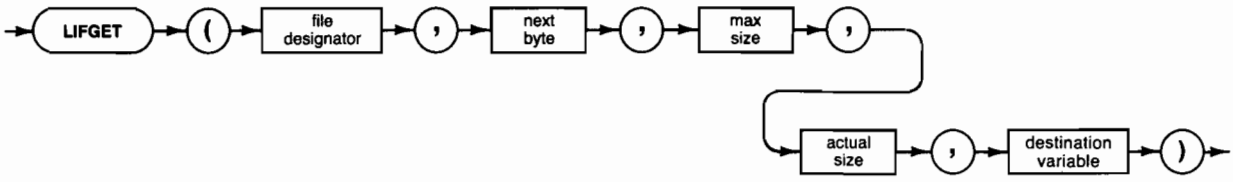
The file designator is used to point to the information record.

LIFGET

IMPORT: liflib

This INTEGER **function** reads data from files of types other than ASCII and BINARY.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
file designator	Variable of TYPE *liffile*.	See Glossary	–
next byte	Variable of TYPE INTEGER.	MININT thru MAXINT	0 thru MAXINT
max size	Expression of TYPE INTEGER.	MININT thru MAXINT	1 thru MAXINT
actual size	Variable of TYPE INTEGER.	–	–
destination variable	Variable of any type.	–	–

Semantics

The file designator is used to point to the information record.

Next byte is the index to the first byte to be read by LIFGET. It is automatically updated to point to the first byte after the last byte transferred from the file to the buffer. The first byte in a file is byte 0.

The max size is the maximum number of bytes to read.

The actual size will contain the number of bytes actually read into the return variable by the function call. Actual size and max size will differ only if end-of-file is encountered during the LIFGET operation.

The destination variable will contain the information read from the designated file after the function call.

Returned Value
The value returned by LIFGET is an error code:

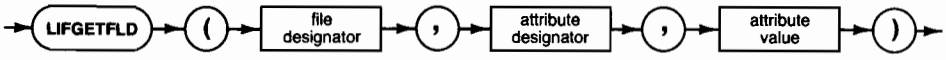
0 No error.
1 File is not open for reading or file type is ASCII or BINARY.
2 File is not open.
3 File is at End Of File (actual size contains the number of bytes read).
4 Max size <= 0, or next byte < 0.

LIFGETFLD

IMPORT: liflib

This INTEGER **function** reads some of the attribute fields of a LIF file.

Syntax



Item	Description/Default	Range Restrictions
file designator	Expression of TYPE *liffile*.	See Glossary
attribute designator	Expression of enumerated TYPE *liffieldname*.	LIFEXTENSION LIFVOLNUMBER LIFLASTVOLFLAG LIFFSIZE LIFRSIZE
attribute value	Variable of TYPE INTEGER.	–

Semantics

The file designator is used to point to the information record.

The attribute designator determines which of the file attributes is to be read:

LIFEXTENSION	File extension field (last 32 bits of the file directory entry).
LIFVOLNUMBER	File volume number (1 thru 32767)
LIFLASTVOLFLAG	1 = > last volume of file 0 = > more volumes in the file
LIFFSIZE	The number of sectors allocated for the file.
LIFRSIZE	For reading ASCII and BINARY files, the number of bytes in the record about to be read. This will be the bytes remaining in the file if the last LIFASCIIGET executed only read part of the record. For LIFASCIIPUT and LIFPUT, this field represents the number of bytes actually written to the file.

The attribute value will contain the value of the designated attribute field after the function is called.

Returned Value

The value returned by LIFGETFIELD is an error code:

0 No error.

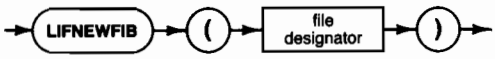
1 File is not open.

LIFNEWFIB

IMPORT: liflib

This **procedure** allocates a file information block from the heap, initializes it, and sets the file designator pointer.

Syntax



Item	Description/Default	Range Restrictions
file designator	Variable of TYPE *liffile*.	See Glossary

Semantics

The file designator is used to point to the information record.

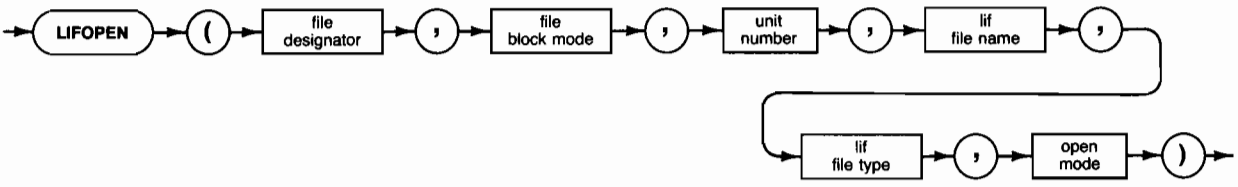
This procedure is used prior to opening a file with a file block mode equal to LIFUSEFIB. This is done when MARK and RELEASE are being used for heap management. File blocks allocated with LIFNEWFIB are disposed of with LIFDISPOSEFIB.

LIFOPEN

IMPORT: liflib

This INTEGER **function** initializes a file information block for use by file access functions.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
file designator	Variable of TYPE *liffile*.	See Glossary	–
file block mode	Expression of enumerated TYPE *liffibop*.	LIFGETFIB, LIFUSEFIB	
unit number	Expression of TYPE INTEGER.	MININT thru MAXINT	3 thru 50, but not 6
lif file name	Expression of TYPE *lifname*.	See Glossary	–
lif file type	Expression of TYPE INTEGER.	MININT thru MAXINT	1, –2
open mode	Expression of enumerated TYPE *lifopenmode*.	LIFR, LIFW LIFU	

Semantics

The file designator is used to point to the information record.

The file block mode determines the heap discipline that LIFOPEN is to assume is being used. If your program is using NEW and DISPOSE, use LIFGETFIB. If your program uses MARK and RELEASE, use LIFUSEFIB. If you are using LIFUSEFIB, you must create a file block on the HEAP with LIFNEWFIB before trying to open the file with LIFOPEN, and once you have finished with the file, you must dispose of the file block yourself with LIFDISPOSEFIB.

The unit number is the workstation unit which contains the LIF volume.

The lif file name is the name of the file in the LIF directory. It must already exist before LIFOPEN is used.

The lif file type is the numeric code used in the LIF directory structure for identifying various types of files.

1	LIF ASCII
−2	LIF BINARY

Other file types are possible on the LIF directory, with any number in the range −32768 thru 32767 acceptable, excluding the values 0 and −1.

The open mode determines the mode in which the file is to be accessed:

LIFR	Read only
LIFW	Write only
LIFU	Update mode (not valid for ASCII or BINARY)

Return Value
The value returned by LIFOPEN is an error code:

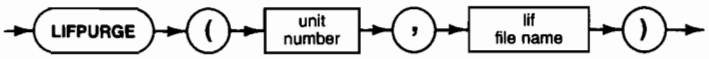
0 No error.
1 Unit Number is not a workstation volume, or volume does not have a LIF header.
2 The file is already open.
3 The LIF file type is invalid (0 or −1) or the open mode is invalid (update mode for ASCII or BINARY).
4 The LIF file name is not in the directory.
5 The LIF file name is in the directory, but is not of the LIF file type specified.

LIFPURGE

IMPORT: liflib

This INTEGER **function** removes the LIF directory entry for the named file.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
unit number	Expression of TYPE INTEGER.	MININT thru MAXINT	3 thru 50, but not 6
lif file name	Expression of TYPE *lifname*.	See Glossary	–

Semantics

The unit number is the workstation file unit which contains the LIF volume.

The LIF file name is the name of the file in the LIF directory. It must exist before LIFPURGE is used.

Return Value

The value returned by LIFPURGE is an error code:

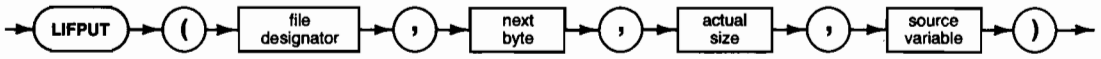
0 No error.
1 Unit Number is not a workstation volume, or volume does not have a LIF header.
2 The file name is not in the directory.

LIFPUT

IMPORT: liflib

This INTEGER **function** writes data to files of types other than ASCII and BINARY.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
file designator	Variable of TYPE *liffile*.	See Glossary	–
next byte	Variable of TYPE INTEGER.	MININT thru MAXINT	0 thru MAXINT
actual size	Expression of TYPE INTEGER.	MININT thru MAXINT	0 thru MAXINT
source variable	Variable of any type.	–	–

Semantics

The file designator is used to point to the information record. The next byte is the index to the first byte to be written by LIFPUT. It is automatically updated to point to the first byte after the last byte transferred from the buffer to the file. The first byte in a file is byte 0.

The actual size specifies the number of bytes to be written into the designated file by the function call.

The source variable should contain the data to be written to the file before LIFPUT is called.

Returned Value

The value returned by LIFPUT is an error code:

0 No error.

1 File is not open for writing or file type is ASCII or BINARY.

2 File is not open.

3 File is at End Of File (no data actually transferred).

4 Actual size <0 , or next byte < 0.

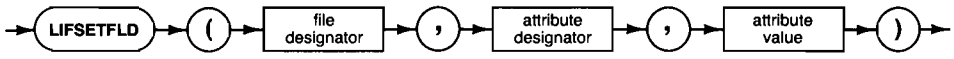
5 File at end-of-file (some data transferred - see LIFRSIZE under the function LIFGET-FIELD).

LIFSETFLD

IMPORT: liflib

This INTEGER **function** modifies some of the attribute fields of a LIF file.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
file designator	Expression of TYPE *liffile*.	See Glossary	–
attribute designator	Expression of enumerated TYPE *liffieldname*.	LIFEXTENSION LIFVOLNUMBER LIFLASTVOLFLAG LIFFSIZE LIFRSIZE	any but LIFFSIZE or LIFRSIZE
attribute value	Expression of TYPE INTEGER.	MININT thru MAXINT	attribute dependent

Semantics

The file designator is used to point to the information record.

The attribute designator determines which of the file attributes is to be modified:

Attribute Designator	Attribute Field Description	Attribute Value Range
LIFEXTENSION	File extension field (last 32 bits of the file directory entry)	MAXINT thru MININT
LIFVOLNUMBER	File volume number	1 thru 32767
LIFLASTVOLFLAG	Not equal to 0 => last volume of file Equal to 0 => more volumes in the file.	
LIFFSIZE	Read only field.	
LIFRSIZE	Read only field.	

The attribute value contains the value to be written to the designated attribute field.

Returned Value
The value returned by LIFSETFIELD is an error code:

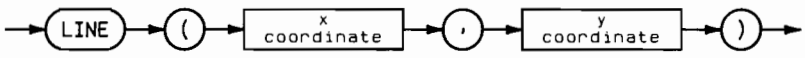
0 No error.
1 File is not open.
2 Attribute value is out of range.
3 Read only field, or file is open for read only.

LINE

IMPORT: dgl_lib

This **procedure** draws a line from the starting position to the world coordinate specified.

Syntax



Item	Description/Default	Range Restrictions
x coordinate	Expression of TYPE REAL	–
x coordinate	Expression of TYPE REAL	–

Procedure Heading

```
PROCEDURE LINE ( Wx, Wy : REAL );
```

Semantics

A line is drawn from the starting position to the world coordinate specified by the X and Y coordinates. The starting position is updated to this point at the completion of this call.

The **x** and **y coordinate** pair is the ending of the line to be drawn in the world coordinate system.

The primitive attributes of line style, line width, and color apply to lines drawn using LINE. Drawing to the starting position generates the shortest line possible. Depending on the nature of the current line-style, nothing may appear on the graphics display surface. See SET_LINE_STYLE for a complete description of how line-style affects a particular point or vector.

Error Conditions

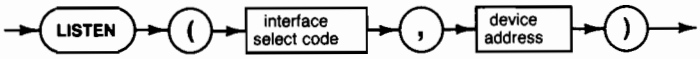
The graphics system must be initialized and a display must be enabled or this call will be ignored, an ESCAPE (–27) will be generated, and GRAPHICSERROR will return a non-zero value.

LISTEN

IMPORT: hpib_2
iodeclarations

This **procedure** will send the specified listen address on the bus. The ATN line will be set true. The interface must be active controller.

Syntax



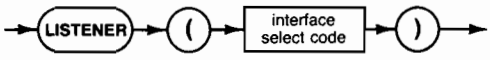
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
device address	Expression of TYPE *type_hpib_address*. This is an INTEGER subrange.	0 thru 31	0 thru 30

LISTENER

IMPORT: hpib_3
iodeclarations

This BOOLEAN **function** will return TRUE if the specified interface is currently addressed as a listener.

Syntax



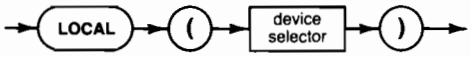
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

LOCAL

IMPORT: hpib_2
iodeclarations

This **procedure** places the device(s) in local mode.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary

Semantics

LOCAL (701) places the device at address 1 on interface 7 in the Local mode. LOCAL(7) places all devices on interface 7 in Local mode.

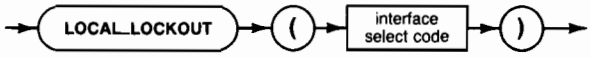
	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	$\overline{REN}$ $\overline{ATN}$	ATN MTA UNL LAG GTL	ATN GTL	ATN MTA UNL LAG GTL
Not Active Controller	$\overline{REN}$	Error	Error	

LOCAL_LOCKOUT

IMPORT: hpib_2
iodeclarations

This **procedure** sends LLO (the local lockout message) on the bus. The interface must be active controller.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

Semantics

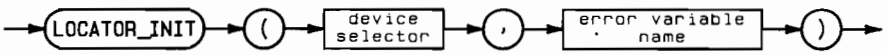
	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN LLO	Error	ATN LLO	Error
Not Active Controller	Error			

LOCATOR_INIT

IMPORT: dgl_lib

This **procedure** enables the locator device for input.

Syntax



Item	Description/Default	Range Restrictions
device selector	Expression of TYPE INTEGER	MININT TO MAXINT
error variable name	Variable of TYPE INTEGER	–

Procedure Heading

```
PROCEDURE LOCATOR_INIT (     Dev_Adr : INTEGER;
                         VAR Ierr    : INTEGER );
```

Semantics

The **device selector** specifies the physical addresses of the graphics locator device.

- device selector = 2 The Knob on Series 200 Computers
- 100 <= device selector <= 3199 composite HPIB/device address

The **error variable** will contain a value indicating whether the locator device was successfully enabled.

Value	Meaning
0	The locator device was successfully initialized.
2	Unrecognized device specified. Unable to communicate with a device at the specified address, non-existent interface card or non-graphics system supported interface card.

If the error variable contains a non-zero value, the call has been ignored.

LOCATOR_INIT enables the logical locator device for input. Enabling the locator includes associating the logical locator device with a physical device and initializing the device. The device name is set to the name of the physical device, the device status is set to 1 (enabled) and the internal device selector used by the graphics library is set equal to the device selector provided by the user. This information is available by calling INQ_WS with operation selectors 11052 and 13052.

LOCATOR_INIT implicitly makes the picture current before attempting to initialize the device.

LOCATOR_INIT enables the logical locator device for input. Enabling the locator includes associating the logical locator device with a physical device and initializing the device.

The graphics library attempts to directly identify the type of device by using its device address in some way. The meanings of the device address are defined above.

At the time that the graphics library is initialized, all devices which are to be used must be connected, powered on, ready, and accessible via the specified physical address. Invalid addressed or unresponsive devices result in that device not being initialized and an error being returned.

The locator device must be enabled before it is used for input. The locator device is disabled by calling LOCATOR_TERM.

If the graphics display and the locator are not the same physical device (e.g. HP 9826 display and HP 9111 locator), then the logical locator limits will be set to the default values for the particular locator used. If the graphics display and locator are the same physical device (e.g., HP 9826 display and HP 9826 knob locator), then the logical locator limits are set to the current view surface limits.

The locator echo position is set to the default value (see SET_ECHO_POS).

Only one locator device may be enabled at a time. If a locator is currently enabled, then the enabled device will be terminated (via LOCATOR_TERM) and the call will continue. The locator device should be disabled before the termination of the application program. LOCATOR_INIT is the complementary routine to LOCATOR_TERM.

HPGL Locator Devices

When the locator device is initialized on an HPGL device, the graphics display is left unaltered. HPGL devices are initialized to the following defaults when LOCATOR_INIT is executed:

Device	Wide mm	High mm	Wide points	High points	Aspect	Resolution points/mm
9872	400	285	16000	11400	.7125	40.0
7580	809.5	524.25	32380	20970	.6476	40.0
7585	1100	890	44000	35670	.809	40.0
7470	257.5	190	10300	7600	.7378	40.0
9111	300.8	217.6	12032	8704	.7234	40.0

The maximum physical limits of the locator for a HPGL device not listed above are determined by the default settings of P1 and P2. The default settings of P1 and P2 are the values they have after an HPGL 'IN' command. Refer to the specific device manual for additional details.

The default logical display surface is set equal to the area defined by P1 and P2 at the time LOCATOR_INIT is invoked.

Note

If the paper is changed in an HP 7580 or HP 7585 plotter while the graphics locator is initialized, it should be the same size of paper that was in the plotter when LOCATOR_INIT was called. If a different size of paper is required, the device should be terminated (LOCATOR_TERM) and re-initialized after the new paper has been placed in the plotter.

No locator points are returned while the pen control buttons are depressed on HPGL plotters.

The Knob as Locator

When the locator device is initialized, the graphics display is left unaltered. The default initialization characteristics for the knob on various Series 200 computers is listed below:

Computer	Wide mm	High mm	Wide points	High points	Aspect	Resolution mm
9816/ 9920	168	126	400	300	.75	2.381
9826	120	90	400	300	.75	3.333
9836	210	160	512	390	.7617	2.438
9836C	217	163	512	390	.7617	2.39

The knob uses the current display limits as its locator limits for locator echoes 2 though 8. For all other echoes the above limits are used. An example of when the two limits may differ follows:

> The knob locator is initialized on a HP 9826. The graphics display is an HP 98627A color output card. The resolution of the locator is 0 through 399 in the x dimension, and 0 through 299 in the y dimension. The resolution of the display is 0 through 511 in x dimension, and 0 through 389 in y dimension. When await_locator is used with echo 4, the locator will effectively have the HP 98627A resolution for the duration of the await_locator call. However if echo 1 is used with await_locator, the cursor will appear on the HP 9826 and the locator has a resolution of 0×399 and 0×299. Note that all conversion routines, and inquiries will used the HP 9826 limits.

The physical origin of the locator device is the lower left corner of the display.

Error Conditions

The graphics system must be initialized or this call will be ignored, an ESCAPE (-27) will be generated, and GRAPHICSERROR will return a non-zero value.

LOCATOR_TERM

IMPORT: dgl_lib

This **procedure** disables the enabled locator device.

Syntax

Procedure Heading

```
PROCEDURE LOCATOR_TERM;
```

Semantics

LOCATOR_TERM terminates and disables the enabled locator device. It transmits any termination sequence required by the device and releases all resources being used by the device. The device name is set to the default device name (' '), the device status is set to 0 (not enabled) and the device address is set to 0.

LOCATOR_TERM is the complementary routine to LOCATOR_INIT.

If a locator device is used, LOCATOR_TERM should be called before the application program is terminated.

Error Conditions

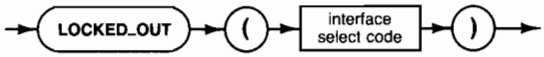
The graphics system must be initialized and a locator device enabled or this call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

LOCKED_OUT

IMPORT: hpib_3
iodeclarations

This BOOLEAN **function** will return TRUE if the specified interface is currently in the local lockout state. If the interface is currently active controller a FALSE value will be returned regardless of the local lockout state.

Syntax



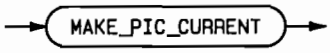
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

MAKE_PIC_CURRENT

IMPORT: dgl_lib

This **procedure** makes the picture current.

Syntax



Procedure Heading

```
PROCEDURE MAKE_PIC_CURRENT;
```

Semantics

The graphics display surface can be made current at any time with a call to MAKE_PIC_CURRENT. This insures that all previously generated primitives have been sent to the graphics display device. Due to operating system delays, all picture changes may not have been displayed on the graphics display upon return to the calling program. MAKE_PIC_CURRENT is most often used in system buffering mode (see SET_TIMING) to make sure that all output has been sent to the graphics display device when required.

Before performing any non-graphics library input or output to an active graphics device, (e.g., a Pascal read or write), it is essential that all of the previously generated output primitives be sent to the device. If immediate visibility is the current timing mode, all primitives will be sent to the device before completion of the call to generate them, but if system buffering is used, MAKE_PIC_CURRENT should be called before performing any non-graphics system I/O.

The following routines implicitly make the picture current:

AWAIT_LOCATOR	DISPLAY_TERM	INPUT_ESC
LOCATOR_INIT	SAMPLE_LOCATOR	

A call to MAKE_PIC_CURRENT can be made at any time within an application program to insure that the image is fully displayed. MAKE_PIC_CURRENT does not modify the current timing mode.

Error Conditions

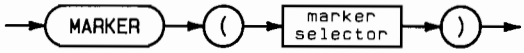
The graphics system must be initialized and a display must be enabled or this call will be ignored, an ESCAPE (− 27) will be generated, and GRAPHICSERROR will return a non-zero value.

MARKER

IMPORT: dgl_lib

This **procedure** outputs a marker symbol at the starting position.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
marker selector	Expression of TYPE INTEGER	MININT TO MAXINT	1 thru 19

Procedure Heading

```
PROCEDURE MARKER ( Marker_num : INTEGER );
```

Semantics

The **marker selector** determines which marker will be output. There are 19 defined invariant marker symbols (1-19). They are defined as follows:

1 - '.'	7 - rectangle	13 - '3'
2 - '+'	8 - diamond	14 - '4'
3 - '*'	9 - rectangle with cross	15 - '5'
4 - 'O'	10 - '0'	16 - '6'
5 - 'X'	11 - '1'	17 - '7'
6 - triangle	12 - '2'	18 - '8'
		19 - '9'

Marker numbers 20 and larger are device dependent.

MARKER outputs the marker designated by the marker selector, centered about the starting position. The starting position is left unchanged at the completion of this call.

If the marker selector specified is greater than the number of distinct marker symbols that are supported by a device, then marker number 1 ('.') will be used. INQ_WS can be used to inquire the number of distinct marker symbols that are available on a particular graphics display device. Depending on a particular display device's capabilities, the graphics library uses either hardware or software to generate the marker symbols.

The size and orientation of markers is fixed and not affected by the viewing transformation. The size of markers is device dependent and cannot be changed.

Only the primitive attributes of color and highlighting apply to markers. However, the marker will appear with these attributes only if the device is capable of applying them to markers.

Error Conditions

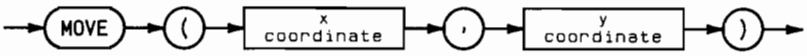
The graphics system must be initialized and a display device enabled or the call will be ignored, an ESCAPE (– 27) will be generated, and GRAPHICSERROR will return a non-zero value.

MOVE

IMPORT: dgl_lib

This **procedure** sets the starting position to the world coordinate specified.

Syntax



Item	Description/Default	Range Restrictions
x coordinate	Expression of TYPE REAL	–
y coordinate	Expression of TYPE REAL	–

Procedure Heading

```
PROCEDURE MOVE ( Wx, Wy : REAL );
```

Semantics

MOVE specifies where the next graphical primitive will be output. It does this by setting the value of the starting position to the world coordinate system point specified by the X,Y coordinate values and then moving the physical beam or pen to that point.

The **x** and **y coordinate** pair is the new starting position in world coordinates.

The starting position corresponds to the location of the physical pen or beam in all but four instances: after a change in the viewing transformation, after initialization of a graphical display device, after the output of a text string, or after the output of a graphical escape function. A call to MOVE or INT_MOVE should therefore be made after any one of the following calls to update the value of the starting position and in so doing, place the physical pen or beam at a known location: SET_ASPECT, DISPLAY_INIT, SET_DISPLAY_LIM, OUTPUT_ESC, TEXT, SET_VIEWPORT, and SET_WINDOW.

Error Conditions

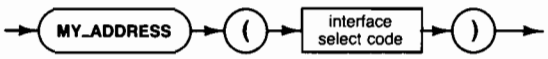
The graphics system must be enabled and a display device enabled or this call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

MY_ADDRESS

IMPORT: hpib_1
iodeclarations

This **function** returns an INTEGER subrange (TYPE type_hpib_addr) representing the HP-IB address of the specified HP-IB interface.

Syntax



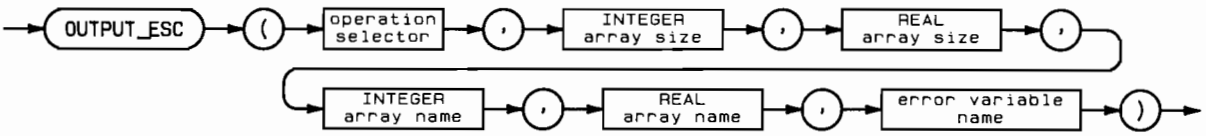
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

OUTPUT_ESC

IMPORT: dgl_lib

This **procedure** performs a device dependent escape function to inquire from the graphics display device.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
operation selector	Expression of TYPE INTEGER	MININT to MAXINT	–
INTEGER array size	Expression of TYPE INTEGER	MININT to MAXINT	>0
REAL array size	Expression of TYPE INTEGER	MININT to MAXINT	>0
INTEGER array name	Any valid variable. Should be INTEGER array	–	–
REAL array name	Any valid variable. Should be REAL array	–	–
error variable name	Variable of TYPE INTEGER	–	–

Procedure Heading

```
PROCEDURE OUTPUT_ESC (         Opcode : INTEGER;
                               Isize  : INTEGER;
                               Rsize  : INTEGER;
                      ANYVAR   Ilist  : Gint_list;
                      ANYVAR   Rlist  : Greal_list;
                         VAR   Ierr   : INTEGER    );
```

Semantics

The **operation selector** determines the device dependent output escape function to be performed. The codes supported for a given device are described in the device handlers section of this document.

The **INTEGER array size** is the number of INTEGER parameters contained in the INTEGER array. The thousand's digit of the operation selector is the number of INTEGER parameters that the graphics system expects.

The **REAL array size** is the number of REAL parameters contained in the REAL array by the escape function. The ten-thousand's digit of the operation selector is the number of REAL parameters that the graphics system expects.

The **INTEGER array** is the array in which zero or more INTEGER parameters are contained.

The **REAL array** is the array in which zero or more REAL parameters are contained.

The **error variable** will contain a value indicating whether the escape function was performed.

Value	Meaning
0	Output escape function successfully sent to the device.
1	Operation not supported by the graphics display device.
2	The INTEGER array size is not equal to the number of required INTEGER parameters.
3	The REAL array size is not equal to the number of required REAL parameters.
4	Illegal parameters specified.

If the error variable contains a non-zero value, the call has been ignored.

OUTPUT_ESC allows application programs to access special device features on a graphics display device. The desired escape function is specified by a unique value for opcode.

The type of information passed to the graphics display device is determined by the value of opcode. The graphics library does not check OUTPUT_ESC parameters which will be sent directly to the display device. This can lead to device dependent results if out of range values are sent.

Output escape functions only apply to the graphics display device.

The starting position may be altered by a call to OUTPUT_ESC.

Error Conditions

The graphics system must be initialized and a display device must be enabled or this call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

Raster Device Escape Operations

Operation Selector	Function
52	Dump graphics to the graphics printer (PRINTER:), if color, all planes are ORed. This operation is not available for the HP 98627A.

53	Await vertical blanking. This escape function will not exit until the CRT is performing vertical blanking.
	The following example shows how to use this function when changing the color table to reduce flicker.
	`OUTPUT_ESC ( 53, 0, 0, dummy, dummy, error );`
	`SET_COLOR_TABLE ( 0, r, g, b );`
	The color table is not changed until the crt is blank (during a refresh cycle). Otherwise changing the color map in the middle of a scan would create a screen that was half the old color, and half the new color for one frame (1/60 sec). To the eye this would look like a flicker.

250	Specify device limits. REAL Array [1] = Points (dots) per mm in X direction REAL Array [2] = Points (dots) per mm in Y direction
1050	Turn on or off the graphics display. INTEGER array [1] = 0 → turn display off. INTEGER array [1] <> 0 → turn display on.
1051	Turn on or off the alpha display. INTEGER array [1] = 0 → turn display off. INTEGER array [1] <> 0 → turn display on.
1052	Set special drawing modes. Using this escape function will redefine the meaning of the set color attribute. For details on how a given drawing mode affects a color see "Drawing Modes" in SET_COLOR. This drawing mode does not apply to device dependent polygons. Out of range values default to dominate drawing mode. INTEGER array[1] = 0 → Dominate drawing mode. = 1 → Non-dominate drawing mode. = 2 → Erase drawing mode. = 3 → Complement drawing mode.
1053	Dump graphics (from the specified color planes) to the graphics printer (PRINTER:). INTEGER array [1] = Color plane selection code. BIT 1 = 1 → Select plane 1. (Blue on HP 98627A) BIT 2 = 1 → Select plane 2. (Green on HP 98627A) BIT 3 = 1 → Select plane 3. (Red on HP 98627A) BIT 4 = 1 → Select plane 4.
1054	Clear selected graphics planes. INTEGER Array [1] = 0 - Clear all planes INTEGER Array [1] <> 0 - Color plane selection code. BIT 1 = 1 Clear plane 1 (Blue on HP 98627A) BIT 2 = 1 Clear plane 2 (Green on HP 98627A) BIT 3 = 1 Clear plane 3 (Red on HP 98627A) BIT 4 = 1 Clear plane 4

Operation Selector	Function
10050	Set all HP 9836C color table locations. This escape function allows the user to change all locations in the hardware color map with one procedure. The software maintained color table will be updated by this call. This escape function is the same as calling SET_COLOR_TABLE with indexes 0 - 15. REAL Array [1] = Parm1 REAL Array [2] = Parm2 Index 0 REAL Array [3] = Parm3 REAL Array [4] = Parm1 REAL Array [5] = Parm2 Index 1 REAL Array [6] = Parm3 : : REAL Array [46] = Parm1 REAL Array [47] = Parm2 Index 15 REAL Array [48] = Parm3 Parm1, Parm2, and Parm3 are defined to be the same as used with SET_COLOR_TABLE. The size of the INTEGER array must equal 0 and the size of the REAL array 48.

The following table shows which escape codes are supported on which series 200 raster displays.

Operation Selector	9816	9826	9836	9836C	98627
52	yes	yes	yes	yes	yes
53	no	no	no	yes	no
250	no	no	no	no	yes
1050	yes	yes	yes	yes	yes
1051	yes	yes	yes	yes	no
1052	yes	yes	yes	yes	yes
1053	no	no	no	yes	yes
1054	yes	yes	yes	yes	yes
10050	no	no	no	yes	no

HPGL Plotter Escape Operations

Operation Selector	Function
1052*	Enable cutter. Provides means to control the Plotter paper cutters. Paper is cut after it is advanced. INTEGER array [1] = 0 Cutter is disabled. INTEGER array [1] <> 0 Cutter is enabled.
1052	Set automatic pen. This instruction provides a means for utilizing the smart pen options of the plotter. Initially, all automatic pen options are enabled. INTEGER array [1]: BIT 1 = 1 Lift pen if it has been down for 60 seconds. BIT 2 = 1 Put pen away if it has been motionless for 20 seconds. BIT 3 = 1 Do not select a pen until a command which makes a mark. This causes the pen to remain in the turret for the longest possible time.
1053	Advance the paper either one half or a full page. INTEGER array [1] = 0 >> Advance page half INTEGER array [1] <> 0 >> Advance page full
2050	Select pen velocity. This instruction allows the user to modify the plotter's pen speed. Pen speed may be set from 1 to the maximum for the given device. INTEGER array [1] = Pen speed (INTEGER from 1 to device max). INTEGER array [2] = Pen number (INTEGER from 1 to 8; other integers select all pens)
2051	Select pen force. The force may be set from 10 to 66 gram-weights. INTEGER array [1] = Pen force (INTEGER from 1 to 8). 1: 10 gram-weights 2: 18 gram-weights 3: 26 gram-weights 4: 34 gram-weights 5: 42 gram-weights 6: 50 gram-weights 7: 58 gram-weights 8: 66 gram-weights INTEGER array [2] = Pen number (INTEGER 1 to 8; other integers select all pens)
2052	Select pen acceleration. The acceleration may be set from 1 to 4 G's. INTEGER array [1] = Pen acceleration (INTEGER from 1 to 4). INTEGER array [2] = Pen number (INTEGER 1 to 8; other integers select all pens)

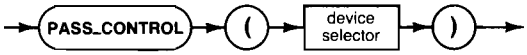
Operation Selector	9872	7580	7585	7470
1052*	S/T	no	no	no
1052	no	yes	yes	no
1053	S/T	no	no	no
2050	yes	yes	yes	yes
2051	no	yes	yes	no
2052	no	yes	yes	no

PASS_CONTROL

IMPORT: hpib_2
iodeclarations

This **procedure** passes active control from the specified interface to another device on the bus.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary

Semantics

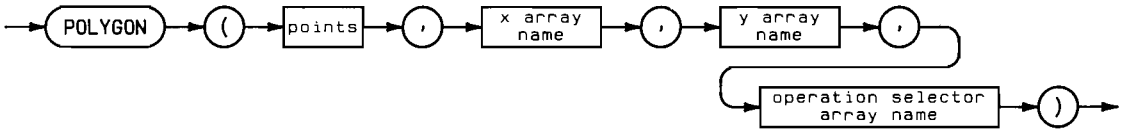
	System Controller		Net System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN TCT ATN	ATN UNL TAG TCT ATN	ATN TCT ATN	ATN UNL TAG TCT ATN
Not Active Controller	Error			

POLYGON

IMPORT: dgl_types
dgl_lib
dgl_poly

This **procedure** displays a polygon-set starting and ending at the specified point adhering to the specified polygon style exactly as specified (i.e., device-independent results).

Syntax



Item	Description/Default	Range Restrictions
points	Expression of TYPE INTEGER	MININT thru MAXINT
x array name	Array of TYPE *Greal_list.*	–
y array name	Array of TYPE *Greal_list.*	–
operation selector array name	Array of TYPE *Gshortint_list. Gshortint* is a sub-range of INTEGER.	–32 768 to 32 767

Procedure Heading

```
PROCEDURE POLYGON ( Npoint     : INTEGER;
                    ANYVAR   Xvec     : Greal_list;
                    ANYVAR   Yvec     : Greal_list;
                    ANYVAR   Opcodes  : Gshortint_list);
```

Semantics

Points is the number of vertices in the polygon set.

The **x** and **y coordinate arrays** contain the world coordinate values for each vertex of the polygon-set. The vertices must be in order. The vertices for the first sub-polygon must be at the beginning of these arrays, followed by the vertices for the second sub-polygon, etc. So, the coordinate arrays must contain a total number of vertices that equals points.

The **operation selector array** contains a series of integer operation selectors defining which vertices start new polygons, and defining which edges should be displayed.

Value	Meaning
0	Don't display the line for the edge extending to this vertex from the previous vertex.
1	Display the line for the edge extending to this vertex from the previous vertex.
2	This vertex is the first vertex of a sub-polygon. Succeeding vertices are part of a sub-polygon until a new start-of-polygon operation selector (2) is encountered. (Or the end of the arrays is encountered.)

Note

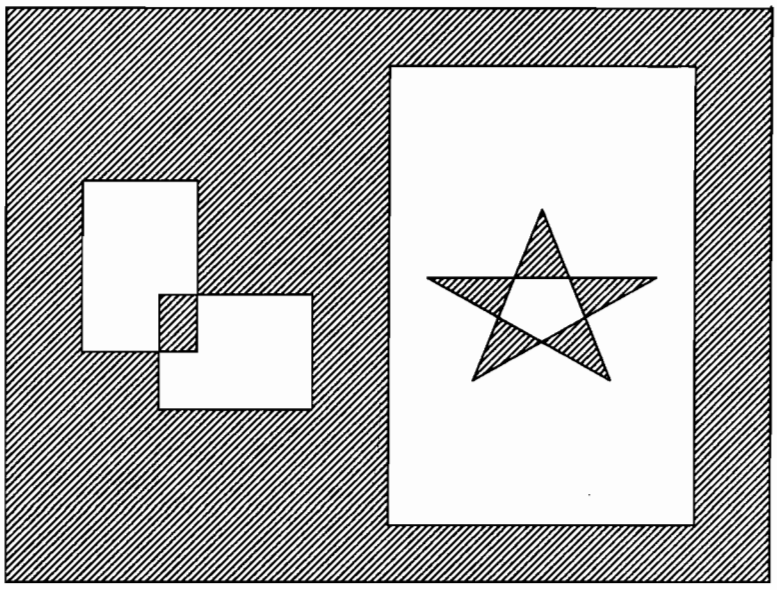
The first entry in the operation selector array **must** be 2, since it is the first vertex of a sub-polygon.

POLYGON is used to output a polygon-set, specified in world coordinates, adhering exactly to the polygon style attributes that are currently specified. A polygon-set is a set of polygons (called "sub-polygons") that are treated graphically as one polygon. This is accomplished by "stacking" the sub-polygons. The subpolygons in a polygon-set may intersect or overlap each other.

The edge of a sub-polygon is defined as the line sequence that connects its vertices in the order specified. If the last vertex specified for a sub- polygon is not the same as the first, they are automatically connected.

When a polygon-set is displayed, the primitive attributes for polygons and lines define its appearance. In particular, the interior of the polygon-set will be filled according to the attributes of polygon style, polygon interior color and polygon interior line-style. If the edges are to be displayed as specified in the polygon style, the edges will adhere to the current line attributes of color, line-style and line-width. A dot will disappear on an edged polygon if the edge is done with a complementing line.

The filling of polygons also depends on how the sub-polygons "nest" within each other. An "even-odd" rule is used for determining which areas will be filled. Moving across the screen, count the edges of the polygon. Odd-numbered edges will turn the fill on and even-numbered edges will turn the fill off. The picture below will help clear up how the fills work.



Polygon Filling

Refer to SET_PGN_TABLE, SET_PGN_STYLE, SET_PGN_COLOR, SET_PGN_LS for a more detailed description of how attributes affect polygons.

As stated above, the values in the operation selector array define how the edges of the sub-polygons are displayed. The edge from the (I-1)th vertex to the Ith vertex will only be displayed if the Ith entry in the operation selector array equals 1. To display the edge from the last vertex to the first vertex of a sub-polygon, the first vertex must be explicitly respecified after all the other vertices of the sub-polygon, with an operation selector equal to 1. Otherwise the edge from the last vertex to the first will not be drawn. It will, however, automatically be connected for polygon filling.

If it is within the capabilities of the device, filling of the sub-polygon will be done to the sub-polygon edges regardless of whether the edges are displayed. If an entry in the operation selector array does not equal 0, 1, or 2, it will be treated as if it were equal to 0 and the edge will not be drawn.

When POLYGON is used, the current position is updated to the end of the last sub-polygon specified in the polygon-set. The end of the last sub-polygon is defined to be the first (implicit last) vertex of the subpolygon. So, if there is only one vertex in a polygon-set this call degenerates to an update of the current position to the first coordinate set in the x and y point arrays (x coordinate array[1], y coordinate array[1]).

It is the application program's responsibility to ensure that the arrays are all dimensioned to at least the number of elements specified by points and that at least that many values are contained in each array.

Polygons are defined to be closed surfaces. When a sub-polygon extends beyond a clipping edge the closed nature of the sub-polygon is destroyed. As with other primitives, unpredictable results may occur if the sub-polygon extends beyond the clipping window.

Error Conditions

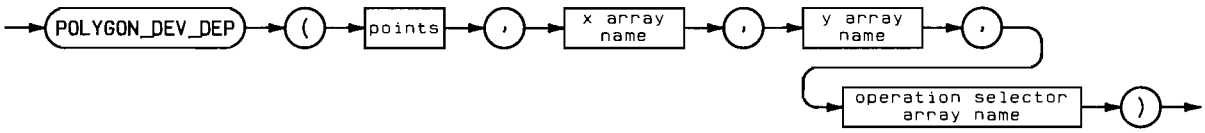
The graphics system must be initialized, a graphics display must be enabled, all parameters must be within specified limits and the number of points specified must be greater than 0 or the call will be ignored, an ESCAPE (− 27) will be generated, and GRAPHICSERROR will return a non-zero value.

POLYGON_DEV_DEP

IMPORT: dgl_types
dgl_lib
dgl_poly

This **procedure** displays a polygon-set starting and ending at the specified point adhering to the specified polygon style in a device- dependent fashion.

Syntax



Item	Description/Default	Range Restrictions
points	Expression of TYPE INTEGER	MININT thru MAXINT
x array name	Array of TYPE *Greal_list.*	–
y array name	Array of TYPE *Greal_list.*	–
operation selector array name	Array of TYPE *Gshortint_list. Gshortint* is a sub-range of INTEGER.	–32 768 to 32 767

Procedure Heading

```
PROCEDURE POLYGON_DEV_DEP ( Npoint    : INTEGER;
                  ANYVAR   Xvec     : Greal_list;
                  ANYVAR   Yvec     : Greal_list;
                  ANYVAR   Opcodes  : Gshortint_list);
```

Semantics

Points is the number of vertices in the polygon set.

The **x** and **y coordinate arrays** contain the world coordinate values for each vertex of the polygon-set. The vertices must be in order. The vertices for the first sub-polygon must be at the beginning of these arrays, followed by the vertices for the second sub-polygon, etc. So, the coordinate arrays must contain a total number of vertices that equals points.

The **operation selector array** contains a series of integer operation selectors defining which vertices start new polygons, and defining which edges should be displayed.

Value	Meaning
0	Don't display the line for the edge extending to this vertex from the previous vertex.
1	Display the line for the edge extending to this vertex from the previous vertex.
2	This vertex is the first vertex of a sub-polygon. Succeeding vertices are part of a sub-polygon until a new start-of-polygon operation selector (2) is encountered. (Or the end of the arrays is encountered.)

Note

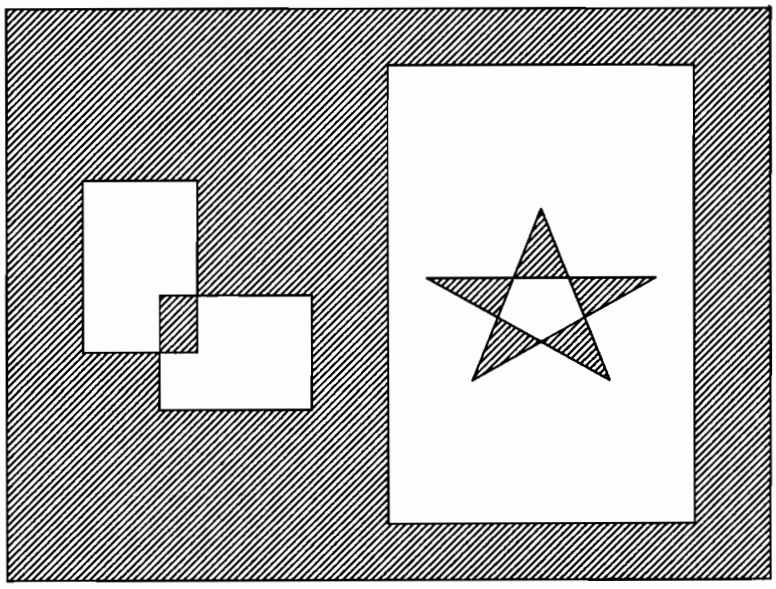
The first entry in the operation selector array **must** be 2, since it is the first vertex of a sub-polygon.

POLYGON_DEV_DEP is used to output a polygon-set, specified in world coordinates, adhering within the capabilities of the device to the polygon style attributes that are currently specified. A polygon-set is a set of polygons (called "sub-polygons") that are treated graphically as one polygon. The subpolygons in a polygon-set may intersect or overlap each other.

The edge of a sub-polygon is defined as the line sequence that connects its vertices in the order specified. If the last vertex specified for a sub- polygon is not the same as the first, they are automatically connected.

When a polygon-set is displayed, the primitive attributes for polygons and lines define its appearance. In particular, the interior of the polygon-set will be filled according to the attributes of polygon style, polygon interior color and polygon interior line-style. If the edges are to be displayed as specified in the polygon style, the edges will adhere to the current line attributes of color, line-style and line-width.

The filling of polygons also depends on how the sub-polygons "nest" within each other. An "even-odd" rule is used for determining which areas will be filled. Moving across the screen, count the edges of the polygon. Odd-numbered edges will turn the fill on and even-numbered edges will turn the fill off. The picture below will help clear up how the fills work.



Polygon Filling

Refer to SET_PGN_TABLE, SET_PGN_STYLE, SET_PGN_COLOR, SET_PGN_LS for a more detailed description of how attributes affect polygons.

As stated above, the values in the operation selector array define how the edges of the sub-polygons are displayed. The edge from the (I-1)th vertex to the Ith vertex will only be displayed if the Ith entry in the operation selector array equals 1. To display the edge from the last vertex to the first vertex of a sub-polygon, the first vertex must be explicitly respecified after all the other vertices of the sub-polygon, with an operation selector equal to 1. Otherwise the edge from the last vertex to the first will not be drawn. It will, however, automatically be connected for polygon filling.

If it is within the capabilities of the device, filling of the sub-polygon will be done to the sub-polygon edges regardless of whether the edges are displayed. If an entry in the operation selector array does not equal 0, 1, or 2, it will be treated as if it were equal to 0, i.e., the edge will not be drawn.

When POLYGON_DEV_DEP is used, the current position is updated to the end of the last sub-polygon specified in the polygon-set. The end of the last sub-polygon is defined to be the first (implicit last) vertex of the subpolygon. So, if there is only one vertex in a polygon-set this call degenerates to an update of the current position to the first coordinate set in the x and y point arrays (x coordinate array[1], y coordinate array[1]).

It is the application program's responsibility to ensure that the arrays are all dimensioned to at least the number of elements specified by points and that at least that many values are contained in each array.

Device capabilities vary widely. Not all devices are able to draw polygon edges as requested. If a device is not able to draw polygon edges as requested, they will be simulated in software. The simulation will always adhere to the edge value in SET_PGN_STYLE and the operation selector in POLYGON_DEV_DEP, but the line-style and color of the edge will depend on the capability of the device to produce lines with those attributes.

Polygon fill capabilities can vary widely between devices. A device may have no filling capabilities at all, may be able to perform only solid fill, or may be able to fill polygons with different fill densities and at different fill line orientations. POLYGON_DEV_DEP tries to match the device capabilities to the request. If the device cannot fill the request at all, then no simulation is done and the polygon will not be filled. For HPGL plotters, the fill is simulated. For raster devices, if the density is greater than 0.5, a solid fill is used, otherwise, the fill is simulated.

In the case where the polygon style specifies non-display of edged, this would result in no visible output although visible output had been specified. To provide some visible output in this case, POLYGON_DEV_DEP will outline the polygon using the color and line-style specified for the fill lines. However, only those edge segments specified as displayable by the operation selector array will be drawn. Therefore, if all edge segments are specified as non-displayed, there will still be no visible output.

Regardless of the capabilities of the device, POLYGON_DEV_DEP sets the starting position to the first vertex of the last member polygon specified in the call. If there is only one polygon specified, the starting position will therefore be set to the first vertex specified.

Polygons are defined to be closed surfaces. When a sub-polygon extends beyond a clipping edge the closed nature of the sub-polygon is destroyed. As with other primitives, unpredictable results may occur if the sub-polygon extends beyond the clipping window.

Error Conditions

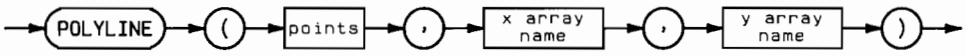
The graphics system must be initialized, a graphics display must be enabled, all parameters must be within specified limits and the number of points (Points) must be greater than 0 or the call will be ignored, an ESCAPE (− 27) will be generated, and GRAPHICSERROR will return a non-zero value.

POLYLINE

IMPORT: dgl_lib

This **procedure** draws a connected line sequence starting at the specified point.

Syntax



Item	Description/Default	Range Restrictions
points	Expression of TYPE INTEGER	MININT thru MAXINT
x array name	Array of TYPE *Greal_list.*	–
y array name	Array of TYPE *Greal_list.*	–

Procedure Heading

```
PROCEDURE POLYLINE (       Npts          : INTEGER;
                    ANYVAR Xvec, Yvec : Greal_list )
```

Semantics

Points is the number of vertices in the polygon set.

The **x** and **y coordinate arrays** contain the world coordinate values for each vertex of the polyline-set. The vertices must be in order. The vertices for the first sub-polyline must be at the beginning of these arrays, followed by the vertices for the second sub-polyline, etc. So, the coordinate arrays must contain a total number of vertices that equals points.

The procedure POLYLINE provides the capability to draw a series of connected lines starting at the specified point. A complete object can be drawn by making one call to this procedure. This call first sets the starting position to be the first elements in the x and y coordinate arrays. The line sequence begins at this point and is drawn to the second element in each array, then to the third and continues until points-1 lines are drawn.

This procedure is equivalent to the following sequence of calls:

```
MOVE (X_coordinate_array[1],Y_coordinate_array[1]);
LINE (X_coordinate_array[2],Y_coordinate_array[2]);
LINE (X_coordinate_array[3],Y_coordinate_array[3]);
            :
            :
LINE (X_coordinate_array[Points],Y_coordinate_array[Points]);
```

The starting position is set to (X_coordinate_array[Points], Y_coordinate_array[Points]) at the completion of this call.

Specifying only one element, or Points equal to 1, causes a move to be made to the world coordinate point specified by the first entries in the two coordinate arrays.

It is the application program's responsibility to ensure that the arrays are all dimensioned to at least the number of elements specified by points and that at least that many values are contained in each array.

Depending on the nature of the current line-style nothing may appear on the graphics display. See SET_LINE_STYLE for a complete description of how line-style effects a particular point or vector.

The primitive attributes of color, line-style, and line-width apply to polylines.

Error Conditions
The graphics system must be initialized, a graphics display must be enabled, all parameters must be within specified limits and the number of points (points) must be greater than 0 or the call will be ignored, an ESCAPE (– 27) will be generated, and GRAPHICSERROR will return a non-zero value.

PPOLL

IMPORT: hpib_3
iodeclarations

This **function** will perform an HP-IB parallel poll. This involves setting the ATN and EOI bus lines on the specified interface and then read the data bus lines after waiting 25usec. The ATN and EOI lines are then returned to the clear state.

Syntax

Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

Semantics

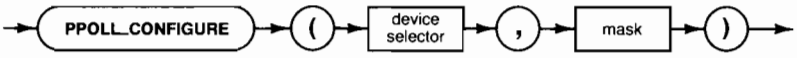
	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN & EOI (duration≥25μs) Read byte $\overline{\text{EOI}}$ Restore ATN to previous state	Error	ATN & EOI (duration≥25μs) Read byte $\overline{\text{EOI}}$ Restore ATN to previous state	Error
Not Active Controller	Error			

PPOLL_CONFIGURE

IMPORT: hpib_2
iodeclarations

This **procedure** programs the logical sense and data bus lines, a devices parallel poll response.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary
mask	Expression of TYPE INTEGER.	MININT thru MAXINT	0 thru 15

Semantics

This procedure assumes that the device's response is bus-programmable. The computer must be active controller to execute this statement.

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	Error	ATN MTA UNL LAG PPC PPE	Error	ATN MTA UNL LAG PPC PPE
Not Active Controller	Error			

The mask is coded. The three least significant bits determine the data bus line for the response. The fourth bit determines the logical sense of the response.

Note

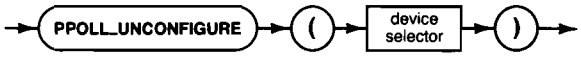
Use of PPOLL_CONFIGURE may interfere with the Pascal Operating System, especially if an external disk is being used. **Be very careful.**

PPOLL_UNCONFIGURE

IMPORT: hpib_2
iodeclarations

This **procedure** will cause the specified device(s) to disable the parallel poll response.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary

Semnantics

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN PPU	ATN MTA UNL LAG PPC PPD	ATN PPU	ATN MTA UNL LAG PPC PPD
Not Active Controller	Error			

Note

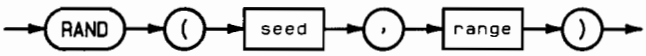
Use of PPOLL_UNCONFIGURE may interfere with the Pascal Operating System, especially if an external disk is being used. **Be very careful.**

RAND

IMPORT: rnd
sysglobals

This SHORTINT function returns a random number greater than or equal to zero and less than the specified SHORTINT range.

Syntax



Item	Description/Default	Range Restrictions
seed	INTEGER	1 thru MAXINT − 1
range	SHORTINT	1 thru 2^{31} − 1

Semantics

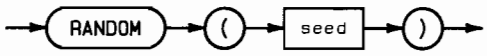
Given a **seed** and a **range**, the random number generator function returns a random number greater than or equal to zero and less than the range. It also randomizes the seed INTEGER.

RANDOM

IMPORT: rnd

This procedure takes a seed INTEGER, randomizes it and returns the new random number in the seed variable.

Syntax



Item	Description/Default	Range Restrictions
seed	INTEGER	1 thru MAXINT −1

Semantics

When the following program is run, the RANDOM procedure returns all $2^{31} - 1$ INTEGERS before repeating a value.

```
program test(output);

import RND;

var seed : INTEGER;
    doomsday : BOOLEAN;

begin
    seed := 1234;
    doomsday := false;

  repeat
    RANDOM(seed);
    write(seed);
  until doomsday;

end.
```

READBUFFER

IMPORT: general_4
iodeclarations

This **procedure** will read a single byte from the buffer space and update the empty pointer in the *buf_info* record. An error will occur when a read is attempted beyond the end of valid data.

Syntax

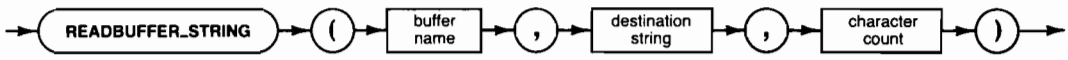
Item	Description/Default	Range Restrictions
buffer name	Variable of TYPE *buf_info_type*.	See Chapter 11
destination character	Variable of TYPE CHAR.	–

READBUFFER_STRING

IMPORT: general_4
iodeclarations

This **procedure** will read the specified number of characters from the buffer and put them into the string variable. The empty pointer is updated. If the string is not big enough or if there is insufficient data in the buffer there will be an error.

Syntax



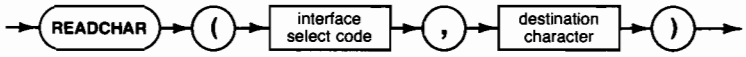
Item	Description/Default	Range Restrictions	Recommended Range
buffer name	Variable of TYPE *buf_info_type*.	See Chapter 11	
destination string	Variable of TYPE STRING.	–	
character count	Expression of TYPE INTEGER.	MININT thru MAXINT	0 thru 255

READCHAR

IMPORT: general_1
iodeclarations

This **procedure** will read a single byte from the specified interface.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
destination character	Variable of TYPE CHAR.		

Semantics

If no character is ready the routine will wait until the character comes in or until a timeout occurs (if any was set up).

An HPIB interface must be addressed as a listener before performing a READCHAR, or an error will be generated. To avoid this, use the following sequence:

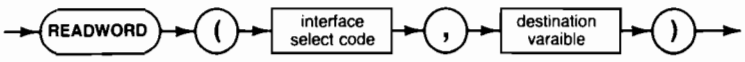
```
    .
    .
TALK (7,24);
UNLISTEN(7)
LISTEN( 7, MY_ADDRESS(7));
READCHAR (7, Character);
    .
    .
```

READWORD

IMPORT: general_1
iodeclarations

This **procedure** will read 2 bytes from interfaces that are byte-oriented. The GPIO card and any other word-oriented interface will read a single 16 bit quantity.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
destination variable	Variable of TYPE INTEGER.		

Semantics

An interface less than 16-bits wide will be read into the most-significant-byte first, then into the lease-significant-byte.

An HP-IB interface must be addressed as a listener before performing a READWORD, or an error will be generated. To avoid this, use the following sequence:

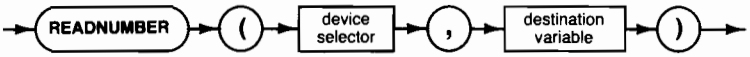
```
    .
    .
TALK (7,24);
LISTEN( 7, MY_ADDRESS(7));
READWORD (7, Character);
    .
    .
```

READNUMBER

IMPORT: general_2
iodeclarations

This **procedure** will perform a free field numeric entry from the specified device.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary
destination variable	Variable of TYPE REAL.		

Semantics

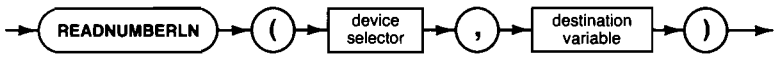
The routine will skip over non-numerics until a valid ASCII number is encountered. The number will be processed until a non-numeric value is read from the interface, or until 256 characters have been read. No further characters are read.

READNUMBERLN

IMPORT: general_2
iodeclarations

This **procedure** will read in a free field number and then search for a line feed:

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary
destination variable	Variable of TYPE REAL.		

Semantics

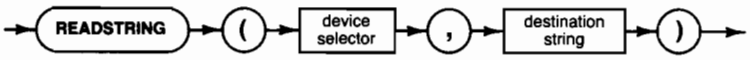
The routine will skip over non-numerics until a valid number is encountered. The number will be processed until a non-numeric value is read from the interface. If a line feed is the next character, no more characters are read, otherwise, characters are read until a line feed is encountered.

READSTRING

IMPORT: general_2
iodeclarations

This **procedure** will read in characters to the specified string.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary
destination string	Variable of TYPE STRING.		

Semantics

This procedure will read characters into the specified string until one of the following conditions occur :

- a carriage return & line feed are read
- a line feed is read
- the string is filled up

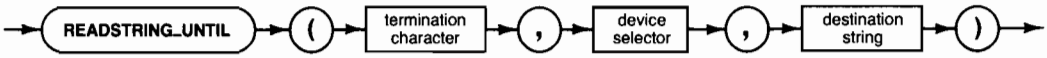
The line feed or carriage return/line feed are not put into the string.

READSTRING_UNTIL

IMPORT: general_2
iodeclarations

This **procedure** will read characters from the selected device into the specified string until the prescribed terminator is encountered.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
termination character	Expression of TYPE CHAR.	–	
device selector	Expression f TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary
destination string	Variable of TYPE STRING.		

Semantics

This procedure will read characters into the string until one of the following conditions occurs :

- termination character is received
- the string is filled

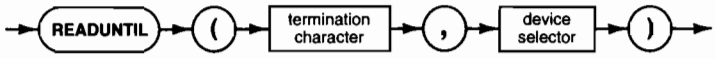
The termination character is placed into the string.

READUNTIL

IMPORT: general_2
iodeclarations

This **procedure** will read characters until the match character occurs. All characters read in will be thrown away.

Syntax



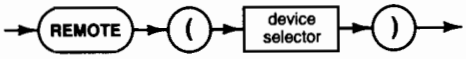
Item	Description/Default	Range Restrictions	Recommended Range
termination character	Expression of TYPE CHAR.	–	
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary

REMOTE

IMPORT: hpib_2
iodeclarations

This **procedure** sends the messages to place the bus device(s) into the remote state.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary

Semantics

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	$\overline{\text{REN}}$ $\overline{\text{ATN}}$	REN ATN MTA UNL LAG	Error	
Not Active Controller	REN	Error	Error	

REMOTED

IMPORT: hpib_3
iodeclarations

This BOOLEAN **function** indicates if the REN line is being asserted. The interface should be non-system controller.

Syntax

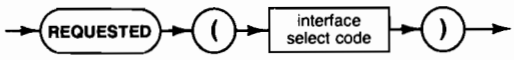
Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary

REQUESTED

IMPORT: hpib_3
iodeclarations

This BOOLEAN **function** returns TRUE if any device is currently asserting the SRQ line. The interface must be active controller.

Syntax



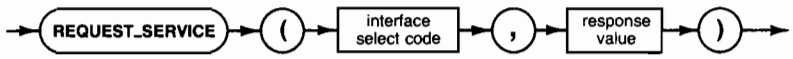
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

REQUEST_SERVICE

IMPORT: hpib_3
iodeclarations

This **procedure** will set up the spoll response byte in the specified interface. If bit 6 is set, SRQ will be set. The interface must not be active controller.

Syntax



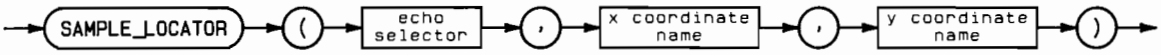
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
response value	Expression of TYPE INTEGER.	MININT thru MAXINT	0 thru 255

SAMPLE_LOCATOR

IMPORT: dgl_lib

This **procedure** samples the current locator device

Syntax



Item	Description/Default	Range Restrictions
echo selector	Expression of TYPE INTEGER	MININT to MAXINT
x coordinate name	Variable of TYPE REAL	–
y coordinate name	Variable of TYPE REAL	–

Procedure Heading

```
PROCEDURE SAMPLE_LOCATOR (     Echo    : INTEGER;
                           VAR Wx, Wy : REAL      );
```

Semantics

The **echo selector** determines the level of input echoing. Possible values are:

0 - No echo.
≥1 - Echo on the locator device.

The **x** and **y coordinates** are the values of the coordinates, expressed in world coordinate units, returned from the enabled locator device.

SAMPLE_LOCATOR returns the current world coordinate value of the locator without waiting for any user intervention. Typically, the locator is sampled in applications involving the continuous input of data points that are very close together.

If the point sampled is outside of the current logical locator limits, the transformed point will still be returned .

The number of echoes supported by a locator device and the correlation between the echo value and the type of echoing performed is device dependent. Most locator devices support at least one form of echoing. Possible echoes are beeping, displaying the point sampled, etc. See the locator descriptions below to find the locators supported by the various devices. If the echo value is larger than the number of echoes supported by the enabled locator device, then echo 1 will be used.

Locator echoing can only be performed on the locator device. The locator echo position is not used in conjunction with any echoes performed while sampling a locator.

SAMPLE_LOCATOR implicitly makes the picture current before sampling the locator.

The Knob as Locator
The keyboard beeper is sounded when the locator is sampled if an echo is selected (echo selector≥1). The sample locator function returns the last AWAIT_LOCATOR result or 0.0, 0.0 if AWAIT_LOCATOR has not been invoked since LOCATOR_INIT.

HPGL Locators
The sample locator function returns the current locator position without waiting for an operator response (pen position on plotters). On a 9111A graphics Tablet, the beeper is sounded when the stylus is depressed. For echo selectors greater than or equal to 9, the same echo as echo selector 1 is used.

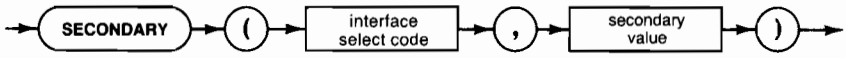
Error Conditions
The graphics system must be initialized and a locator device enabled or this call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

SECONDARY

IMPORT: hpib_2
iodeclarations

This **procedure** will send a secondary command byte over the bus. The interface must be active controller.

Syntax



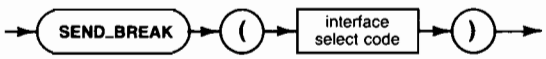
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
secondary value	Expression of TYPE *type_hpib_addr*. This is an INTEGER subrange.	0 thru 31	

SEND_BREAK

IMPORT serial_3
iodeclarations

This **procedure** will send a break to the selected serial interface. (A break is an extended mark period followed by an extended space period.)

Syntax



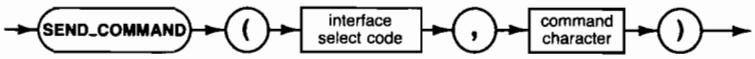
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

SEND_COMMAND

IMPORT: hpib_1
iodeclarations

This **procedure** sends a single byte over the HP-IB interface with attention true. The computer needs to be active controller when this happens.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
command character	Expression of TYPE CHAR.		

Semantics

Note

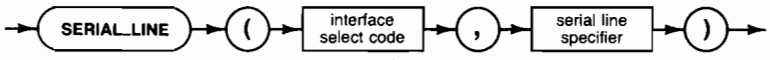
Use of PPOLL_CONFIGURE may interfere with the Pascal Operating System, especially if an external disk is being used. **Be very careful**.

SERIAL_LINE

IMPORT: serial_0
iodeclarations

This BOOLEAN **function** returns TRUE if the specified line on the serial interface is asserted.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
serial line specifier	Expression of enumerated TYPE *type_serial_line*.	rts_line cts_line dcd_line dsr_line drs_line ri_line dtr_line	

Semantics

The values of the enumerated TYPE type_serial_line have the following definitions:

name	RS-232 line
rts	ready to send
cts	clear to send
dcd	data carrier detect
dsr	data set ready
drs	data rate select
dtr	data terminal ready
ri	ring indicator

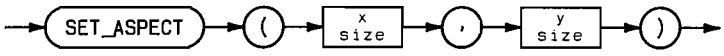
The access to the various lines is determined by the use of an Option 1 or Option 2 connector on the selected interface.

SET_ASPECT

IMPORT: dgl_lib

This **procedure** redefines the aspect ratio of the virtual coordinate system.

Syntax



Item	Description/Default	Range Restrictions
x size	Expression of TYPE REAL	–
y size	Expression of TYPE REAL	–

Procedure Heading

```
PROCEDURE SET_ASPECT ( X_size, Y_size : REAL );
```

Semantics

The **x size** is the width of the virtual coordinate system in dimensionless units. The size must be greater than zero.

The **y size** is the height of the virtual coordinate system in dimensionless units. The size must be greater than zero.

SET_ASPECT sets the aspect ratio of the virtual coordinate system, and hence the view surface, to be y size divided by x size. A ratio of 1 defines a square virtual coordinate system, a ratio greater than 1 specifies it to be higher than it is wide; and a ratio less than 1 specifies it to be wider than it is high. Since x size and y size are used to form a ratio, they may be expressed in any units as long as they are the same units.

The range of coordinates for the virtual coordinate system is calculated based on the value of the aspect ratio. The coordinates of the longer axis are always set to range from 0.0 to 1.0 and those of the shorter axis from 0 to a value that achieves the specified aspect ratio. SET_ASPECT defines the limits of the virtual coordinate system.

ASPECT RATIO (AR)	X LIMITS	Y LIMITS
AR < 1	0.0, 1.0	0.0, 1.0 * AR
AR = 1	0.0, 1.0	0.0, 1.0
AR > 1	0.0, 1.0 / AR	0.0, 1.0

When a call to SET_ASPECT is made, the graphics system sets the viewport equal to the limits of the virtual coordinate system. This routine can therefore be used to access the entire logical display surface. A program could display an image on the entire HP 9826 graphics display, which has an aspect ratio of 399/299, in the following manner:

```
SET_ASPECT ( 399, 299 );
```

To set the aspect ratio to the entire display in a device independent manor, INQ_WS may be used as follows:

```
PROCEDURE Set_max_aspect;

    CONST   Get_aspect=254;

VAR     Dummy      : INTEGER;
        Error      : INTEGER;
        Ratio_list: ARRAY[1..2] OF REAL;

    BEGIN {PROCEDURE Set_max_aspect}
       INQ_WS (Get_aspect,0,0,2,Dummy,Dummy, Ratio_list, Error);
       IF Error=0 THEN
          SET_ASPECT(1.0,Ratio_list[2]);
    END; {PROCEDURE Set_max_aspect}
```

The initial value of the aspect ratio is 1, setting the virtual coordinate system to be a square. This square is mapped to the largest inscribed square on any display surface, so that the viewable area is maximized. As a result, the initial virtual coordinate system limits range from 0.0 to 1.0 in both the X and Y directions. A program can access the largest inscribed rectangle on any display surface by modifying the value of the aspect ratio. The exact placement of the rectangle on the display surface is device dependent, but it is centered on CRT's and justified in the lower left hand corner of plotters.

The starting position is not altered by this call. Since this call redefines the viewing transformation, the starting position may no longer represent the last world coordinate position. A call to MOVE or INT_MOVE should therefore be made after this call to update the starting position.

If the logical locator is associated with the same physical device as the graphics display, then a call to SET_ASPECT will set the logical locator limits equal to the new limits of the virtual coordinate system.

Since the window is not affected by the SET_ASPECT procedure, distortion may result in the window to viewport mapping if the window does not have the same aspect ratio as the virtual coordinate system (see SET_WINDOW).

The locator echo position is set to the default value by this procedure.

Error Conditions

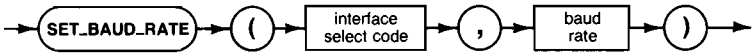
The graphics system must be initialized and both X and Y size must be greater than zero or this call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

SET_BAUD_RATE

IMPORT: serial_3
iodeclarations

This **procedure** will set the serial interface to the specified baud rate.

Syntax



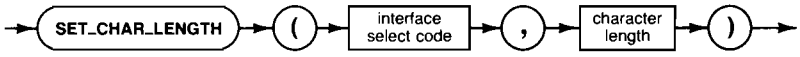
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
baud rate	Expression of TYPE REAL.	–	50 thru 19200 (for 98628)

SET_CHAR_LENGTH

IMPORT: serial_3
iodeclarations

This **procedure** specifies the character length for serial communications, in bits. The valid range of values is 5..8.

Syntax



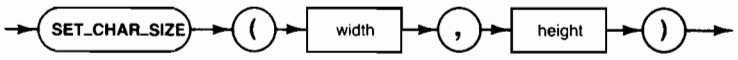
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
character length	Expression of TYPE INTEGER.	MININT thru MAXINT	5 thru 8

SET_CHAR_SIZE

IMPORT: dgl_lib

This **procedure** sets the character size attribute for graphical text.

Syntax



Item	Description/Default	Range Restrictions
width	Expression of TYPE REAL	–
height	Expression of TYPE REAL	–

Procedure Heading

```
PROCEDURE SET_CHAR_SIZE ( Width, Height : REAL );
```

Semantics

The **width** is the requested graphics character cell width in world coordinate units. (width <> 0.0)

The **height** is the requested graphics character cell height in world coordinate units. (height <> 0.0)

SET_CHAR_SIZE sets the character size for subsequently output graphics text. The absolute value of width and height are used to specify the world coordinate size of a character cell. Therefore, the actual physical size of a character output is determined by applying the current viewing transformations to the world coordinate units specification.

The default character size (set by GRAPHICS_INIT and DISPLAY_INIT) is dependent upon the physical device associated with the graphical display device. The size is determined as follows:

- Height := .05 x (height of the world coordinate system)
- Width := .035 x (width of the world coordinate system)

If a change is made to the viewing transformation (by SET_WINDOW, SET_VIEWPORT, SET_DISPLAY_LIM, or SET_ASPECT), the value of the character size attribute will not be changed, but the actual size of the characters generated may be modified.

Error Conditions

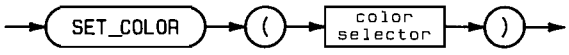
The graphics system must be initialized, a display must be enabled, and width and height must both be non-zero or this call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

SET_COLOR

IMPORT: dgl_lib

This **procedure** sets the color attribute for output primitives except for polygon interior fill.

Syntax



Item	Description/Default	Range Restrictions
color selector	Expression of TYPE INTEGER	–

Procedure Heading

```
PROCEDURE SET_COLOR ( Color : INTEGER );
```

Semantics

SET_COLOR sets the color attribute for the following primitives:

Lines
Markers
Polylines
Polygon Edges
Text

At device initialization a default color table is created by the graphics system. The size and contents of the table are device dependent. At least one entry exists for all devices. A call to INQ_WS with OPCODE equal to 1053 will return the number of colors available on a given graphics device. Some devices allow the color table to be modified with SET_TABLE.

The **color selector** is an index into the color table. The contents of the color table are then used to specify the color when primitives are drawn. On some devices (HPGL plotters), the color selector maps directly to a pen number for the device. On the HP 9836C, the entries in the color table can be modified with SET_COLOR_TABLE.

The default value of the color attribute is 1. If the value of the color selector is not supported on the graphics display, the color attribute will be set to 1.

A color selector of 0 has special effects depending on the graphics display used. For raster devices, a color selector of 0 means to draw in the background color. For most plotters, it puts the pen away.

If the device is not capable of reproducing a color in the color table, the closest color which the device is capable of reproducing is used instead. On some devices, this may depend on the primitive being displayed. For example, the HP98627A color output interface card is capable of a large selection of polygon fill colors, but only 8 line colors. Thus, the fill color could match the selected color much more closely than the line color used to outline the polygon.

Default Raster Color Map

The following table shows the default (initial) color table for the black and white displays (HP 9816 / HP 9920 / HP 9826 / HP 9836):

Index #	Hue	Saturation	Luminosity
0	0	0	0
1	0	0	1.0000
2	0	0	0.9375
3	0	0	0.8750
4	0	0	0.8125
5	0	0	0.7500
6	0	0	0.6875
7	0	0	0.6250
8	0	0	0.5625
9	0	0	0.5000
10	0	0	0.4375
11	0	0	0.3750
12	0	0	0.3125
13	0	0	0.2500
14	0	0	0.1875
15	0	0	0.1250
16	0	0	0.0625

Colors 17 though 31 are set to white.

The following table shows the default (initial) color table for the color displays (HP 9836C and HP 98627A):

Index #	Color name	Red	Green	Blue
0	Black	0.000000	0.000000	0.000000
1	White	1.000000	1.000000	1.000000
2	Red	1.000000	0.000000	0.000000
3	Yellow	1.000000	1.000000	0.000000
4	Green	0.000000	1.000000	0.000000
5	Cyan	0.000000	1.000000	1.000000
6	Blue	0.000000	0.000000	1.000000
7	Magenta	1.000000	0.000000	1.000000
8	Black	0.000000	0.000000	0.000000
9	Olive green	0.800000	0.733333	0.200000
10	Aqua	0.200000	0.400000	0.466667
11	Royal blue	0.533333	0.400000	0.666667
12	Violet	0.800000	0.266667	0.400000
13	Brick red	1.000000	0.400000	0.200000
14	Burnt orange	1.000000	0.466667	0.000000
15	Grey brown	0.866667	0.533333	0.266667

Colors 9 though 15 are a graphic designers idea of colors for business graphics. Color table entries not shown above are set to white.

Raster Drawing Modes

For raster devices (e.g., HP 9836 display) the effect of the color selectors depends on the current drawing mode (drawing mode is set using the OUTPUT_ESC function). The color selectors and their effects are listed below:

Mode	Color Selector = 0	Color Selector >=1
DOMINATE (Default mode)	Background (erase, set bits to 0)	Draw (set bits to 1, overwrite current pattern)
NON-DOMINATE	Background (erase, set bits to 0)	Draw (set bits to 1 Inclusive OR with current pattern)
ERASE	Background (erase, set bits to 0)	Background (erase, set bits to 0)
COMPLEMENT	Background (erase, set bits to 0)	Complement (Invert bits in selected planes)

Plotters

A Color Selector of 0 selects no pens (the current pen is put away). The supported range of Color Selectors for each supported plotter is:

- 9872A - 0 thru 4
- 9872B - 0 thru 4
- 9872C/S/T - 0 thru 8
- 7580A/7585A - 0 thru 8
- 7470A - 0 thru 2

Error Conditions

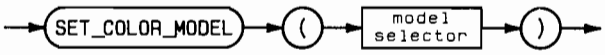
The graphics system must be initialized and a display must be enabled or this call will be ignored, an ESCAPE (– 27) will be generated, and GRAPHICSERROR will return a non-zero value.

SET_COLOR_MODEL

IMPORT: dgl_lib

This **procedure** chooses the color model for interpreting parameters in the color table.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
model selector	Expression of TYPE INTEGER	MININT thru MAXINT	0 or 1

Procedure Heading
PROCEDURE SET_COLOR_MODEL (MODEL : integer);

Semantics

The **model selector** determines the color model which will be used to interpret the values passed to the color table with SET_COLOR_TABLE or read from it with INQ_COLOR_TABLE.

Value	Meaning
1	RGB (Red-Green-Blue) color cube.
2	HSL (Hue-Saturation-Luminosity) color cylinder.

The RGB physical model is a color cube with the primary additive colors (red, green, and blue) as its axes. With this model, a call to SET_COLOR_TABLE specifies a point within the color cube that has a red intensity value (X-coordinate), a green intensity value (Y-coordinate) and a blue intensity value (Z-coordinate). Each value ranges from zero (no intensity) to one.

Effects of RGB color parameters

Parm 1 (RED)	Parm 2 (GREEN)	Parm 3 (BLUE)	Resultant color
1.0	1.0	1.0	White
1.0	0.0	0.0	Red
1.0	1.0	0.0	Yellow
0.0	1.0	0.0	Green
0.0	1.0	1.0	Cyan
0.0	0.0	1.0	Blue
1.0	0.0	1.0	Magenta
0.0	0.0	0.0	Black

The HSL perceptual model is a color cylinder in which:

- The angle about the axis of the cylinder, in fractions of a circle is the hue (red is at 0, green is at 1/3 and blue is at 2/3).
- The radius is the saturation. Along the center axis of the cylinder, (saturation equal zero) the colors range from white through grey to black. Along the outside of the cylinder (saturation equal one) the colors are saturated with no apparent whiteness.
- The height along the center axis is the luminosity (the intensity or brightness per unit area). Black is at the bottom of the cylinder (luminosity equal zero) and the brightest colors are at the top of the cylinder (luminosity equal one) with white at the center top.

Hue (angle), saturation (radius), and luminosity (height) all range from zero to one. Using this model, a call to SET_COLOR_TABLE specifies a point within the color cylinder that has a hue value, a saturation value, and a luminosity value.

Effects of HSL color parameters

Parm 1 (Hue)	Parm 2 (Sat)	Parm 3 (Lum)	Resultant color
Don't Care	0.0	1.0	White
0.0	1.0	1.0	Red
1/6	1.0	1.0	Yellow
2/6	1.0	1.0	Green
3/6	1.0	1.0	Cyan
4/6	1.0	1.0	Blue
5/6	1.0	1.0	Magenta
Don't Care	Don't Care	0.0	Black

When a call to SET_COLOR_MODEL switches color models, parameter values in subsequent calls to SET_COLOR_TABLE then refer to the new model. Switching models does not affect color definitions that were previously made using another model. Note that when the value of a color table entry is inquired (INQ_COLOR_TABLE), it is returned in the current model, which may not be the model in which it was originally specified.

Not all color specifications can be displayed on every graphics device, since the devices which the graphics library supports differ in their capabilities. If color specification is not available on a device, the graphics system will request the closest available color.

Error Conditions

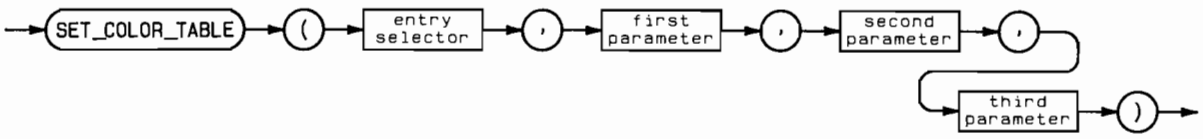
The graphics system must be initialized and the color selector must evaluate to 0 or 1 or this call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

SET_COLOR_TABLE

IMPORT: dgl_lib

This **procedure** redefines the color description of the specified entry in the color table. This color definition is used when the color index is selected via SET_COLOR.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
entry selector	Expression of TYPE INTEGER	MININT to MAXINT	device dependent (see below)
first parameter	Expression of TYPE REAL	0 thru 1	–
second parameter	Expression of TYPE REAL	0 thru 1	–
third parameter	Expression of TYPE REAL	0 thru 1	–

Procedure Heading

```
PROCEDURE SET_COLOR_TABLE ( Index : INTEGER;
                            Colr1 : REAL;
                            Colr2 : REAL;
                            Colr3 : REAL    );
```

Semantics

SET_COLOR_TABLE is ignored by some devices (such as pen plotters) which do not allow their color table to be changed. The procedure INQ_WS (opcode 1073) tells whether the color table can be changed.

The **entry selector** specifies the location in the color capability table that is to be redefined. For raster displays in Series 200 computers, 32 entries are available.

The **first parameter** represents red intensity if the RGB model has been selected with the SET COLOR statement, or hue if the HSL model has been selected.

The **second parameter** represents green intensity if the RGB model has been selected with the SET COLOR statement, or saturation if the HSL model has been selected.

The **third parameter** represents blue intensity if the RGB model has been selected, or luminosity if the HSL model has been selected.

A more detailed description of the color models and the meaning of their parameters can be found under the procedure definition of SET_COLOR_MODEL.

The effect of redefinition of the color table on previously output primitives is device dependent. On most devices changing the color table will only affect future primitives; however, on the Model 36C changing a color table entry with a color selector from 0 through 15 will immediately change the color of primitives previously drawn with that entry. The procedure INQ_WS (opcode 1071) tells whether retroactive color change is supported.

Monochromatic Displays
All Series 200 computers except the Model 36C have a monochromatic internal CRT. Changing an entry in the table will not affect the current display; however, future changes to the display will use the new contents of the table. Device dependent polygons use the color table entry when performing dithering.

The color that lines are drawn with (black or white) is determined from the perceived intensity of the color table entry. This is calculated as follows:

```
if (red * 0.3 + green * 0.59 + blue * 0.11) > 0.1
    then
      color := white
    else
      color := black;
```

The HP 98627A Display
Changing an entry in the table will not affect the current display; however, future changes to the display will use the new contents of the table. Device dependent polygons use the color table entry when performing dithering.

The color that lines are drawn with (one of the 8 non-dithered colors) is determined from the closest HSL value to the requested value.

The Model 36C
The first 16 locations (0..15) of the color table map directly to the hardware color map. Changing one of these color table locations will immediately change the display (assuming the color has been used).

The next 16 locations (16..31) will not affect the current display; however, future changes to the display will use the new contents of the color table.

Device dependent polygons drawn with color table locations 0..15 will be drawn in a solid color without using dithering. When drawn with color table location above 15 dithering will be used.

Note

Since dithering on the HP 9836C uses the current color map values (i.e., color table locations 0..15) changing the first 16 color table locations will affect the dither pattern used. This leads to two major effects. First, changing the first 16 locations after a polygon was generated using dithering will change the dither pattern such that its averaged color no longer matches the color that it was generated with. Second, since the dither pattern is based on the first 16 colors, the first 16 colors can be set to produce a dither pattern with minimum color changes between pixels within the pattern. The following example produces a continuous shaded polygon across the crt:

```
$RANGE OFF$
PROGRAM T;

IMPORT dgl_types, dgl_lib, dgl_poly;

VAR I          : INTEGER;
    Xvec,Yvec  : ARRAY [1..2] OF REAL;
    Ovec       : ARRAY [1..2] OF Gshortint;
    C          : REAL;

BEGIN
  GRAPHICS_INIT;
  DISPLAY_INIT(3,0,i);
  SET_ASPECT(511,389);
  SET_WINDOW(0,511,0,389);

FOR I := 0 to 15 DO
SET_COLOR_TABLE(I,I/15,I/15,I/15);  { set up color map }

SET_PGN_COLOR ( 16 );
SET_PGN_STYLE ( 16 );

Yvec[1] := 100; Yvec[2] := 150; Ovec[1] := 0; Ovec[2] := 0;
FOR I := 0 to 511 DO
BEGIN
  Xvec[1] := I; Xvec[2] := I;
  C := I/511;
  SET_COLOR_TABLE(16,C,C,C); { set polygon color }
  POLYGON_DEV_DEP(2,Xvec,Yvec,Ovec);
 END;
END.
```

The color that lines are drawn with (one of the first 16 non-dithered colors) is determined from the closest HSL value to the requested value.

Dithered Polygon Fills

All the raster displays use a technique called dithering for filling device dependent polygons. The polygon is divided into 4 pixel by 4 pixel 'dither cells'. The colors that are placed in each pixel location inside the dither cells average to the current polygon color. The eye will average the pixels, and see the intended color.

The 98627A has 3 memory planes thus, providing 8 non-dithered colors (white, red, green, blue, cyan, magenta, and black). Using dithering 4913 polygon colors may be generated. To obtain a polygon color of half-tone yellow (R = 0.5 G = .0.5 B = 0.0) the dither cell would contain 8 black pixels and 8 yellow pixels.

On black and white displays, the largest r,g,b value of the current_polygon color is used to determine the dither pattern.

On the HP 9836C the current values of the color map are used to determine the dither cell pixel colors. This leads to a very very large number of colors that the HP 9836C can produce when performing device dependent polygon fill.

The Background Color

Color index 0 represents the background color. The ability to redefine this index is device-dependent. Many devices do not allow the redefinition of their background color. Whether a display device has the ability to redefine the background color can be inquired via a call to INQ_WS with opcode = 1072. All raster displays in the 200 Series are capable of redefining the background color.

Error Conditions

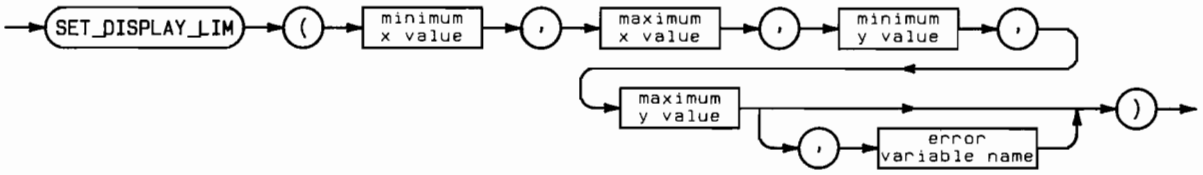
The graphics system must be initialized and a display device must be enabled or this call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

SET_DISPLAY_LIM

IMPORT: dgl_lib

This **procedure** redefines the logical display limits of the graphics display.

Syntax



Item	Description/Default	Range Restrictions
minimum x value	Expression of TYPE REAL	–
maximum x value	Expression of TYPE REAL	–
minimum y value	Expression of TYPE REAL	–
maximum y value	Expression of TYPE REAL	–
error variable name	Variable of TYPE INTEGER	–

Procedure Heading

```
PROCEDURE SET_DISPLAY_LIM (      Xmin, Xmax,
                                 Ymin, Ymax : REAL,
                           VAR           Ierr : INTEGER );
```

Semantics

The **minimum x value** is the distance in millimetres that the left side of the logical display limits is offset from the left side of the physical display limits.

The **maximum x value** is the distance in millimetres that the right side of the logical display limits is offset from the left side of the physical display limits.

The **minimum y value** is the distance in millimetres that the bottom of the logical display limits is offset from the bottom of the physical display limits.

The **maximum y value** is the distance in millimetres that the top of the logical display limits is offset from the bottom of the physical display limits.

The **error variable** will contain an integer indicating whether the limits were successfully set.

Value	Meaning
0	The display limits were successfully set.
1	The minimum x value was greater than or equal to the maximum x value and/or the minimum y value was greater than the maximum y value.
2	The parameters specified were outside the physical display limits.

If the error variable is non-zero, the call was ignored.

SET_DISPLAY_LIM allows an application program to specify the region of the display surface where the image will be displayed. The limits of this region are defined as the logical display limits. Upon initialization, the graphics system sets these limits equal to some portion of the specified physical device. This routine allows a programmer to set the plotting surface of a very large plotter equal to the size of an 8 1/2 x 11 inch paper, for example.

The pairs (minimum x value, minimum y value) and (maximum x value, maximum y value) define the corner points of the new logical display limits in terms of millimetres offset from the origin of the physical display. The exact position of the physical display origin is device dependent. The specifics of various devices are covered later in this entry.

This procedure causes a new virtual coordinate system to be defined. SET_DISPLAY_LIM calculates the new limits of the virtual coordinate system as a function of the current aspect ratio and the new limits of the logical display. This does not affect the limits of the viewport. Since it changes the size of the area onto which the viewport is mapped, it may scale the size of the image displayed. It will not distort the image; it can only make it smaller or larger.

SET_DISPLAY_LIM should only be called while the graphics display is enabled.

Neither the value of the starting position nor the location of the physical pen or beam is altered by this routine. Since this routine may redefine the viewing transformation, the starting position may be mapped to a different coordinate on the display surface. A call to MOVE or INT_MOVE should therefore be made after this call to update the value of the starting position and in so doing, place the physical pen or beam at a known location.

If the logical display and logical locator are associated with the same physical device, a call to SET_DISPLAY_LIM will set the logical locator limits equal to the new limits of the virtual coordinate system. A call to SET_DISPLAY_LIM also sets the locator echo position to its default value, the center of the world coordinate system.

Display Limits of Raster Devices

The internal CRT's for Series 200 computers have the following limits:

Plotter	Wide mm	High mm	Wide points	High points	Aspect	Resolution mm
9816/ 9920	168	126	400	300	.75	2.381
9826	120	90	400	300	.75	3.333
9836	210	160	512	390	.7617	2.438
9836C	217	163	512	390	.7617	2.39

The physical size of the HP 98627A display (needed by the SET_DISPLAY_LIM procedure) may be given to the graphics system by an escape function (OPCODE = 250). The physical limits assumed until the escape function is given are:

CONTROL	=	256	153.3mm wide and 116.7mm high.
		512	153.3mm wide and 116.7mm high.
		768	153.3mm wide and 142.2mm high.
		1024	153.3mm wide and 153.3mm high.
		1280	153.3mm wide and 153.3mm high.

The default logical display surface of the graphics display device is the maximum physical limits of the screen. The physical origin is the lower left corner of the display.

The view surface is always centered within the current logical display surface. The origin of a raster display is the lower-left dot.

HPGL Plotter Display Limits

Plotter	Wide mm	High mm	Wide points	High points	Aspect	Resolution points/mm
9872	400	285	16000	11400	.7125	40.0
7580	809.5	524.25	32380	20970	.6476	40.0
7585	1100	890	44000	35670	.809	40.0
7470	257.5	190	10300	7600	.7378	40.0

The maximum physical limits of the graphics display for a HPGL device not listed above are determined by the default settings of P1 and P2. The default settings of P1 and P2 are the values they have after an HPGL 'IN' command. Refer to the specific device manual for additional details.

The default logical display surface is set equal to the area defined by P1 and P2 at the time DISPLAY_INIT is invoked. The view-surface is always justified in the lower left corner of the current logical display surface (corner nearest the turret for the HP 7580 and HP 7585 plotters). The physical origin of the graphics display is at the lower left boundary of pen movement.

Note

If the paper is changed in an HP 7580 or HP 7585 plotter while the graphics display is initialized, it should be the same size of paper that was in the plotter when DISPLAY_INIT was called. If a different size of paper is required, the device should be terminated (DISPLAY_TERM) and re-initialized after the new paper has been placed in the plotter.

Spooling on an HP 7585A plotter can only be done on "D" size drafting paper.

Error Conditions

The graphics system must be initialized and a display device enabled or this call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

SET_ECHO_POS

IMPORT: dgl_lib

This **procedure** defines the locator echo position on the graphics display.

Syntax

Item	Description/Default	Range Restrictions
x coordinate	Expression of TYPE REAL	–
y coordinate	Expression of TYPE REAL	–

Procedure Heading

```
PROCEDURE SET_ECHO_POS ( Wx, Wy : REAL );
```

Semantics

The **x** and **y coordinate** pair is the new echo position in world coordinates.

When echoing on the display device, SET_ECHO_POS allows a programmer to define the position of the locator echo position. This is a point in the world coordinate system that represents the initial position of the locator. It is used with certain locator echoes on the graphics display. For example, it is used as the anchor point when a rubber band echo is performed. With this echo, the graphics cursor is initially turned on at the locator echo position. From that time on, the cursor reflects the position of the locator and a line extends from the locator echo position to the locator as it moves around the graphics display. To be used in echoing, the point must be displayable. Therefore, if the point specified is outside of the limits of the window the call is ignored.

The locator echo position will only be used when AWAIT_LOCATOR is called with echo types 2 through 8, e.g., type 4 is a rubber band line echo. The locator echo position is only used when the locator echo is being sent to the graphics display device, and is not used when sampling the locator.

SET_ECHO_POS should only be called while the graphics display and locator are initialized. If the point passed to SET_ECHO_POS is outside the current window limits, then the call to SET_ECHO_POS is ignored and no error is given.

The default locator echo position is the center of the limits of the window. When the locator is initialized, the locator echo position is set to the default value. When a call is made which affects the viewing transformations for the graphics display surface or the logical locator limits, the locator echo position is set to the default value. The calls which cause this are SET_ASPECT, DISPLAY_INIT, SET_DISPLAY_LIM, LOCATOR_INIT, SET_LOCATOR_LIM, SET_WINDOW, and SET_VIEWPORT.

Once the locator echo position is set, it retains this value until the next call to SET_ECHO_POS or until a call is made which resets it to the default value.

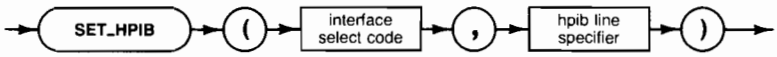
Error Conditions
The graphics system must be initialized, and a display device and a locator device must be enabled, or this call will be ignored, an ESCAPE (– 27) will be generated, and GRAPHICSERROR will return a non-zero value.

SET_HPIB

IMPORT: hpib_0
iodeclarations

This **procedure** will set the specified HP-IB control line. Not all HP-IB lines are acessible at all times.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
hpib line specifier	Expression of enumerated TYPE *hpib_line*.	atn_line dav_line ndac_line nrfd_line eoi_line srq_line ifc_line ren_line	

Semantics

All possible hpib_line types are not legal when using this procedure. Handshake lines (DAV, NDAC, NRFD) are never accessible, and an error is generated if an attempt is made to set them.

The Service Request line (SRQ) is not accessible and should be set with REQUEST_SERVICE.

Setting the Interface Clear line (IFC) and the Remote Enable line (REN) requires the system to be system controller.

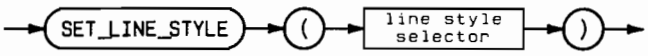
Setting the Attention line (ATN) requires the interface to be active controller.

SET_LINE_STYLE

IMPORT: dgl_lib

This **procedure** sets the line style attribute.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
line style selector	Expression of TYPE INTEGER	MININT thru MAXINT	Device Dependent

Procedure Heading

```
PROCEDURE SET_LINE_STYLE (Line_Style : INTEGER);
```

Semantics

The **line style selector** is the line style to be used for lines, polylines, polygon edges, and text.

Markers are not affected by line-style. Polygon interior line-style is selected with SET_PGN_LS.

SET_LINE_STYLE sets the line style attribute for lines and text. The mapping between the value of the line style attribute and the line style selected is device dependent. If a line style attribute is requested that the device cannot perform exactly as requested, line style 1 will be performed.

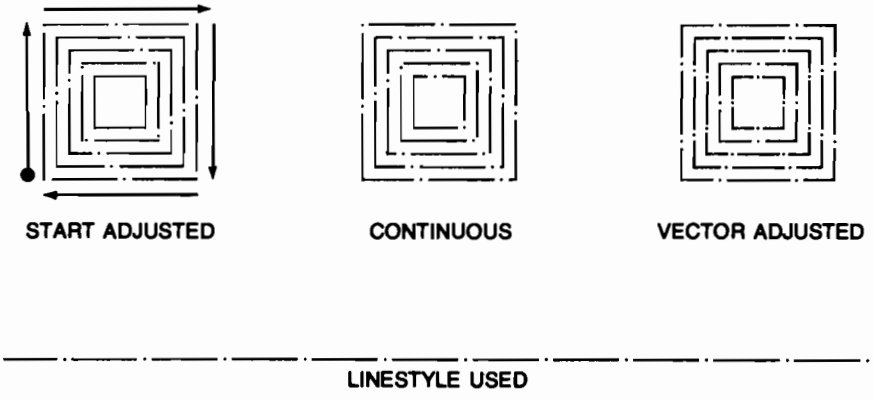
There are three types of line-styles: start adjusted, continuous, and vector adjusted:

Start adjusted line-styles always start the cycle at the beginning of the vector. Thus if the current line-style starts with a pattern, each vector drawn will start with that pattern. Likewise, if the current line-style starts with a space and then a dot, each vector will be drawn starting with a space and then a dot. In this case if the vectors are short, they might not appear at all.

Continuous line styles are generated such that the pattern will be started with the first vector drawn. Subsequent vectors will be continuations of the pattern. Thus, it may take several vectors to complete one cycle of the pattern. This type of line-style is useful for drawing smooth curves, but does not necessarily designate either endpoint of a vector. A side effect of this type of line-style is if a vector is small enough it might be composed only of the space between points or dashes in the line-style. In that case, the vector may not appear on the graphics display at all.

Vector adjusted line-styles treat each vector individually. Individual treatment guarantees that a solid component of the dash pattern will be generated at both ends of the vector. Thus, the endpoints of each vector will be clearly identifiable. This type of line-style is good for drawing rectangles. The integrity of the line-style will degenerate with very small vectors. Since some component of the dash pattern must appear at both ends of the vector, the entire vector for a short vector will often be drawn as solid.

The following figure illustrates how one pattern would be displayed using each one of the different line-style types:



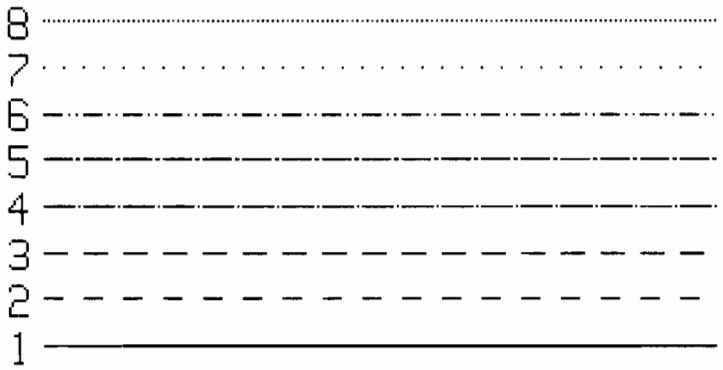
It should be apparent from the above discussion that drawing to the starting position will generate a point (the shortest possible line) only if the line-style is such that the pen is down (or the beam is on) at the start of that vector. Likewise, whole vectors may not appear on the graphics display surface if the line-style is such that the vector is smaller than the blank space in the line-style. The device handlers section of this document details the line-styles available for each device.

Note

When using continuous line styles, complement and erase drawing modes (available on some raster displays e.g., HP 9826) may not completely remove lines previously drawn. This happens since the line style pattern may not be in sync with the first line when the second line is drawn. By setting the line-style to solid when using complement and erase drawing modes the application program can insure that the line is completely removed.

Raster Line Styles

Eight pre-defined line-styles are supported on the graphics display. All of the line-styles may be classified as being "continuous":

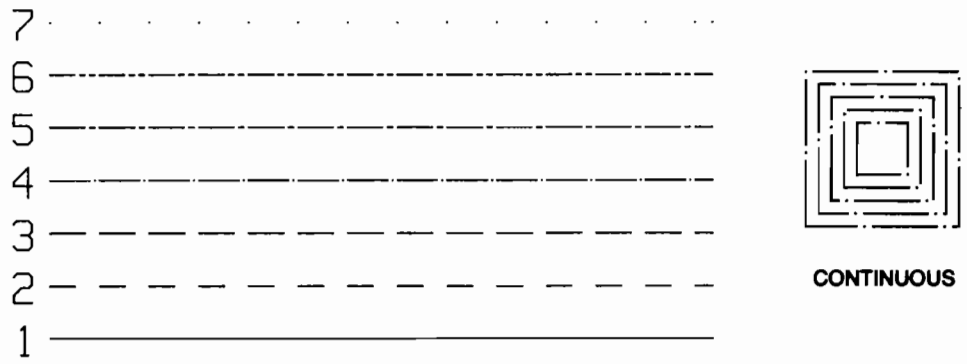


Raster Line Styles

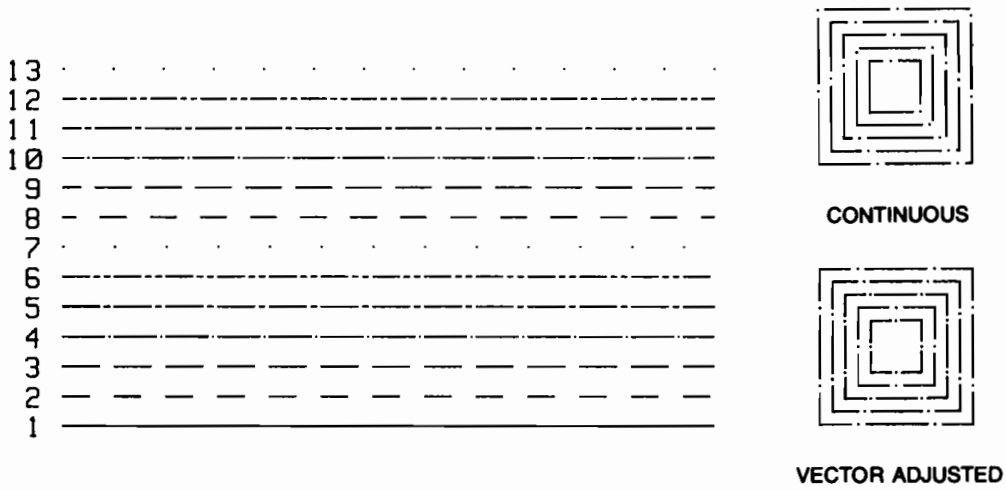
Plotter Line Styles

The following table describes the line styles available on the supported plotters.

Device	Number of continuous line-styles	Number of vector adjusted line-styles
9872	7	0
7580	7	6
7585	7	6
7470	7	0
Other	7	0



HP 9872 and 7470 Line Styles (all are continuous)



HP 7580 and 7585 Line Styles

If the line style specified is not supported by the graphics display, the call is completed with LINE_STYLE = 1 and no error is reported.

Error Conditions

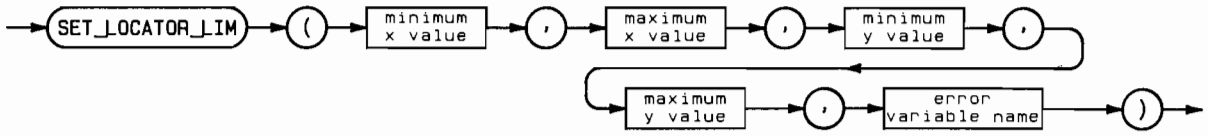
The graphics system must be enabled and a display device must be enabled or this call will be ignored and GRAPHICSERROR will return a non-zero value.

SET_LOCATOR_LIM

IMPORT: dgl_lib

This **procedure** redefines the logical locator limits of the graphics locator.

Syntax



Item	Description/Default	Range Restrictions
minimum x value	Expression of TYPE REAL	–
maximum x value	Expression of TYPE REAL	–
maximum y value	Expression of TYPE REAL	–
minimum y value	Expression of TYPE REAL	–
error variable name	Variable of TYPE INTEGER	–

Procedure Heading

```
PROCEDURE SET_LOCATOR_LIM (      Xmin, Xmax,
                                 Ymin, Ymax : REAL,
                           VAR Ierr         : INTEGER );
```

Semantics

The **minimum x value** is the distance in millimetres that the left side of the logical locator limits is offset from the left side of the physical locator limits.

The **maximum x value** is the distance in millimetres that the right side of the logical locator limits is offset from the left side of the physical locator limits.

The **minimum y value** is the distance in millimetres that the bottom of the logical locator limits is offset from the bottom of the physical locator limits.

The **maximum y value** is the distance in millimetres that the top of the logical locator limits is offset from the bottom of the physical locator limits.

The **error variable** will contain an integer indicating whether the limits were successfully set.

Value	Meaning
0	The display limits were successfully set.
1	The minimum x value was greater than or equal to the maximum x value and/or the minimum y value was greater than the maximum y value.
2	The parameters specified were outside the physical display limits.
3	Attempt to explicitly define locator limits on a device which is both the logical locator and the logical display. The logical display limits are used when a device is shared for both purposes, and they cannot be redefined with this call.

If the error variable is non-zero, the call was ignored.

SET_LOCATOR_LIM allows an application program to specify the portion of the physical locator device that should be used to perform locator functions. When the logical locator device is enabled (via LOCATOR_INIT) the logical device limits are set to a device dependent portion of the physical locator device. With a call to this routine the user can set the logical locator limits by specifying a new area within the physical locator limits.

The pairs (minimum x value, minimum y value) and (maximum x value, maximum y value) define the corner points of the new logical locator limits in terms of millimetres offset from the origin of the physical locator. The exact position of the physical locator origin is device dependent. Specific origins are covered later in this entry.

If a logical locator and a logical display are associated with the same physical device, then the logical locator limits must be the same as the logical view surface limits. Specifically, the effects of the association with the same physical device are as follows:

- The logical locator limits are initialized to the same values as the virtual coordinate system.
- Any call which redefines the virtual coordinate system limits will also redefine the logical locator limits.
- The logical locator limits can not be defined by a call to SET_LOCATOR_LIM.

By changing the logical locator limits any portion of the graphics locator can be addressed, with the restrictions stated above.

The logical locator limits always map directly to the view surface, therefore, distortion may result in the mapping between the logical locator and the display when the logical locator limits and the view surface have different aspect ratios. If the distortion is not desired it can be avoided by assuring that the logical locator limits maintain the same aspect ratio as that of the view surface.

SET_LOCATOR_LIM should only be called while the graphics locator is enabled. SET_LOCATOR_LIM sets the locator echo position to the default value (see SET_ECHO_POS).

Locator Limits: The Knob

The knob may be used as a locator on Series 200 computers. The default characteristics of the knob on various Series 200 computers is listed in the table below.

Plotter	mm	Wide mm	High points	Wide points	High Aspect	Resolution mm
9816	168	126	400	300	.75	2.381
9826	120	90	400	300	.75	3.333
9836	210	160	512	390	.7617	2.438
9836C	217	163	512	390	.7617	2.39

The knob uses the current display limits as its locator limits for locator echoes 2 though 8. For all other echoes the above limits are used. An example of when the two limits may differ follows:

> The knob locator is initialized on an HP 9826. The graphics display is an HP 98627A color output card. The resolution of the locator is 0 through 399 in x dimension, and 0 through 299 in y dimension. The resolution of the display is 0 through 511 in x dimension, and 0 through 389 in y dimension. When await_locator is used with echo 4, the locator will effectively have the HP 98627A resolution for the duration of the await_locator call. However if echo 1 is used with await_locator, the cursor will appear on the HP 9826 and the locator has a resolution of 0×399 and 0×299. Note that all conversion routines, and inquiries will use the HP 9826 limits.

The physical origin of the locator device is the lower left corner of the display.

Locator Limits: HPGL Devices

HPGL devices can be used as locators. The default characteristics of some HPGL devices are listed below.

Device	Wide mm	High mm	Wide points	High points	Aspect	Resolution points/mm
9872	400	285	16000	11400	.7125	40.0
7580	809.5	524.25	32380	20970	.6476	40.0
7585	1100	890	44000	35670	.809	40.0
7470	257.5	190	10300	7600	.7378	40.0
9111	300.8	217.6	12032	8704	.7234	40.0

The maximum physical limits of the locator for a HPGL device not listed above are determined by the default settings of P1 and P2. The default settings of P1 and P2 are the values they have after an HPGL 'IN' command. Refer to the specific device manual for additional details.

The default logical display surface is set equal to the area defined by P1 and P2 at the time LOCATOR_INIT is invoked.

Note

If the paper is changed in an HP 7580 or HP 7585 plotter while the graphics locator is initialized, it should be the same size of paper that was in the plotter when LOCATOR_INIT was called. If a different size of paper is required, the device should be terminated (LOCATOR_TERM) and re-initialized after the new paper has been placed in the plotter.

Error Conditions

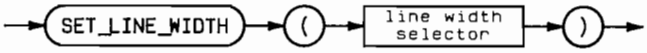
The graphics system must be initialized and a display device enabled or this call will be ignored, an ESCAPE (– 27) will be generated, and GRAPHICSERROR will return a non-zero value.

SET_LINE_WIDTH

IMPORT: dgl_lib

This **procedure** sets the line-width attribute. The number of line-widths possible is device dependent.

Syntax



Item	Description/Default	Range Restrictions
line-width selector	Expression of TYPE INTEGER	MININT thru MAXINT

Procedure Headings

```
PROCEDURE SET_LINE_WIDTH ( Linewidth : INTEGER );
```

Semantics

SET_LINE_WIDTH sets the line-width attribute for lines, polylines and text. The line-width attribute does not affect markers which are defined to be always output with the thinnest line-width supported on the device. All devices support at least one line-width. The range of line-widths is device dependent but line-width 1 is always the thinnest line-width supported. For devices that support multiple line-widths, the line-width increases as line-width does until the device supported maximum is reached. For example, line-width = 1 specifies the thinnest, line-width = 2 specifies the next wider line-width, etc.

If line-width is greater than the number of line-widths supported by the graphics display or line-width is less than 1, then the line-width will be set to the thinnest available width (line-width = 1). All subsequent lines and text will then be drawn with the thinnest available line-width. A call to INQ_WS with OPCODE equal to 1063 to inquire the value of the line-width will then return a 1.

The initial line-width is the thinnest width supported by the device (line-width = 1).

Note

All current devices support a single line-width.

Error Conditions

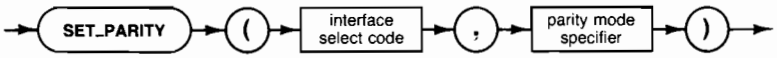
The graphics system must be initialized and a display device must be enabled or this call is ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

SET_PARITY

IMPORT: serial_3
iodeclarations

This **procedure** determines what parity mode the serial interface will use.

Syntax



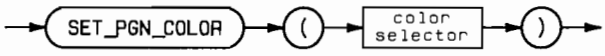
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
parity mode specifier	Expression of enumerated TYPE *type_parity*.	no_parity odd_parity even_parity one_parity zero_parity	

SET_PGN_COLOR

IMPORT: dgl_lib
dgl_poly

This **procedure** selects the polygon interior color attribute for subsequently generated polygons by providing a selector for the color table.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
color selector	Expression of TYPE INTEGER	MININT thru MAXINT	Device dependent.

Procedure Heading

```
PROCEDURE SET_PGN_COLOR ( Cindex : INTEGER );
```

Semantics

The **color selector** is an index into the color table. The contents of the color table are then used to specify the color when primitives are drawn. On some devices (HPGL plotters), the color selector maps directly to a pen number for the device. On the HP 9836C, the entries in the color table can be modified with SET_COLOR_TABLE. The color actually used depends on the value in a device dependent color table.

At device initialization a default color table is created by the graphics system. The size and contents of the table are device dependent. At least one entry exists for all devices. A call to INQ_WS with OPCODE equal to 1053 will return the number of colors available on a given graphics device. Some devices allow the color table to be modified with SET_TABLE.

The default value of the color attribute is 1. If the value of the color selector is not supported on the graphics display, the color attribute will be set to 1.

A color selector of 0 has special effects depending on the graphics display used. For raster devices, a color selector of 0 means to draw in the background color. For most plotters, it puts the pen away.

Dithering

If the device is not capable of reproducing a color in the color table, the closest color which the device is capable of reproducing is used instead. For polygon fill (in a device dependent mode) this may involve dithering. For example, the HP 98627A color output interface card is capable of a large selection of polygon fill colors, but only 8 line colors. Thus, the fill color could match the selected color much more closely than the line color used to outline the polygon. See SET_COLOR_TABLE for details on how colors are matched to the devices.

Default Raster Color Map

The following table shows the default (initial) color table for the black and white displays (HP 9816 / HP 9920 / HP 9826 / HP 9836):

Index #	Hue	Saturation	Luminosity
0	0	0	0
1	0	0	1.0000
2	0	0	0.9375
3	0	0	0.8750
4	0	0	0.8125
5	0	0	0.7500
6	0	0	0.6875
7	0	0	0.6250
8	0	0	0.5625
9	0	0	0.5000
10	0	0	0.4375
11	0	0	0.3750
12	0	0	0.3125
13	0	0	0.2500
14	0	0	0.1875
15	0	0	0.1250
16	0	0	0.0625

Colors 17 though 31 are set to white.

The following table shows the default (initial) color table for the color displays (HP 9836C and HP 98627A):

Index #	Color name	Red	Green	Blue
0	Black	0.000000	0.000000	0.000000
1	White	1.000000	1.000000	1.000000
2	Red	1.000000	0.000000	0.000000
3	Yellow	1.000000	1.000000	0.000000
4	Green	0.000000	1.000000	0.000000
5	Cyan	0.000000	1.000000	1.000000
6	Blue	0.000000	0.000000	1.000000
7	Magenta	1.000000	0.000000	1.000000
8	Black	0.000000	0.000000	0.000000
9	Olive green	0.800000	0.733333	0.200000
10	Aqua	0.200000	0.400000	0.466667
11	Royal blue	0.533333	0.400000	0.666667
12	Violet	0.800000	0.266667	0.400000
13	Brick red	1.000000	0.400000	0.200000
14	Burnt orange	1.000000	0.466667	0.000000
15	Grey brown	0.866667	0.533333	0.266667

Colors 9 though 15 are a graphic designers idea of colors for business graphics. Color table entries not shown above are set to white.

Raster Drawing Modes
Raster drawing modes have no effect on polygon fill color.

Plotters
A Color Selector of 0 selects no pens (the current pen is put away). The supported range of Color Selectors for each supported plotter is:

- 9872A - 0 thru 4
- 9872B - 0 thru 4
- 9872C/S/T - 0 thru 8
- 7580A/7585A - 0 thru 8
- 7470A - 0 thru 2

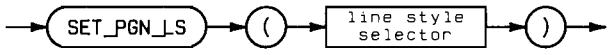
Error Conditions
The graphics system must be initialized and a display must be enabled or this call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR returns a non-zero value.

SET_PGN_LS

IMPORT: dgl_lib
dgl_poly

This **procedure** selects the polygon interior line-style attribute for subsequently generated polygons by providing a selector for the device dependent line-style table.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
line-style selector	Expression of TYPE INTEGER	MININT thru MAXINT	Device dependent

Procedure Heading

```
PROCEDURE SET_PGN_LS ( Lindex : INTEGER );
```

Semantics

The **line style selector** is the line style to be used for polygon interiors.

Line-styles for other primitives are selected using SET_LINE_STYLE.

The mapping between the value of the line style attribute and the line style selected is device dependent. If a line style attribute is requested that the device cannot perform exactly as requested, line style 1 will be performed.

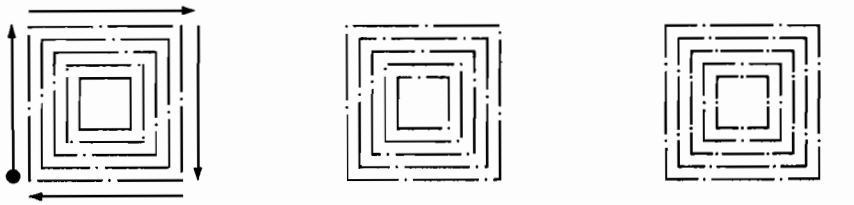
There are three types of line-styles - start adjusted, continuous, and vector adjusted:

Start adjusted line-styles always start the cycle at the beginning of the vector. Thus if the current line-style starts with a pattern, each vector drawn will start with that pattern. Likewise, if the current line-style starts with a space and then a dot, each vector will be drawn starting with a space and then a dot. In this case if the vectors are short, they might not appear at all.

Continuous line styles are generated such that the pattern will be started with the first vector drawn. Subsequent vectors will be continuations of the pattern. Thus, it may take several vectors to complete one cycle of the pattern. This type of line-style is useful for drawing smooth curves, but does not necessarily designate either endpoint of a vector. A side effect of this type of line-style is if a vector is small enough it might be composed only of the space between points or dashes in the line-style. In that case, the vector may not appear on the graphics display at all.

Vector adjusted line-styles treat each vector individually. Individual treatment guarantees that a solid component of the dash pattern will be generated at both ends of the vector. Thus, the endpoints of each vector will be clearly identifiable. This type of line-style is good for drawing rectangles. The integrity of the line-style will degenerate with very small vectors. Since some component of the dash pattern must appear at both ends of the vector, the entire vector for a short vector will often be drawn as solid.

The following figure illustrates how one pattern would be displayed using each one of the different line-style types:



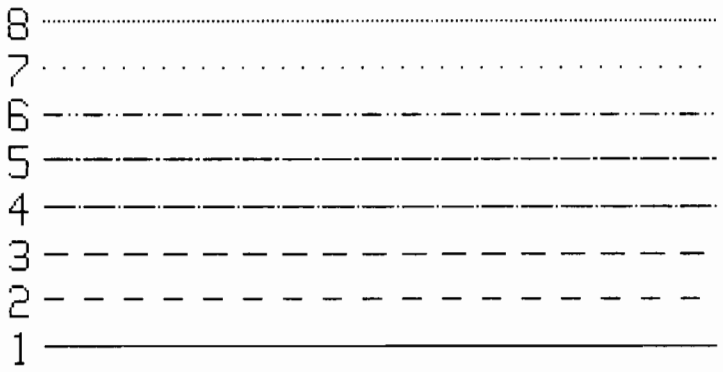
It should be apparent from the above discussion that drawing to the starting position will generate a point (the shortest possible line) only if the line-style is such that the pen is down (or the beam is on) at the start of that vector. Likewise, whole vectors may not appear on the graphics display surface if the line-style is such that the vector is smaller than the blank space in the line-style. The device handlers section of this document details the line-styles available for each device.

Note

When using continuous line styles, complement and erase drawing modes (available on some raster displays e.g., HP 9826) may not completely remove lines previously drawn. This happens since the line style pattern may not be in sync with the first line when the second line is drawn. By setting the line style to solid when using complement and erase drawing modes the application program can insure that the line is completely removed.

Raster Line Styles

Eight pre-defined line-styles are supported on the graphics display. All of the line-styles may be classified as being "continuous":

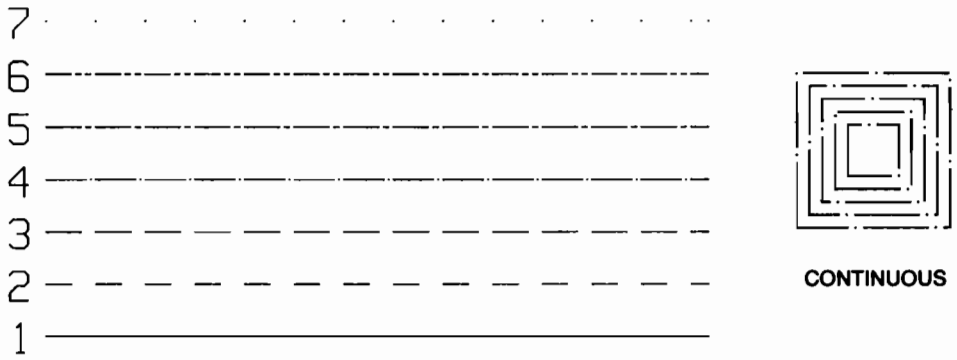


Raster Line Styles

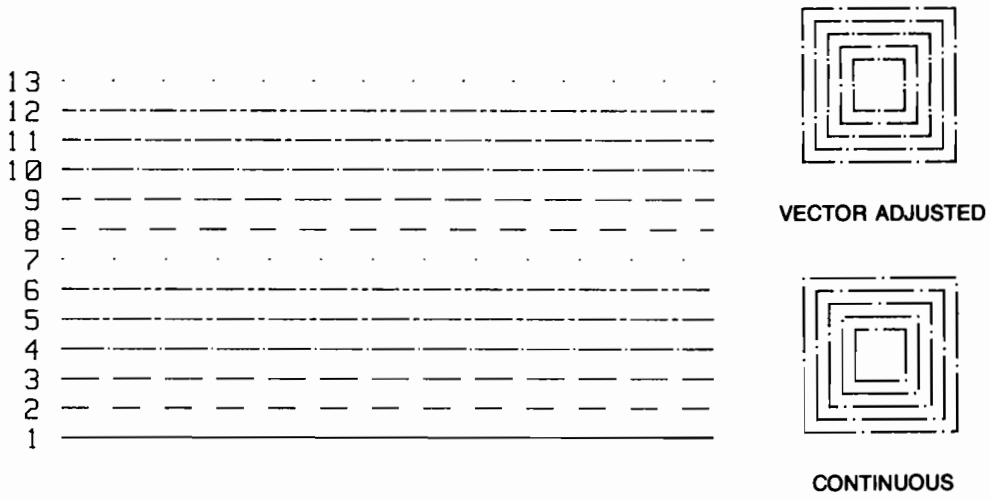
Plotter Line Styles

The following table describes the line styles available on the supported plotters.

Device	Number of continuous line-styles	Number of vector adjusted line-styles
9872	7	0
7580	7	6
7585	7	6
7470	7	0
Other	7	0



HP 9872 and 7470 Line Styles (all are continuous)



HP 7580 and 7585 Line Styles

If the line style specified is not supported by the graphics display, the call is completed with LINE_STYLE = 1 and no error is reported.

The graphics system must be enabled and a display device must be enabled or this call will be ignored and GRAPHICSERROR will return a non-zero value.

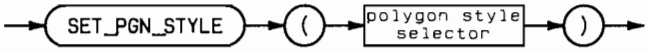
Error conditions:
The graphics system must be initialized and a display device must be enabled or this call will be ignored, an ESCAPE (– 27) will be generated, and GRAPHICSERROR will return an non-zero value.

SET_PGN_STYLE

IMPORT: dgl_lib
dgl_poly

This **procedure** selects the polygon style attribute for subsequently generated polygons by providing a selector for the polygon style table.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
polygon style selector	Expression of TYPE INTEGER	MININT thru MAXINT	Device dependent

Procedure Heading

```
PROCEDURE SET_PGN_STYLE ( Pindex : INTEGER );
```

Semantics

Polygon styles can vary in polygon interior density, polygon interior orientation and polygon edge display. See SET_PGN_TABLE for details on default styles, and how the polygon style table may be changed.

Error Conditions

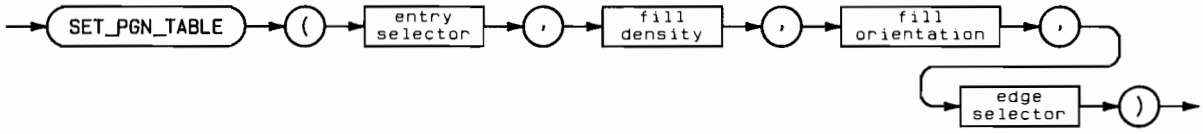
The graphics system must be initialized and a display device must be enabled or this call will be ignored and GRAPHICSERROR will return an non-zero value.

SET_PGN_TABLE

IMPORT: dgl_lib
dgl_poly

This **procedure** defines a polygon style attribute, i.e. an entry in a polygon style table.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
entry selector	Expression of TYPE INTEGER	MININT thru MAXINT	Device dependent
fill density	Expression of TYPE REAL	MININT thru MAXINT	-1 thru 1
fill orientation	Expression of TYPE REAL	MININT thru MAXINT	-90 thru 90
edge selector	Expression of TYPE INTEGER	MININT thru MAXINT	–

Procedure Heading

```
PROCEDURE SET_PGN_TABLE ( Index  : INTEGER;
                          Densty : REAL;
                          Orient : REAL;
                          Edge   : INTEGER );
```

Semantics

This routine defines the attribute of polygon style, i.e. it specifies an entry in a polygon style table. This entry contains information that specifies polygon interior density, polygon interior orientation, polygon edge display, and device-independence of polygon display.

The **entry selector** specifies the entry in the polygon style table that is to be redefined.

The **fill density** determines the density of the polygon interior fill. The magnitude of this value is the ratio of filled area to non-filled area. Zero means the polygon interior is not filled. One represents a fully filled polygon interior. All non-zero values specify the density of continuous lines used to fill the interior.

Positive density values request parallel fill lines in one direction only. Negative values are used to specify crosshatching. For a given density, the distance between two adjacent parallel lines is greater with cross hatching than in the case of pure parallel filling. Calculations for fill density are based on the thinnest line possible on the device and on continuous line-style.

The distance between fill lines – hence density – does not change with a change of scale caused by a viewing transformation. If the interior line-style is not continuous, the actual fill density may not match that found in the polygon style table.

The **fill orientation** represents the angle (in degrees) between the lines used for filling the polygon and the horizontal axis of the display device. The interpretation of fill orientation is device-dependent. On devices that require software emulation of polygon styles, the angle specified will be adhered to as closely as possible, within the line-drawing capabilities of the device. For hardware generated polygon styles, the angle specified will be adhered to as closely as is possible given the hardware simulation of the requested density. If crosshatching is specified, the fill orientation specifies the angle of orientation of the first set of lines in the crosshatching, and the second set of lines is always perpendicular to this.

The value of the **edge selector** determines whether the edge of the polygon is displayed. If the edge selector is 0, the edges will not be displayed. If the edge selector is 1, display of individual edge segments depends on the operation selector in the call that draws the polygon set, POLYGON, INT_POLYGON, POLYGON_DEV_DEP, or INT_POLYGON_DD.

If polygon edges are displayed, they adhere to the current line attributes of color, line-style, and line-width, in effect at the time of polygon display.

A device-dependent number of polygon styles are available. All devices support at least 16 entries in the polygon table. The polygon styles defined in the default tables are defined to exploit the hardware capabilities of the devices they are defined for.

Polygon interiors can be generated in either a device-dependent or device-independent fashion, by calling POLYGON_DEV_DEP or POLYGON respectively.

Polygons generated in a device-dependent fashion will utilize the available hardware polygon generation capabilities of the device to increase the speed and efficiency of polygon generation. The output may vary depending on the device. Devices that have no hardware polygon generation capabilities will only do a minimal representation of the polygon if a device-dependent representation of the polygon is requested. If an edge is not requested, an outline of the non-clipped boundaries of the polygon interior will be drawn in the current polygon interior color and polygon interior line-style if the density of the polygon interior was not zero.

Polygons generated in a device-independent fashion will adhere strictly to the polygon style specification. The polygon interior generated would look similar when generated on different devices for a given polygon style specification. However, on raster devices rasterization of the fill lines may leave empty pixels when solid fill is requested with an orientation that is not 0 or 90 degrees. Available hardware would only be used where the polygon style could be generated exactly as specified.

The number of entries in the polygon style table and the default contents of the table are device dependent. However, all devices support the following polygon style table:

Entry	Density	Angle	Edge
1	0.0	0.0	1
2	0.125	90.0	1
3	0.125	0.0	1
4	−0.125	0.0	1
5	0.125	45.0	1
6	0.125	−45.0	1
7	−0.125	45.0	1
8	0.25	90.0	1
9	0.25	0.0	1
10	−0.25	0.0	1
11	0.25	45.0	1
12	0.25	−45.0	1
13	−0.25	45.0	1
14	−0.5	0.0	1
15	1.0	0.0	0
16	1.0	0.0	1

Error Conditions

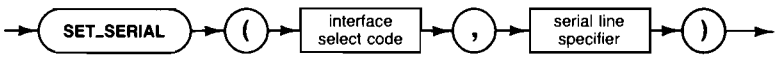
The graphics system must be initialized, a display must be enabled, and the parameters must be within the specified limits or this call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

SET_SERIAL

IMPORT: serial_
iodeclarations

This **procedure** will set the specified modem line on the connector. Not all lines are available at all times. The use of an Option 1 or Option 2 connector determines which lines are accessible.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
serial line specifier	Expression of enumerated TYPE *type_serial_line*.	rts_line cts_line dcd_line dsr_line drs_line ri_line dtr_line	

TABLE HERE

Semantics

The values of the enumerated TYPE *type_serial_line* have the following definitions:

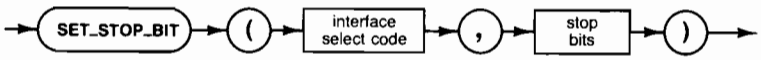
Name	RS-232 line
rts	ready to send
cts	clear to send
dcd	data carrier detect
dsr	data set ready
drs	data rate select
dtr	data terminal ready
ri	ring indicator

SET_STOP_BITS

IMPORT: serial_3
iodeclarations

This **procedure** will set the number of stop bits on the serial interface. The valid range of values includes 1, 1.5, and 2.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
stop bits	Expression of TYPE REAL.	–	1, 1.5, 2

SET_TEXT_ROT

IMPORT: dgl_lib

This **procedure** specifies the text direction.

Syntax

Item	Description/Default	Range Restrictions
x-axis offset	Expression of TYPE REAL	–
y-axis offset	Expression of TYPE REAL	–

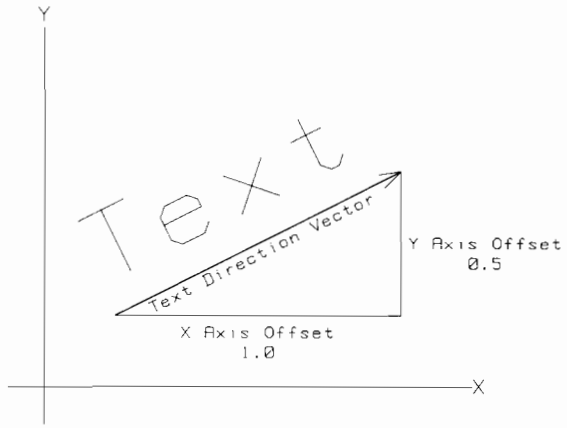
Procedure Heading

```
PROCEDURE SET_TEXT_ROT ( Dx, Dy : REAL );
```

Semantics

The **x axis offset** and the **y axis offset** specify the world coordinate components of the text direction vector relative to the world coordinate origin. These components cannot both be zero.

This procedure specifies the direction in which graphics text characters are output. The default value (X-axis offset = 1.0; Y-axis offset = 0.0) for the text direction vector is such that characters are drawn in a horizontal direction left to right. The default value is set during GRAPHICS_INIT and DISPLAY_INIT. With X-axis offset = - 1.0 and Y-axis offset = 1.0 a 135 degree rotation from the horizontal (in a counter clockwise direction) may be obtained.



Text Rotation Angle

Error Conditions

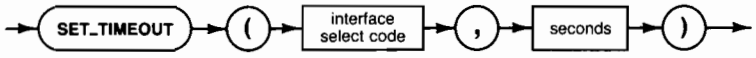
The graphics system must be initialized, a display must be enabled, and the parameters must be within the specified limits or this call will be ignored, an ESCAPE (– 27) will be generated, and GRAPHICSERROR will return a non-zero value.

SET_TIMEOUT

IMPORT: general_1
iodeclarations

This **procedure** will set up a timeout for all I/O Library input and output operations except transfer.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
seconds	Expression of TYPE REAL.	–	0, .001 thru 8192.000, inc. by .001

Semantics

Zero (0) is no timeout (infinite).

The resolution is to 1 millisecond.

If the select codes do not respond within the specified time an ESCAPE will be performed. Refer to the chapter on Errors and Timeouts.

Example:

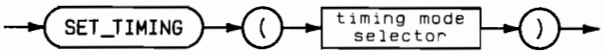
```
.
.
.
TRY
    SETTIMEOUT(12,1000);
    READCHAR(12,character);
RECOVER BEGIN
    IF Escapecode = Ioescapecode AND
       Ioe_result = Ioe_timeout  AND
       Ioe_isc = 12
    THEN WRITELN ('TIMEOUT on Interface 12')
END; {end of RECOVER}
.
.
.
```

SET_TIMING

IMPORT: dgl_lib

This **procedure** selects the timing mode for graphics output.

Syntax



Item	Description/Default	Range Restrictions
timing mode selector	Expression of TYPE INTEGER	0 or 1

Procedure Heading

```
PROCEDURE SET_TIMING ( Opcode : INTEGER );
```

Semantics

The **timing mode selector** determines the timing mode used.

Value	Meaning
0	Immediate visibility mode
1	System buffering mode

Graphics library timing modes are provided to control graphics throughput and picture update timing. Picture update timing refers to the immediacy of visual changes to the graphics display surface. Regardless of the timing mode used, the same final picture is sent to the graphics display. SET_TIMING only controls when a picture appears on the graphics display, not what appears.

The graphics system supports two timing modes:

- *Immediate visibility* Requested picture changes will be sent to the graphics display device before control is returned to the calling program. Due to operating system delays there may be a delay before the picture changes are visible on the graphics display device.
- *System buffering* Requested picture changes will be buffered by the graphics system. This means that the graphics output will not be immediately sent to the display device. This allows the graphics library to send several graphics commands to the graphics display device in one data transfer, therefore, reducing the number of transfers. System buffering is the initial timing mode.

The following routines implicitly make the picture current:

AWAIT_LOCATOR	DISPLAY_TERM	INPUT_ESC
LOCATOR_INIT	SAMPLE_LOCATOR	

The immediate visibility mode is less efficient than the system buffering mode. It should only be used in those applications that require picture changes to take place as soon as they are defined, even if the finished picture takes longer to create. When changing the timing mode to immediate visibility the picture is made current.

An alternative to immediate visibility that will solve many application needs is the use of system buffering together with the MAKE_PIC_CURRENT procedure. With this method, an application program places graphics commands into the output buffer and flushes the buffer (see MAKE_PIC_CURRENT) only at times when the picture must be fully displayed.

A call to MAKE_PIC_CURRENT can be made at any time within an application program to insure that the image is fully defined. MAKE_PIC_CURRENT flushes the output buffer but does not modify the timing mode.

Before performing any non-graphics system input or output (to a graphics system device) such as a PASCAL read or write, the output buffer must be empty. If the buffer is not flushed (via immediate visibility of MAKE_PIC_CURRENT) prior to non-graphics system I/O, the resulting image may contain some 'garbage' such as escape functions or invalid graphics data.

Note

Although SET_TIMING can be used with all display devices, only HPGL plotters buffer commands.

Error Conditions

The graphics system must be initialized and all parameters must be in range or this call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value.

SET_TO_LISTEN

IMPORT: hpib_1
iodeclarations

Note

This function is provided for use by the internal I/O Procedure Library drivers, only. Unexpected and possible undesirable results may occur if it is used.

SET_TO_TALK

IMPORT: hpib_1
iodeclarations

Note

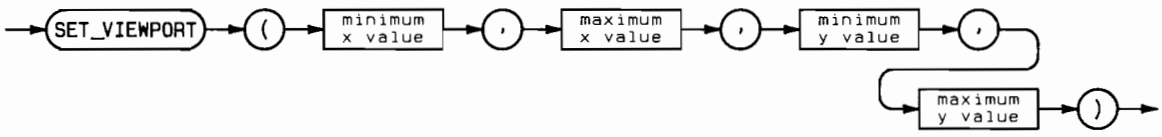
This function is provided for use by the internal I/O Procedure Library drivers, only. Unexpected and possible undesirable results may occur if it is used.

SET_VIEWPORT

IMPORT: dgl_lib

This **procedure** sets the boundaries of the viewport in the virtual coordinate system.

Syntax



Item	Description/Default	Range Restrictions
minimum x value	Expression of TYPE REAL	0.0-1.0
maximum x value	Expression of TYPE REAL	0.0-1.0
minimum y value	Expression of TYPE REAL	0.0-1.0
maximum y value	Expression of TYPE REAL	0.0-1.0

Procedure Heading

```
PROCEDURE SET_VIEWPORT ( Vxmin, Vxmax,
                         Vymin, Vymax : REAL );
```

Semantics

The **minimum x value** is the minimum boundary in the X-direction expressed in virtual coordinates.

The **maximum x value** is the maximum boundary in the X-direction expressed in virtual coordinates.

The **minimum y value** is the minimum boundary in the Y-direction expressed in virtual coordinates.

The **maximum y value** is the maximum boundary in the Y-direction expressed in virtual coordinates.

SET_VIEWPORT sets the limits of the viewport in the virtual coordinate system. The viewport must be within the limits of the virtual coordinate system; otherwise the call will be ignored.

The initial viewport is set up with the minimum x and y values set to 0.0 and the maximum X and Y values set to 1.0.

The initial viewport is set by GRAPHICS_INIT and SET_ASPECT. This initial viewport is mapped onto the maximum visible square within the logical display limits. This area is called the view surface. The placement of the view surface within the logical display limits is dependent upon the device being used. It is generally centered on CRT displays and is placed in the lower left-hand corner of plotters.

By changing the limits of the viewport, an application program can display an image in several different positions on the same graphics display device. A program can make a call to SET_VIEWPORT anytime while the graphics system is initialized.

The starting position is not altered by this call. Since this call redefines the viewing transformation, the starting position may no longer represent a known world coordinate position. A call to MOVE or INT_MOVE should be made after this call to update the starting position.

Error Conditions

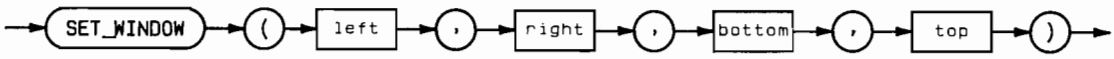
The graphics system must be initialized, all parameters must be within the specified range, the minimum X value must be less than the maximum X value and the minimum Y value must be less than the maximum Y value and all parameters must be within the current virtual coordinate system boundary, or this call will be ignored, an ESCAPE (−27) will be generated, and GRAPHICSERROR will return a non-zero value..

SET_WINDOW

IMPORT: dgl_lib

This **procedure** defines the boundaries of the window.

Syntax



Item	Description/Default	Range Restrictions
left	Expression of TYPE REAL	See below
right	Expression of TYPE REAL	See below
bottom	Expression of TYPE REAL	See below
top	Expression of TYPE REAL	See below

Procedure Heading

```
PROCEDURE SET_WINDOW ( Wxmin, Wxmax,
                       Wymin, Wymax : REAL );
```

Semantics

The **left** is the minimum boundary in the X-direction expressed in world coordinates. (i.e., the left window border). Must not equal maximum x value.

The **right** is the maximum boundary in the X-direction expressed in world coordinates. (i.e. the right window border). Must not equal minimum x value.

The **bottom** is the minimum boundary in the Y-direction expressed in world coordinates. (i.e. the bottom window border). Must not equal maximum y value.

The **top** is the maximum boundary in the Y-direction expressed in world coordinates. (i.e. the top window border). Must not equal minimum y value.

SET_WINDOW defines the limits of the window. All positional information sent to and received from the graphics system is specified in world coordinate units. This allows the application program to specify coordinates in units related to the application.

If the top value is less than the bottom value, the Y-axis will be inverted. If the right value is less than the left boundary, the X-axis will be inverted.

The window is linearly mapped onto the viewport specified by SET_VIEWPORT. This is done by mapping the left boundary to the minimum X-viewport boundary, the right boundary to the maximum X-viewport boundary, the bottom boundary to the minimum Y-viewport boundary, and the top boundary to the maximum Y-viewport boundary. If distortion of the graphics image is not desired, the aspect ratio of the window boundaries should be equal to the aspect ratio of the viewport.

The default window limits range from -1.0 to 1.0 on both the X and Y axis. GRAPHICS_INIT is the only procedure which sets the window to its default limits.

The starting position is not altered by this call. Since this call redefines the viewing transformation, the starting position may no longer represent a known world coordinate position. A call to MOVE or INT_MOVE should therefore be made after this call to update the starting position.

SET_WINDOW can be called at anytime while the graphics system is initialized.

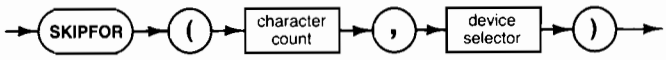
Error Conditions
The graphics system must be initialized, the minimum value for either axis must not equal the maximum value for that axis or this call will be ignored, an ESCAPE (-27) will be generated, and GRAPHICSERROR will return a non-zero value.

SKIPFOR

IMPORT: general_2
iodeclarations

This **procedure** will read the specified number of characters from the selected device. The characters will be thrown away.

Syntax



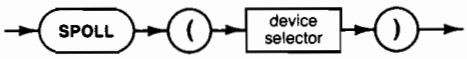
Item	Description/Default	Range Restrictions	Recommended Range
character count	Expression of TYPE INTEGER.	MININT thru MAXINT	–
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary.

SPOLL

IMPORT: hpib_3
iodeclarations

This INTEGER **function** will perform a serial poll to the selected device. The serial poll byte is returned by the function.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary.

Semantics

The interface must be active controller and the device must be a device address (i.e. 701, not 7). The bus sequence will look like:

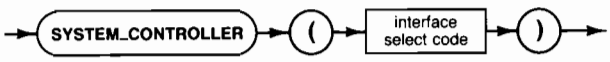
	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	Error	ATN UNL MLA TAD SPE $\overline{\text{ATN}}$ Read data ATN SPD UNT	Error	ATN UNL MLA TAD SPE $\overline{\text{ATN}}$ Read data ATN SPD UNT
Not Active Controller	Error			

SYSTEM_CONTROLLER

IMPORT: hpib_1
iodeclarations

This BOOLEAN **function** returns TRUE if the specified interface is the system controller.

Syntax



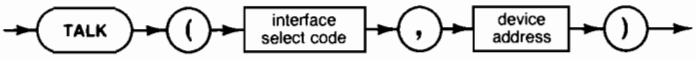
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

TALK

IMPORT: hpib_2
iodeclarations

This **procedure** will send a talk address over the bus. The interface must be active controller.

Syntax



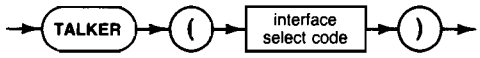
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
device address	Expression of TYPE *type_hpib_address*. This is an INTEGER subrange.	0 thru 3	Interface dependent

TALKER

IMPORT: hpib_3
iodeclarations

This BOOLEAN **function** will return TRUE if the specified interface is currently addressed as a talker.

Syntax



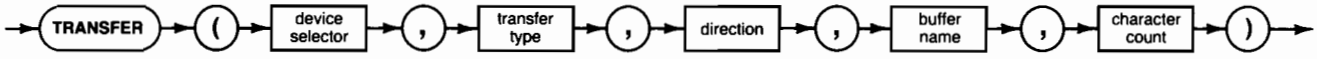
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

TRANSFER

IMPORT: general_4
iodeclarations

This **procedure** will transfer the specified number of bytes to or from the buffer space using the specified transfer type.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary
transfer type	Expression of the enumerated TYPE *user_tfr_type*.	serial_dma serial_fhs serial_fastest overlap_intr overlap_dma overlap_fhs overlap_fastest overlap	
direction	Expression of the enumerated TYPE *dir_of_tfr*.	to_memory from_memory	
buffer name	Variable of TYPE *buf_info_type*.	See glossary	
character count	Expression of TYPE INTEGER.	MININT thru MAXINT	

TRANSFER_END

IMPORT: general_4
iodeclarations

This **procedure** will transfer data to or from the buffer.

Syntax

Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary
transfer type	Expression of the enumerated TYPE *user_tfr_type*.	serial_dma serial_fhs serial_fastest overlap_intr overlap_dma overlap_fhs overlap_fastest overlap	
direction	Expression of the enumerated TYPE *dir_of_tfr*.	to_memory from_memory	
buffer name	Variable of TYPE *buf_info_type*.	See glossary	

Semantics

If the transfer is into the computer then the transfer will terminate when an END condition (like EOI) comes true or the buffer is filled. If The transfer is out of the computer then the transfer will send all of the available data with the END condition sent with the last byte.

TRANSFER_SETUP

IMPORT: general_4
iodeclarations

Note

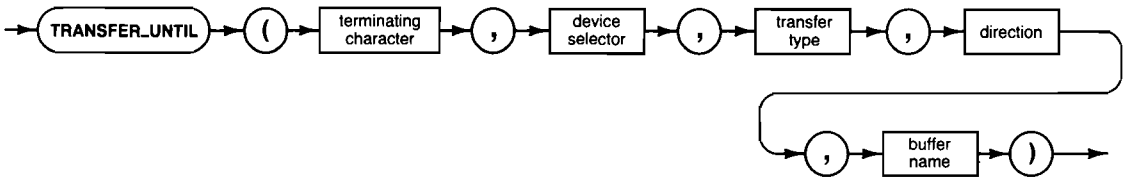
This function is provided for use by the internal I/O Procedure Library drivers, only. Unexpected and possible undesirable results may occur if it is used.

TRANSFER_UNTIL

IMPORT: general_4
iodeclarations

This **procedure** will transfer bytes into the buffer until the buffer is full or the termination character was received. (The DMA transfer type is not allowed).

Syntax



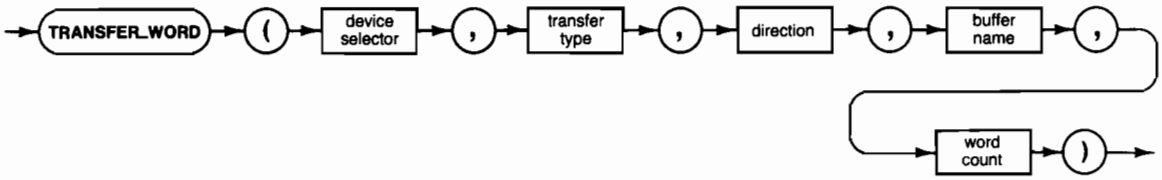
Item	Description/Default	Range Restrictions	Recommended Range
terminating character	Expression of TYPE CHAR.	–	
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary
transfer type	Expression of the enumerated TYPE *user_tfr_type*.	serial_dma serial_fhs serial_fastest overlap_intr overlap_dma overlap_fhs overlap_fastest overlap	
direction	Expression of the enumerated TYPE *dir_of_tfr*.	to_memory from_memory	
buffer name	Variable of TYPE *buf_info_type*.	See glossary	

TRANSFER_WORD

IMPORT: general_4
iodeclarations

This **procedure** will transfer the specified number of words into the buffer. This transfer will only work with 16-bit interfaces.

Syntax



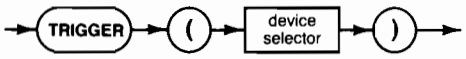
Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary
transfer type	Expression of the enumerated TYPE *user_tfr_type*.	serial_dma serial_fhs serial_fastest overlap_intr overlap_dma overlap_fhs overlap_fastest overlap	
direction	Expression of the enumerated TYPE *dir_of_tfr*.	to_memory from_memory	
buffer name	Variable of TYPE *buf_info_type*.	See glossary	
word count	Expression of TYPE INTEGER.	MININT thru MAXINT	

TRIGGER

IMPORT: hpib_2
iodeclarations

This **procedure** sends a trigger command to the specified device(s).

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary

Semantics

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN GET	ATN UNL LAG GET	ATN GET	ATN MTA UNL LAG GET
Not Active Controller	Error			

UNLISTEN

IMPORT: hpib_2
iodeclarations

This **procedure** will send an unlisten command on the bus. The interface must be active controller.

Syntax

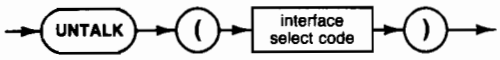
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

UNTALK

IMPORT: hpib_2
iodeclarations

This **procedure** will send an untalk command on the bus. The interface must be active controller.

Syntax



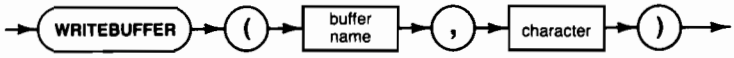
Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31

WRITEBUFFER

IMPORT: general_4
iodeclarations

This **procedure** will write a single byte into the buffer space and update the fill pointer in the *buf_info* record.

Syntax



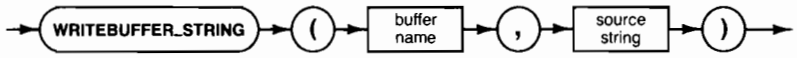
Item	Description/Default	Range Restrictions
buffer name	Variable of TYPE *buf_info_type*.	See Chapter 11
character	Expression of TYPE CHAR.	–

WRITEBUFFER_STRING

IMPORT: general_4
iodeclarations

This **procedure** will take the specified string and place it in the buffer and update the fill pointer. An error will occur if there is insufficient space.

Syntax



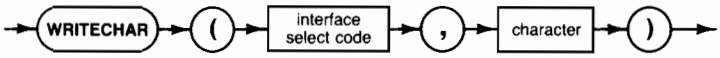
Item	Description/Default	Range Restrictions
buffer name	Variable of TYPE *buf_info_type*.	See Chapter 11
source string	Expression of TYPE *io_string*. This is STRING[255].	–

WRITECHAR

IMPORT: general_1
iodeclarations

This **procedure** will send a single byte as data over the interface path (writechar will drop the "ATN" line on an HP-IB interface).

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
source character	Expression of TYPE CHAR.	–	

Semantics

An HPIB interface must be addressed as a talker before performing a WRITECHAR, or an error will be generated. To avoid this, use the following sequence:

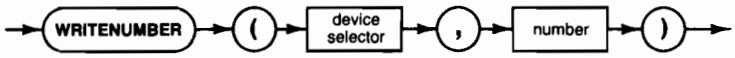
```
    .
    .
LISTEN (7,24);
TALK (7, MY_ADDRESS(7));
WRITECHAR (7, Character);
   .
   .
```

WRITENUMBER

IMPORT: general_2
iodeclarations

This **procedure** outputs a free field number to the specified device. The format rules follow the HP Pascal standard for WRITE. No additional characters are sent after the number.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary
number	Expression of TYPE REAL	–	

WRITENUMBERLN

IMPORT: general_2
iodeclarations

This procedure will output the number and a carriage return/ linefeed.

Syntax

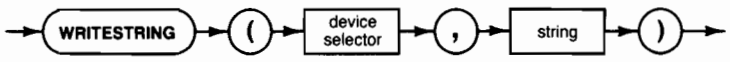
Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary
number	Expression of TYPE REAL	–	

WRITESTRING

IMPORT: general_2
iodeclarations

This **procedure** will send the specified string to the specified device. No additional characters are sent.

Syntax



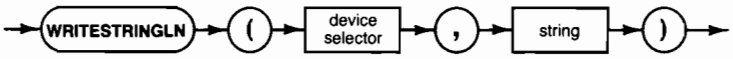
Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary
string	Expression of TYPE STRING	–	

WRITESTRINGLN

IMPORT: general_2
iodeclarations

This **procedure** will write out the string followed by a carriage return/line feed.

Syntax



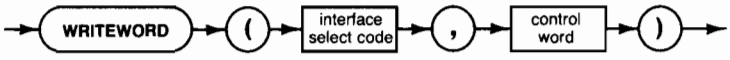
Item	Description/Default	Range Restrictions	Recommended Range
device selector	Expression of TYPE *type_device*. This is an INTEGER subrange.	0 thru 3199	See glossary
string	Expression of TYPE STRING	–	

WRITEWORD

IMPORT: general_1
iodeclarations

This **procedure** will write 2 consecutive bytes to a byte-oriented interface. A word oriented interface will write a single 16-bit quantity.

Syntax



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	Expression of TYPE *type_isc*. This is an INTEGER subrange.	0 thru 31	7 thru 31
control word	Expression of TYPE INTEGER.	MININT thru MAXINT	

Glossary

aspect ratio - The ratio of the height to width of an area (e.g. the area of a display surface).

attribute - See primitive attribute.

buffer name - A structured variable of TYPE *buf_info_type*.

complement drawing mode - A device dependent drawing mode for raster graphic displays in which a line is drawn by inverting bits in the display memory.

character cell - An imaginary rectangle placed around a character which defines its dimensions. The character size attribute determines the size of the character cell.

clipping - The elimination from view of all visible primitives or parts of primitives which lie outside the clipping limits (see window clipping).

default - See initial value.

device selector - An INTEGER expression used to specify the source or destination of an I/O transfer. A device selector can use either an interface select code or a combination of an interface select code and a primary address. To construct a device selector with a primary address, multiply the interface select code by 100 and add the primary address.

echoing - A mechanism for reflecting the status of an input function. Echoing is manifested in several ways as a function of the different input functions and the different physical devices being used.

erase drawing mode - A device dependent drawing mode for raster graphic displays in which a line is drawn by setting bits in the display memory to zero (off).

escape function - A facility within the graphics system which allows access to device dependent functions of a graphics display device.

file designator - A variable which points to the file informaton block for a lif file. It is a structured variable of the form:

```
LIFFILE = RECORD
            FPOINTER:INTEGER;
          END;
```

graphics display device - A device which displays graphics output.

initial value - The value of an attribute, viewing component, or characteristic of a work station which is in effect when the graphics system is initialized.

inquiry - User request for the current status, value, or characteristics of the graphics environment.

lif file name - The name of a lif file in the lif directory. A variable of TYPE *lifname*, which is a packed array of characters, of the form:

```
LIFNAME=PACKED ARRAY[1..10] OF CHAR;
```

line - A vector drawn from the current position to a specified point.

linestyle - An output primitive attribute which controls the pattern with which lines and text primitives are drawn.

locator device - An input device which returns a world coordinate point.

locator input function - An input function which returns a world coordinate point corresponding to a location on a locator device.

logical device - An abstraction of a typical graphics device, defined in terms of the type of data input or output. The logical devices supported by the graphics system are locator and graphics display.

logical display limits - The bounds of the logical display surface.

logical display surface - The portion of a graphics display device within which all output will appear.

mapping - The transformation of data from one coordinate system to another.

move - Moving the starting position to a specified point without generating a line.

object - The conceptual graphics entity in the application program. Objects are defined in terms of output primitives and primitive attributes. Their units are the units of the world coordinate system.

output primitive - The basic element of an object. The output primitives which the graphics system supports are: move, draw and text. Values of the primitive attributes determine aspects of the appearance of output primitives.

picture - A collective reference to all the images on a display device.

primary address - An INTEGER in the range 0 thru 31 that specifies an individual device on an interface which is capable of supporting more than one device. The HP-IB interace can support more than one device. (Also see "device selector.")

primitive - See output primitive.

primitive attribute - A characteristic of an output primitive, such as color, linestyle, character size, etc.

raster display - A type of graphics display in which all vectors are defined by turning on dots across a screen. TV is an example of a raster display.

sampled input - An input operation which does not require operator intervention; the routine returns with the current value as soon as the input device can respond.

viewing operation - See viewing transformation.

viewing transformation - An operation which maps positions in the world coordinate system to positions in device coordinates, thereby transforming objects into images.

viewport - The rectangular region of the view surface onto which the window will be mapped.

view surface - The largest rectangle within the logical display limits having the same aspect ratio as the virtual coordinate system.

virtual coordinate system - A two-dimensional coordinate system representing an idealized display device. Virtual coordinates are always in the range 0.0 to 1.0.

window - A rectangular region in the viewplane which may delimit the portion of the projected image which will be output.

world coordinate system - The two dimensional left handed cartesian coordinate system in which objects are described by the user program (user units).