STSDAS Version 2.0
Third Edition
September 1998

IGI Reference Manual

SPACE
TELESCOPE
SCIENCE
INSTITUTE

Science Software Group
Science Support Division
3700 San Martin Drive
Baltimore, Maryland 21218

Operated by the Association of Universities for Research in Astronomy, Inc., for the National Aeronautics and Space Administration

The STSDAS Group

Rick White	Branch Chief, Science Software Group
Perry Greenfield	Programming Supervisor
Ellyne Kinney	System Administration and Distribution
Howard Bushouse	NICMOS Calibration and Pipeline, Synthetic Photometry
Ivo Busko	Isophote, Fitting, Graphics
Michele de le Peña	GHRS, FOS
Warren Hack	FOC, igi, Graphics, Paper Products
Phil Hodge	Table System, FOC and STIS Calibration and Analysis, Fourier analysis
J.C. Hsu	HSP, FGS, WF/PC, WF/PC2 Calibration and Analysis, Paper products, Convfile, Ctools, Timeseries
Dick Shaw	nebular
Bernie Simon	Calibration Database, Synthetic Photometry, Table Editor, FITSIO, IRAF System Support
Anand Sivaramakrishnan	FGS Calibration and Analysis
Eric Wyckoff	User Support, Testing

Version 3: Written and Edited by Warren J. Hack (March 1998)
Version 2: Written and Edited by Zoltan G. Levay (October 1992)
Version 1: Written by Zoltan G. Levay

Send comments or corrections to:
Science Support Division
Space Telescope Science Institute
3700 San Martin Drive
Baltimore, Maryland 21218
E-mail: help@stsci.edu

M3.17

Table of Contents

Preface

Introduction to igi

In This Preface...

About igi...

The Interactive Graphics Interpreter (**igi**) is a program that draws high-quality data plots within the IRAF/STSDAS environment; **igi** works in a command-driven manner rather than the WYSIWYG[1] manner made popular by programs such as MacDraw. **igi** provides a full range of tools for creating and customizing plots and images to produce output that looks the way you want. In addition to basic plotting commands, **igi** can display images, annotate and label plots and images, and produce PostScript output. It also includes a scripting mechanism, allowing a user to automate the graphics creation process.

Using This Manual

This manual describes all of the major features incorporated in **igi** and provides tutorial information and a complete command reference. This section describes the organizational strategy behind this manual and explains the typographic and design features used throughout.

1. What You See Is What You Get.

Organization

Chapter 1 provides basic tutorial information and instructions for setting up and using **igi** for the first time. It includes brief examples of some of the types of graphics that can be created using **igi**.

Chapters 2 and 3 provide more details about how **igi** works and how it can work for you. *Chapter 2* concentrates on underlying concepts, such as command syntax and usage, operating modes, and parameter usage. *Chapter 3* explains available features in **igi** and how they are used; topics include basic drawing techniques, coordinate systems, axes, labels, image display, and working with windows.

Chapter 4 is an exhaustive command reference; it lists every command available in **igi** in alphabebetical order. Each command is described, along with any required and optional parameters.

Script examples and a glossary of terms are provided in the Appendixes.

Typographic Conventions

To help you understand the material in the *IGI Reference Manual*, we will use a few consistent typographic and design conventions.

Visual Cues

The following typographic cues are used:

- Figures and examples are often labelled with annotations and arrows to help explain their meaning. These annotations are displayed in **bold sans serif** type.
- **bold** words identify a STSDAS or IRAF task or package name.
- `typewriter-like` words identify a file name, system command, or response that is typed or displayed as shown.
- *italic* type indicates a new term, an important point, or a mathematical variable.
- `ALL CAPS` identifies an individual **igi** command.
- [BRACKETED LISTS] represent options, of which only one will typically be used. For example, [XY]DRAW refers to both of the commands XDRAW and YDRAW.

Keystrokes

Keystroke commands are described as follows.

- Shift-Q - When two keys are linked by a *dash*, both keys should be pressed at the same time.
- Esc D - When a *space* separates two keys, a sequence is indicated. Press one key, release it, then press the other.

- M - Press *only* the key. If we meant that you should press the shift key along with the indicated key, we would say so, such as in the first example above.

Comments

Occasional side comments point out three types of information, each identified by an icon in the left margin.

Tip: No problems...just another way to do something or a suggestion that might make life a bit easier.

Heads Up: Indicates something that is often done incorrectly or that is not obvious.

Warning: You could corrupt data, produce incorrect results, or create some similar problem.

Learning More About igi

This manual is a comprehensive document describing the **igi** software and its usage. Additional sources of information are:

- ***On-line help***: Type `help igi` at the IRAF cl prompt, or simply type `HELP` if you are already in **igi**.
- ***Web page***: A web page providing links to **igi** resources is at: `http://ra.stsci.edu/igi.html`
- ***Quick reference card***: A Quick Reference Card providing brief examples of **igi** usage is available in either printed form, or electronically via the STSDAS web page: `http://ra.stsci.edu/`
- ***Help Desk***: Questions about using **igi** can be sent to the STScI Help Desk via E-mail to: `help@stsci.edu`

CHAPTER 1:

Getting Started

In This Chapter...

This chapter is primarily for the first-time user: it provides an overview of **igi**'s capabilities and describes some basic techniques for using the software for the first time. More details about how **igi** works and how the software can be used to create specific types of graphics will be covered in Chapters 2 and 3.

What is igi?

Because it is an IRAF[1] application, **igi** is portable to any host architecture and graphics device supported by IRAF. It uses built-in graphics cursor interaction and produces standard IRAF graphics metacode, making it fully compatible and complementary with other IRAF graphics tasks. Input data may be of any IRAF-supported data format. **igi** provides a full range of capabilities from simple move and draw commands to powerful features for drawing full plots using only a few straightforward commands.

1. Image Reduction and Analysis Facility, available from NOAO.

What igi Can Do...

The ability to create plots is the most obvious use for **igi**—but the software does far more than just simple plots. This section presents three examples of the myriad ways that **igi**'s power can be harnessed to create, edit, and combine high-quality graphics that meet *your* needs:

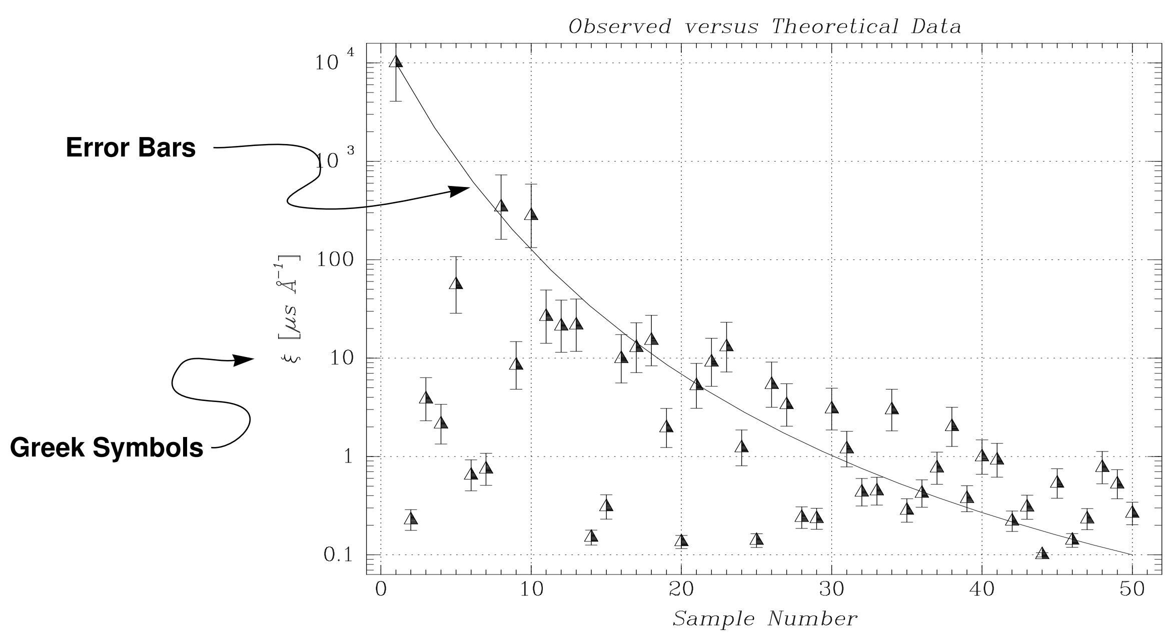
- Log plot.
- Labelled spectrum.
- Complex page with image, graphics, and text combined.

When possible, the data or script used to generate a plot is shown; in other cases, you may need to see Appendix A for details.

Sample Log Plot

One of the most basic uses for **igi** is to plot correlated data, such as spectra or color-magnitude plots. Plots with linear axes can be done using default settings in most cases, but sometimes log axes are needed in order to view all the data or to see the relationship clearly. This example shows how to set up a log-linear plot with the use of error-bars for the data. The data in this example are fabricated random numbers that bear no resemblance to any real observations or theory. The plot is shown in Figure 1.1; the script used to generate it is described on page 120.

Figure 1.1: Log Plot

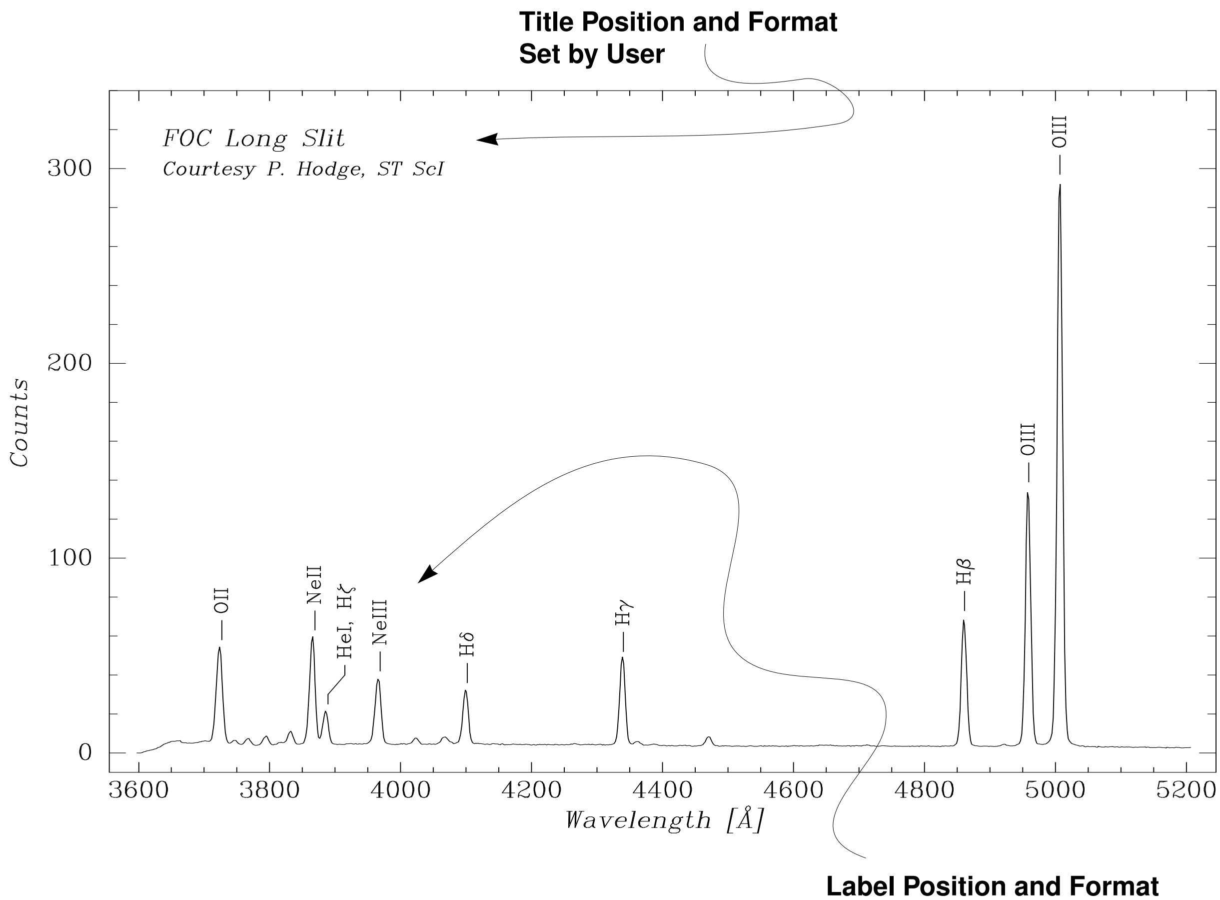


Labelled Spectrum

A relatively simple plot of a long-slit spectrum is shown in Figure 1.2, which demonstrates several common features including the text labelling capabilities of **igi**. In producing the example, some cursor interaction was used to determine the positions for the identifications. The script used to generate this plot is described on page 121.

A more advanced use of **igi** displays the image from which this spectrum was derived at the same scale as the background plot so that plotted spectral features line up with image features.

Figure 1.2: Labelled Spectrum

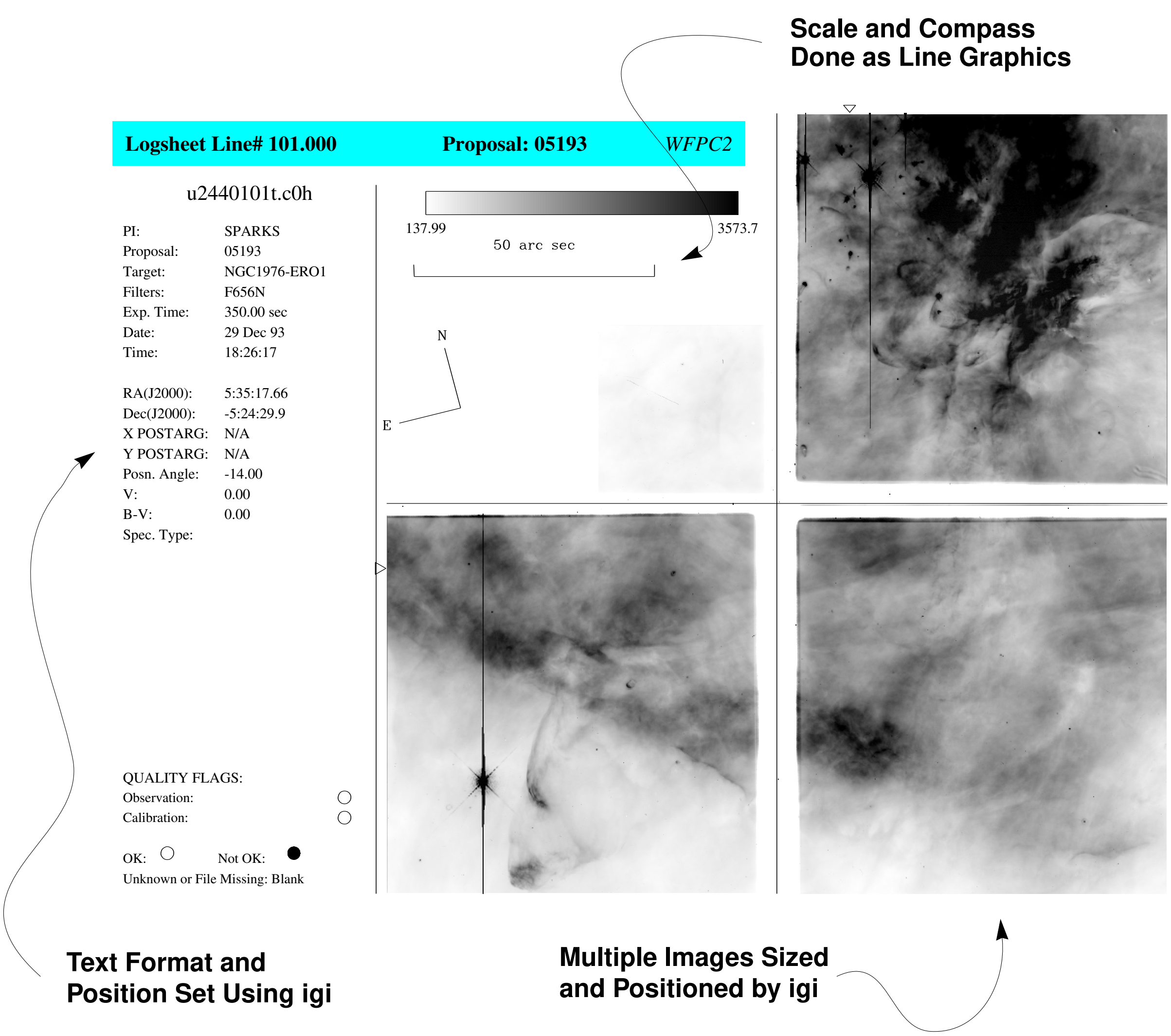


Complex Image Display

The STSDAS paper products task, **pp_dads**, produces complex graphics that blend image, plot, and text information on a single page. The task uses **igi** as its engine to produce graphical output. The **pp_dads** task creates **igi** scripts that describe the output pages—complete with annotated image displays, multiple images on a page, and plots of spectra and pointing positions. These **igi** scripts in turn create PostScript output which is printed and sent to HST observers as a summary of their observations.

Paper products represent an example of the type of image display manipulations that are possible with **igi** and are shown in Figure 1.3. The image is positioned in one part of the page after being rotated and block-averaged by **igi**, while the text is positioned in another section.

Figure 1.3: Complex Graphic with Multiple Images, Text, and Line Drawing



Starting igi

How to Get In

To start an interactive session of **igi**, first start IRAF using the `cl` command at your system prompt, load the **stsdas** and **stplot** packages, then type `igi`. The **tbplot** package in TABLES contains the same version of **igi**.

How to Get Out

To quit **igi**, type `BYE` or `END`.

Getting Help

On-line help is available either from within the cl or within **igi**.

At the cl prompt:

- To see the complete on-line help, type:
  `cl> help igi`
- To get help on a specific command, use the `section` parameter of the `help` command. For example, to see the help for the `DATA` command, type:
  `cl> help igi section = data`

From within igi:

- To see the *complete* on-line help, type:
  `igi> help`
- To get help on a *specific command*, simply type the name of the command after the word `help`. For example:
  `igi> help data`
- To see a list of *available commands*, type:
  `igi> menu`
- To find commands related to a specific *topic*, use the `APROPOS` command, for example:
  `igi> apropos axis`

User Support

If you have a question or problem using **igi**, user support is available through the STScI Help Desk:

- ***E-mail:*** `help@stsci.edu`
- ***Phone:*** (410) 338-1082

WWW Documentation

The STSDAS web page (`http://ra.stsci.edu/`) contains an electronic version of this document, complete with scripts and examples, as well as the on-line help and other resources.

A First Session...

The best way to learn a program is to use it. In this section we describe the commands and data used to produce a simple **igi** plot—one that demonstrates basic commands and serves as a foundation for more complex graphics.

Each step and command for this first session will be shown as it would be done within **igi**, along with an explanation of its purpose and result. Subsequent examples will simply list and explain **igi** commands.

Input Data

For this first session, a data set is needed. The simple ASCII file shown below (called `sample1.dat`) will be used to generate the plot.

```
0.2        5        400
0.3        7        240
0.4        9        150
0.5        16       120
0.6        25       100
```

Commands to Create the Plot

1. Start **igi** in interactive mode:

```
cl> igi
igi>
```

2. Tell **igi** what data file will be used (using the `DATA` command) and what columns will serve as the data points (using the `XCOLUMN` and `YCOLUMN` commands):

```
igi> data sample1.dat
igi> xcolumn c1
igi> ycolumn c3
```

The columns are designated in the same way as they would be with the **tables** tasks: ASCII file columns have default names of `c1`, `c2`, and so on. For this example, we will plot the first and third columns of data (`c1` and `c3`).

3. Now that **igi** knows what data to work with, let it calculate the limits for the plot:

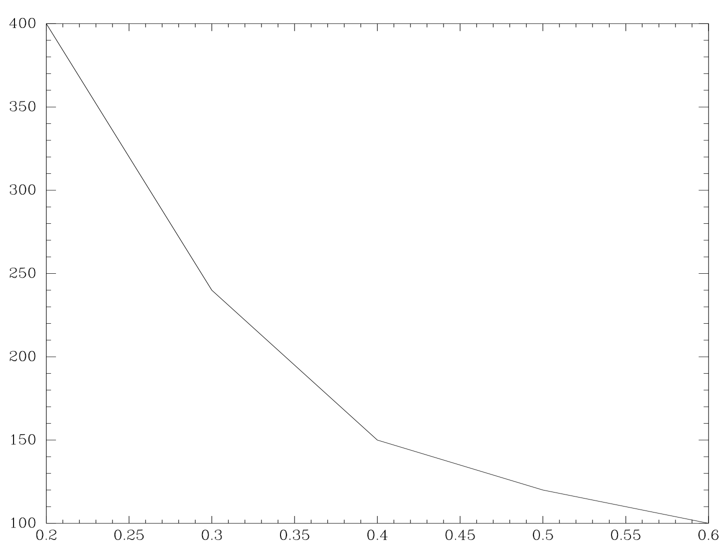
```
igi> limits
```

4. We can now draw the plot axes and plot the data:

```
igi> box
igi> connect
```

That's it! The plot is done. If this session were being run on an xterm or xgterm window, the plot would show up in a separate graphics terminal like Figure 1.4.

Figure 1.4: Basic Plot



Using xtermjh or xgterm provides best graphics output control.

Using a Script

Creating plots with **igi** can either be done *interactively*, as in the previous example, or *non-interactively* through the use of a script. The next example shows how to create an **igi** script and use it to produce a plot.

Creating the Script

An **igi** script is simply a text file containing all the commands needed to create a plot—the commands are listed, line by line, just as if they were being typed interactively. The script lets you save a set of plot commands, run them through **igi**, and then easily reiterate the process to tweak the plot, add features, or annotate as you wish. Because the script is a text file, you can create or edit it using any available text editor. In this example, we saved the script to a file called `sample1.igi`.

Running the Script

When you use a script, there is no need to run **igi** in interactive mode—the script can be passed to **igi** from the cl prompt (make sure the **tbplot.stplot** package is loaded). Run the **igi** script from the IRAF command line as shown below:

```
st> igi < sample1.igi
```

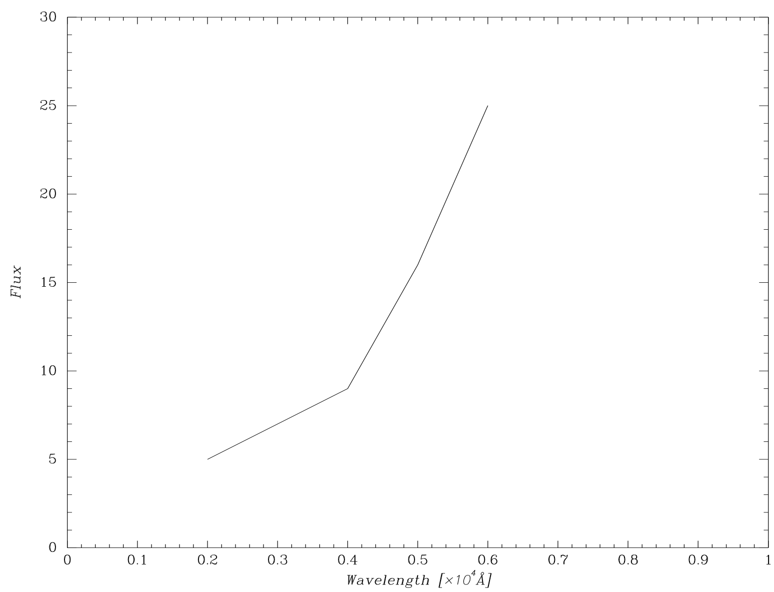
The way in which the plot is displayed depends on the setting of your `stdgraph` environment variable—such as a Tek window associated with an xterm window. Figure 1.5 shows the sample script and the plot that it creates. Changes can now be made to the script and the command re-run to to re-plot the data or change its appearance (perhaps changing labels, limits, and text sizes).

Figure 1.5: Generating a Plot Using a Script

Script to Generate Plot (sample2.igi)

Read File "test.dat"	`DATA sample1.dat`
X Data is from Column 1	`XCOLUMN c1`
Y Data is from Column 2	`YCOLUMN c2`
Scale the Plot	`LIMITS 0 1 0 30`
Size for Text	`EXPAND 0.9`
Draw Axes	`BOX`
Draw Curve	`CONNECT`
Label for Y Axis	`YLABEL "\\iFlux"`
Label for X Axis	`XLABEL "\\iWavelength [\\g*10\u4V\r"`

Generated Plot



Complex Scripts with Macros

Macros are named lists of commands—like a mini-script—that can handle specific (often repetitive) functions and that are called from within a complex script. The use of macros can dramatically increase the flexibility and ease of use, but is a topic well beyond the scope of these examples. Macros are discussed more fully with examples in “Macros” on page 31.

Printing and Saving Using PostScript

When preparing graphics for publication or use in presentations, you will need to use PostScript to either print a hardcopy version of your work, or to save as a file that can be used by other software. The IRAF/STSDAS environment includes a simple mechanism that allows **igi** to produce PostScript output—the `device` parameter.

Saving Graphics to PostScript Files

The output mechanism is defined within IRAF using the `device` parameter. (After setting the parameter, use **gflush** to actually write the file.) Some useful `device` values are available for creating PostScript files are:

- ***Portrait mode*** (normal page orientation): `psi_port`
- ***Portrait mode—monochrome***: `psi_portm`
- ***Landscape mode*** (page oriented lengthwise): `psi_land`
- ***Landscape mode—monochrome***: `psi_landm`

Here's how to create a PostScript file of the plot shown in Figure 1.5:

Name of Output PostScript File →

```
st> igi device=psi_land < sample2.igi
st> gflush
/tmp/psk0000a
```

After your file is written, you will probably want to copy it to a more permanent home and give it a more useful file name.

Notice in the example above that the `sample2.igi` script is run from the command line, just as in Figure 1.5, and that `device` is simultaneously set directly from the command line. Once `device` is set, it remains open for graphic output until it is explicitly closed, which is the function of **gflush**—your file isn't written until you issue the **gflush** command. Each plot is added as a spearate page before **gflush** is used, unless `append=yes` is set, either within **igi** or another IRAF graphics task.

Printing igi Plots

To print a plot, set `device` to the name of the printer you want to use. Available printers are listed in the `graphcap` file, which is displayed with the **devices** command. For example, if the printer we wanted to use was named "lw1", we would print out the sample plot from Figure 1.5 as follows:

```
st> igi device=lw1 < sample1.igi
st> gflush
```

The IRAF system task **devices** can be used to print a list of all available printers and tape drives.

Adding Text to Plots

The simple plot in Figure 1.5 used two **igi** commands to label the axes: `XLABEL` and `YLABEL`. Text can also be added to the graphic to add titles, call out certain features, or perhaps label points and curves. A range of commands to control the look of text, such as fonts, sizes, and position, is also available.

Adding a Title

Titles are added using the `TITLE` command. This command puts a text string at the top of the current plot centered above the top X (horizontal) axis. For example, the following command will add the title “This is a sample plot” to our graphic:

```
TITLE This is a sample plot
```

Annotating a Figure

Text commands (Table 1.1) can be used to place a label of almost any font, size, and orientation almost anywhere on the page. Table 1.1 lists some of the commands that can be used to control the appearance and placement of text labels in a figure.

Figure 1.6 shows some **igi** commands that could be used to add text labels to the basic plot shown in Figure 1.5. Full details on these commands will be described in “Text Annotation” on page 58. In this example, a different PostScript font is used for each label, and the labels are placed at different positions. The script uses the following commands:

- `PSFONT`: Defines the font to be used for a label.
- `PUTLABEL`: Defines the contents of the label. The number following the command specifies the justification to be used. The `\fP` is an escape sequence that selects the newly-defined font.

- `RELOCATE`: Sets the starting point of the label.

Table 1.1: Text-Handling Commands

Operation	Command	Purpose
Font Selection	`FONTSET`, `PSFONT`	Select typeface; can use software (igi), hardware, or PostScript fonts
Text Size	`EXPAND`	Control size of letters (1.0 is default)
Orientation	`ANGLE`	Set angle of text across plot (rotation angle)
Special Characters		Escape codes are used to print special characters, e.g., Å
Location	`RELOCATE`, `MOVE`	Determines where the text is positioned in the window
Justification	`JUSTIFY`, `PUTLABEL`	Set the position and text justification (left, right, center)

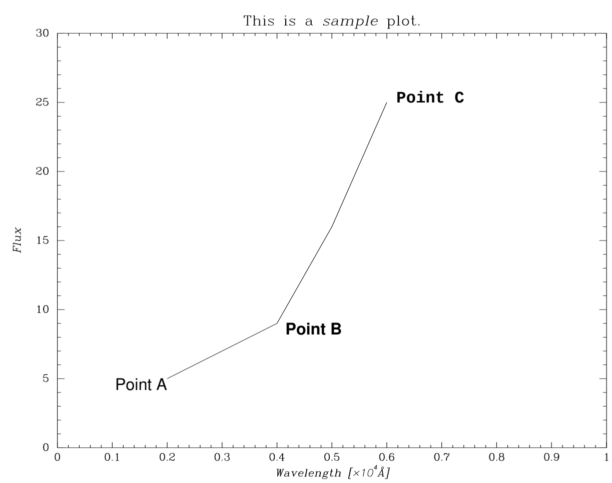
Figure 1.6: Adding Text Labels to Graphic

Additional Commands for Labelling Data Points

```
FONTSET hard
PSFONT Helvetica
expand 1.25
relocate 0.2 5
putlabel 1 \fP Point A
relocate 0.4 9
PSFONT Helvetica-Bold
putlabel 3 " \fP Point B"
relocate 0.6 25
PSFONT Courier-Bold
putlabel 9 \fP Point C
```

Choose Font with PSFONT
Set Text Size with EXPAND
Position with RELOCATE
Define Text with PUTLABEL

Plot with Labelled Points



Working with Tables

Many types of data can be used by **igi**, including ASCII text files, images, and tables. Up to this point, the examples were based on text files. This section describes how to work with tables (images will be covered in the next section).

There are two ways to read input data from 3-D tables:

- Row selectors with the `DATA` command.
- Extra row parameter in the `[XY]COLUMN` command.

Row Selectors

Data in 3-D tables is read using the `DATA` command, but with additional text after the file name to select the rows to be read. Figure 1.7 shows an example of how the row selector syntax works. If the 3-D table `spec_tab3d.fits` contains three spectral scans, each representing one row, and we wanted to plot only the first scan, we would produce a script like that shown in Figure 1.7.

Figure 1.7: Reading Table Data with Row Selector Syntax

Row Selector

```
FONTSET hard
DATA spec_tab3d.fits[r:row=1]
XCOLUMN wavelength
YCOLUMN flux
LIMITS
BOX
CONNECT
FONTSET soft
XLABEL Wavelength (\gV)
YLABEL Relative Flux
```

The table also has a column with the spectral order listed in a column named “sporder” with orders 3–6. The row selector could, therefore, have been specified based on the spectral order desired:

```
DATA spec_tab3d.fits[r:sporder=3]
```

The row selector syntax is flexible and efficient, but can be complex. We recommend reading the on-line help (type `help selectors`) before using row selectors.

Row Parameter Usage in [XY]COLUMN

The `[XY]COLUMN` commands allow you to simply specify the row number to be extracted. To produce the same plot as would be created by the script shown in Figure 1.7, but without using row selector syntax, we would read the input data using the `DATA` command specifying only a file name—no row. We would than add a "1" to the end of the `XCOLUMN` command to plot only the first row (spectral order) of the data. The rest of the script would be the same as given in Figure 1.7.

Figure 1.8: Reading Rows Using [XY]COLUMN

```
FONTSET hard
DATA spec_tab3d.fits
XCOLUMN wavelength 1
YCOLUMN flux
LIMITS 1545 1560 0 1
BOX
CONNECT
FONTSET soft
XLABEL Wavelength (\gV)
YLABEL Relative Flux
```

Select Only Row 1

Working with Images

igi provides a fairly flexible set of tools for displaying, re-sizing, and annotating images. There are just a few **igi** commands which must be given in the correct order to get everything to work right. The following sample will demonstrate how to work with images.

Basic Image Display

Simply reading in an image and displaying it on a page forms the basis for working with images in **igi**. The following sample session will demonstrate how to read in an image, in this case the default IRAF image `dev$wpix`, and print it to a page. Each step will be explained as if the commands were being given in an interactive session, although the complete script will be provided at the end as a summary.

1. Start **igi** with output sent to a landscape-oriented PostScript file.

   ```
   cl> igi device=psi_land
   ```

2. Read in the image `dev$wpix` (the default IRAF image with WCS information) using the `ZSECTION` command.

   ```
   ig> ZSECTION dev$wpix
   ```

3. Scale the image to fit a specified region of the page while preserving the original aspect ratio of the image, as determined from the number of pixels in each dimension. In this example, the image starts at a position 25% (0.25) of the way from the left edge of the page, and at the bottom of the page and will be fit into a square section that is 75% (0.75) of the page size on a side. The pixels will be expanded (magnified) to fill this region—**igi** will display them so that the entire image fits in the region with its original aspect ratio, regardless of how much space remains on the page.

   ```
   ig> FITPIX 0.25 1.0 0.0 0.75
   ```

4. Determine the limits of the image coordinates.

   ```
   ig> LIMITS
   ```

5. Produce the grayscale transformation to best display the image. In this case, we will reverse the order of the pixel value limits to produce a reverse video display. We will have **igi** linearly map the grayscales to pixel values between 0 and 400 using the `ZRANGE` command. By specifying the maximum value first, the grayscale will be inverted to map the higher pixel values as black and fainter pixel values as white.

   ```
   ig> ZRANGE 400 0
   ```

6. Output the image to the page using the limits and scaling determined in the previous steps.

   ```
   ig> PIXMAP
   ```

7. Label the image using the coordinates derived from the image header.

   ```
   ig> WCSLAB
   ```

8. Write output to PostScript file.

   ```
   ig> end
   cl> gflush
   ```

The **igi** script that would be identical to this session, and the displayed image, can be seen in Figure 1.9.

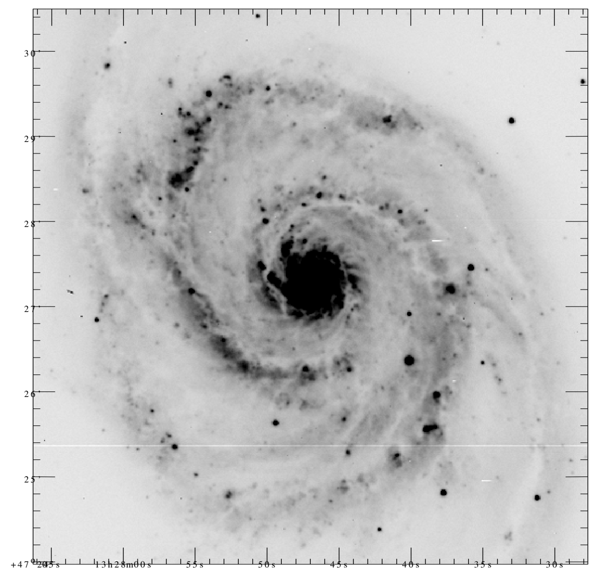
The order of these commands is important. Unpredictable things happen to the image display if the order is not preserved, however, inserting commands to perform other tasks among these commands causes no problems.

Figure 1.9: Basic Image Display

Script

```
zsection dev$wpix
fitpix 0.25 1 0.0 0.75
limits
zrange 400 0
pixmap
wcslab
```

Image Display



Advanced Image Manipulations

The basic image display provided in Figure 1.9 will be sufficient for most purposes, however, it is possible to run other commands to improve the display quality and to make it more efficient (often without noticeable degradation) This image manipulation uses the commands `ZSECTION and  ZEVALUATE`. In the section "Image Display" on page 64 we will discusses these and other methods of image display in more detail.

ZSECTION Usage

Optional parameters for the `ZSECTION` command were used in the paper products example shown in Figure 1.3 to mosaic the individual WFPC2 chip images into one image (see the scriptFigure A.3 on page 122A.3). To apply these parameters to the `dev$wpix` image, we would re-write the script shown in Figure 1.9 to look like the one in Figure 1.10. The `ZSECTION` command would block average the image using a 2 x 2 kernel and rotate the image counter-clockwise by 90°.

Figure 1.10: Displaying Image with Coordinate Labels

```
zsection dev$wpix 2 90
fitpix 0.0 1 0.0 0.75
limits
zrange 400 0
pixmap
wcslab
```

Parameters on ZSECTION for Block-Averaging and Image Rotation

Scaling Image Data

One of the more obscure functions in **igi** is the ability to scale the pixel values of an image in memory before displaying them. Typically, only 256 grayscale values are available to a PostScript printer to reproduce an image. Therefore, dynamic-range data needs to be scaled in order to effectively display it. This can be done with the `ZEVALUATE` command to take the log of the image, for example. The following command could be inserted into the script in Figure 1.9 right after the `ZSECTION` command:

```
ZEVALUATE log10(z+1)
```

where *z* is the pixel value from the image. A value of 1 is added, in this case, to insure that the minimum value in the image is greater than 0 to avoid problems with log(0). A zero value would cause the output of the log of the image with the 256 grayscales to span a much smaller range of pixel values.

Quick Guide to igi Commands

This section lists every **igi** command, grouped by function, and provides a brief description of what the command does. Complete descriptions of each command are found in Chapter 4. The commands can be classified into the following groups:

- ***Execution***: These commands control the execution of scripts and macros, along with providing information about current **igi** settings.
- ***Data Input:*** These commands deal with reading in the data for use within **igi**.
- ***Data Evaluation:*** These commands perform arithmetic operations on the input data.
- ***Text Attributes***: These commands control all attributes of text output.
- ***Text and Labels***: These commands create axis labels and other text annotations.

- ***Coordinates and Scaling***: These commands are used in specifying the coordinate systems to be used and how to scale the input data.
- ***Plotting and Image Display***: These commands display the plotted data or image.

Table 1.2: Commands by Function

Command	Page	Description
		Command Execution
`!`	76	Escape a command to the cl
`?`	77	Page the help summary menu
`^`	77	Re-execute a previous command
`APROPOS`	78	List commands associated with a keyword
`BYE`	81	Terminate igi (alias for END*)*
`CURSES`	84	Start interactive cursor mode
`DEFINE`	75	Enter macro text
`EDITCMD`	87	Edit the command buffer or macro text
`END`	88	Exit igi or terminate macro define (end the current mode)
`HELP`	96	Print help
`INPUT`	97	Execute commands from a file
`LIST`	100	List the command buffer or macro text
`MACROS`	102	List defined macros
`MENU`	103	Page the help summary
`PAGECMD`	105	Page the command buffer or macro text
`PLAYBACK`	106	Execute the commands in the command buffer
`SHOW`	113	Page the plot limits and attributes
`UNDO`	115	Selectively erase the last plot command
`VERSION`	115	Show igi version and date
`WRITE`	117	Write the command buffer to a file
		Data Input Commands
`DATA`	85	Specify the input data file
`DLIST`	86	Print the data values
`ECOLUMN`	82	Read errors from a column of the data file
`ESECTION`	111	Read an image section into the error data buffer
`LCOLUMN`	82	Read limits data from a column of the data file

Table 1.2: Commands by Function (Continued)

Command	Page	Description
LINES	99	Specify the range of input rows to read
LSECTION	111	Read an image section into the limits data buffer
PCOLUMN	82	Read point specifications from the input data file
PSECTION	111	Read an image section into the point-type data buffer
READ	108	Read commands into the command buffer without execution
SAOCMAP	110	Specify a gray or color map for display
SCOLUMN	82	Read scratch data from a column of the data file
SSECTION	111	Read an image section into the scratch data buffer
XCOLUMN	82	Read X coordinates from a column of the data file
XSECTION	111	Read X coordinates from an image section
YCOLUMN	82	Read Y coordinates from a column of the data file
YSECTION	111	Read Y coordinates from an image section
ZSECTION	111	Read Z values from an image section
		Data Evaluation Commands
EEVALUATE	91	Operate on error values
LEVALUATE	91	Operate on limit arrays
PEVALUATE	91	Operate on point code values
SEVALUATE	91	Operate on scratch array values
XEVALUATE	91	Operate on X data values
XLOGARITHM	101	Take common log of X data
YEVALUATE	91	Operate on Y data values
YLOGARITHM	101	Take common log of Y data
ZEVALUATE	91	Operate on image pixel values
ZRANGE	117	Map image values display brightness
		Plotting Commands
AXIS	78	Draw and label an arbitrary axis
BARGRAPH	79	Draw a bar graph
BOX	80	Draw and label the axes
COLOR	81	Set the drawing color
CONNECT	84	Draw a (polyline) curve connecting the data coordinates
DDRAW	86	Pen down move (draw) in NDC

Table 1.2: Commands by Function (Continued)

Command	Page	Description
DMOVE	103	Pen up move in NDC (alias for DRELOCATE)
DOT	86	Draw a single marker
DRAW	86	Pen down move (draw) in WC
DRELOCATE	109	Pen up move in NDC
ELLIPSE	87	Draw an ellipse
ERASE	88	Erase the screen
ERRORBAR	89	Draw error bars
HISTOGRAM	96	Draw a histogram plot (bar graph)
GRID	96	Draw lines between major axis ticks
MINMAX	103	Print the data range
MOVE	103	Pen up move (alias for RELOCATE)
NUMBER	105	Draw numerals representing index of each coordinate
PDRAW	86	Pen down move (draw) in PC (Physical coordinates)
PIXMAP	106	Draw (render) an image (pixmap)
PMOVE	103	Pen up move in PC (alias for DRELOCATE)
POINTS	106	Draw a polymarker at input data coordinates
POLYGON	106	Draw a filled polygon using the input data coordinates
PRELOCATE	109	Pen up move in PC
RELOCATE	109	Pen up move in WC
STEP	113	Draw a stepped curve (histogram without vertical lines)
ULLIM	114	Draw upper or lower limit symbols
VDRAW	86	Pen down move (draw) in VPC (virtual page coordinates)
VMOVE	103	Pen up move in VPC (alias for VRELOCATE)
VRELOCATE	109	Pen up move in VPC
		Text and Label Commands
ID	97	Write identification label
LABEL	98	Draw text at current position
PUTLABEL	108	Draw a label with a specified justification
TITLE	114	Specify the plot title
XLABEL	98	Specify the X axis label
XSIXTY	113	Specify the use of sexagesimal numbering for the X axis

Table 1.2: Commands by Function (Continued)

Command	Page	Description
YLABEL	98	Specify the Y axis label
YSIXTY	113	Specify the use of sexagesimal numbering for the Y axis
WCSLAB	115	Label the viewport in WCS coordinates
		Text Attributes
ANGLE	77	Set the marker and text rotation angle
ETYPE	90	Change the error bar style
EXPAND	92	Set the marker and text size
FILLPAT	92	Select a filled-area pattern
FMTICK	94	Specify the axis tick label format
FONTSET	95	Select igi or gio fonts
JUSTIFY	97	Set text justification
LTYPE	101	Set the line style (hardware dependent)
LWEIGHT	102	Set the line width (hardware dependent)
NOTATION	104	Set the format for axis labels
PSFONT	107	Select PostScript font to use for special characters
PTYPE	107	Specify the point marker type
RESET	110	Return settable plot and data parameters to defaults
TICKSIZE	114	Set the axis tick spacing
		Coordinates and Scaling
FITPIX	93	Set a viewport to match an image's aspect
IMGWCS	97	Toggle using image WCS for X values
LIMITS	99	Set the plot scale
LOCATION	100	Set the viewport on the virtual page
MARGIN	102	Rescale to include a border between curve and axes
PHYSICAL	105	Set the virtual page in physical device coordinates
VPAGE	115	Set the virtual page in normalized coordinates
WINDOW	116	Divide screen into panes
XFLIP	94	Flip X axis
YFLIP	94	Flip Y axis

CHAPTER 2: igi Environment

In This Chapter...

This chapter explains the command environment within **igi** and how **igi** interacts with the IRAF environment.

Using igi

Since **igi** offers a separate environment and command syntax from the cl, this section will discuss the IRAF parameters used by **igi** and how to start up **igi** from the IRAF command line.

Startup Options

The command line examples in Table 2.1 illustrate several ways in which you can start **igi**.

Table 2.1: Sample igi Task Command Lines

Sample Command Line	Intended Action
`st> igi`	Interactive commands and immediate graphics display
`st> igi <command.igi`	Redirect input from a command file
`st> igi >G igi.gmc`	Interactive commands, graphics saved in a file
`st> igi <cmd.igi >G igi.gmc`	Commands from a file and graphics to a metacode file
`st> sgraph mydata` `st> igi append+`	Add to graphics drawn by another IRAF task
`st> doplot | igi`	Pipe commands written by the user-created **doplot** task
`st> igi init="data mydat" <mycmd.igi`	Start with an initial command (see "Task Parameters", below)

The `append` parameter allows **igi** to add graphic overlays onto a plot drawn by any other IRAF graphics task. The `initcmd` parameter lets **igi** accept a command string to be immediately executed upon start-up; this allows the same command to be run with different initial states.

Task Parameters

All **igi** task parameters are *hidden*—their values need not be specified on the command line because they have default values. If the parameters are set on a command line, they must be specified using *both* the parameter name and value. Each of these parameters is described in more detail below.

Table 2.2: igi Task Parameters

Parameter	Type	Default	Description
`initcmd`	string	"" (null)	Command to execute upon starting **igi**
`device`	string	`stdgraph`	The graphics device on which to plot
`metacode`	file	"" (null)	Output binary metacode file (optional)
`append`	boolean	`no`	Append to existing plot?
`debug`	boolean	`no`	Write debugging information?
`cursor`	gcur	`"" (null)`	Graphics cursor

Initial Command (initcmd)

The `initcmd` task parameter is a command that will be executed when **igi** starts, regardless of the operating mode (i.e., interactive or non-interactive). Multiple commands can be executed by separating them with semicolons, as with commands entered interactively or from a command file. Note, however, that the command line is restricted to a maximum size of 160 characters.

Graphics Device (device)

The graphics device parameter, `device`, is consistent with other IRAF graphics tasks. Its default value, `stdgraph`, uses the device set in the IRAF cl environment variable named `stdgraph`. For example, if `stdgraph` = `xterm`, and you are running the cl in an xterm window, then the plot will appear in the xterm graphics window. If `device` = `file`, the graphics output is stored in the file specified by the parameter `metacode`, equivalent to `igi >G file`. The parameter `metacode` is the name of the output GKI[1] binary *metacode*[2] file. This parameter is required if `device` = `"file"` but is otherwise ignored. Alternately, you can redirect graphics output by using `>G file` on the command line.

The Graphics Buffer and Your Graphics Window (append)

If `append = no` (the default), **igi** will clear the screen (start a new page) upon starting. If `append = yes` it will preserve any graphics already in the graphics buffer drawn by any other IRAF graphics task. This lets arbitrary annotations appear on plots. Note that this only works if you preserve the IRAF graphics buffer, that is, do not use `gflush` between running graphics tasks. If you do flush the graphics buffer, the new task will behave as a new plot regardless of the value of `append`. A third possibility is to create a new frame but write to the same graphics file, either *metacode* or eventually a PostScript file, for example. In this case, use the `ERASE` command in **igi** to create a new frame.

Debugging (debug)

The `debug` parameter is a boolean parameter whose default value is `no`. If `debug` = `yes`, *internal debugging* information is written to standard output—not particularly relevant to most users but sometimes useful in determining why something went wrong.

---

1. Graphics Kernel Interface; type `help gio$doc/gio.hlp fit` for more explanation.

2. IRAF-specific, device-independent binary file containing plotting instructions.

Cursor Mode (cursor)

To use cursor mode commands, specify the file name of a cursor list file For an in-depth explanation of cursor mode, see the cl cursor help by typing `help cursor` in the cl.

Graphics Kernel

igi creates its output using graphics routines provided by the IRAF graphics kernel and descriptions of the graphics devices in the IRAF `graphcap` file. The graphics are then translated to files using either the IRAF graphics kernel or the STSDAS PostScript kernel, called `psikern`. This close-knit relationship with IRAF's graphics system provides the flexibility to provide output to many types of output devices with one basic interface, the `device` task parameter. A full discussion of how IRAF sets up this interface is beyond the scope of this manual, however, this section provides a gentle introduction to the way in which **igi** interacts with the PostScript kernel.

psikern and igi

PostScript files are created whenever the output parameter `device` is defined as a PostScript device—either a printer or a PostScript file. The graphics commands used by **igi** to create the output are sent to the device specified by the `device` parameter for the task. IRAF then forwards those graphics instructions to the kernel specified for the device, which for PostScript output, is `psikern`. There are a few standard PostScript output devices distributed with IRAF, including `psi_land` for creating landscape-oriented (lengthwise) PostScript files and `psi_port` for creating portrait-oriented PostScript files. Other PostScript devices will be specified in the `graphcap` file applicable to the current session. PostScript files created by `psikern` will be written to a directory specified by the `tmp$` variable, which may be defined for your own IRAF configuration. The files will have file names beginning with "psk" followed by a unique number derived by IRAF, for example, `psk1234a`. This file can then be copied from the `tmp$` directory to a more permanent and logical directory and file name, or it can be simply previewed or printed.

The PostScript kernel contains an interactive task that can be run from the IRAF command line and used to interactively convert IRAF *metacode* (GKI) files into PostScript files: the task is called **psikern**. This task is used when the **igi** parameter `device`—or any other IRAF task output parameter—is specified as any PostScript device. Rarely would you ever explicitly run **psikern**, but sometimes its interactive parameters can be useful when working with color output.

In addition to specifying a PostScript output device, **igi** allows you to pass some commands directly into the PostScript output for modifying the text output. These capabilities will be discussed in more detail in "Text Annotation" on page 58.

Commands

Interaction with **igi** takes place through text commands that perform various functions. In some ways **igi** represents a sort of "mini-language" distinct from the IRAF cl. Some commands have arguments that modify the action. Arguments to the primary commands specify needed information such as coordinates or data file names. Some commands establish global parameters that modify the actions of other commands—such as changing the marker style or line width.

The following list describes some of the characteristics and features of **igi**'s language:

- ***Case independence***: Commands are *case independent*. That is, they may be in upper or lower case or any combination. Literal text strings, however, retain the case as typed. However, cl syntax, e.g., file names and escaped cl commands, *is* case-sensitive. Single-character cursor mode keystrokes and IRAF native cursor mode (e.g., `Z` or `:.snap`) commands are also case-sensitive. In this text, **igi** commands are printed in upper case. Macro names are presented in lower case. Command arguments are written in lower case and optional arguments are enclosed in square brackets.
- ***Minimum matching***: Commands may be minimum-matched. That is, you only need to type the minimum number of letters needed to distinguish uniquely the command from other commands. For example, the command `LIMITS` will be recognized if it is typed as `LIM` (or `lim`), but will be ambiguous if typed as `LI` since it would be confused with `LINES`.
- ***Multiple commands***: A line may contain more than one command, but multiple commands must be separated by a semicolon.

- ***Comments***: Any line beginning with the character # will be ignored and may be used as a comment. Any text following a # character will also be ignored. Commands typed interactively may be stored in a file and this command file may then be edited and subsequently used as input in the form of a command *script*.
- ***Piping and redirection***: Commands passed to **igi** as input may be typed interactively, redirected from a file, or piped from another application. Graphics output may be directed to the interactive graphics terminal, a hard-copy plotter, or a graphics *metacode* file.

Typing `END` or `BYE` (or the end-of-file marker in a command file) ends an **igi** session (terminates the interpreter). In interactive use, control returns to the cl. Commands redirected or piped from `STDIN` don't strictly require the `END` command; the end of file (`EOF`) is sufficient to terminate **igi**.

Redirecting Command Input

igi commands are read from the standard IRAF input stream (`STDIN`), which means that they can be typed interactively, piped from another task, or redirected from a file directly on the command line when **igi** is started. To direct a file containing **igi** commands to **igi** from the IRAF command line, use the < character followed by the name of the command file. For example, the following cl command would execute the **igi** commands in file `mycmd` just as if they had been typed interactively:

```
st> igi <mycmd
```

It is also possible to *pipe* the standard output (`STDOUT`) of another task into the command input of **igi**. In this case, there is no command interaction and control returns to the cl after **igi** terminates. This makes it possible to write a script or a task to create plots. The task needs only to write text to `STDOUT` using `printf()` statements (`print` command in the cl). For example:

```
st> mytask | igi
```

In either case, there is no interaction. No command prompt appears, and no interactive cursor operation is allowed.

Graphics output may be directed to the terminal or re-directed to a plotter or a file. When the graphics device is the terminal, command interaction (user input and single-line information and error messages) takes place in the status line (normally, a single scrolling line at the bottom of the screen). Text output consisting of more than one line, e.g., listing the command buffer, is directed to the text window. The `igi>` prompt in the status line indicates that **igi** is ready to interpret a command. Note that the specific operation of the graphics and text interaction depend on the type of terminal used for interaction.

Escaping to the cl

IRAF or host operating system commands can be passed outside of an igi session, this is called *escaping* commands. That is, you can type IRAF cl commands and host command from the **igi** prompt. To escape commands to the cl, precede the command with the `!` character. Because the cl recognizes the same character as an escape to the host system, commands can further be escaped to the host operating system with a second `!`.

For example, to get a listing of the current working directory, use:

```
igi> !dir
```

or if you prefer the Unix directory listing:

```
igi> !!ls
```

Command Classes

igi commands fall into one of five classes, distinguished by the action of the command.

- ***Execution***: Alters subsequent **igi** command operation or shows the current state. These commands are generally not saved in the command buffer (the command buffer will be explained in more detail on page 32).
- ***Data***: Reads input data into or modifies existing data in the **igi** data buffers. The `SHOW` command displays data buffer status.
- ***Attribute***: Sets or modifies a characteristic of a drawing command. Most attribute commands are saved in the command buffer. The `SHOW` command displays the current state of the attributes.
- ***Scaling***: Modifies or sets the transformation between various coordinate systems[3] including the data (WCS), viewport, virtual page (VPC), and device (NDC). Most scaling commands are saved in the command buffer but may not be undone. The `SHOW` command displays the current plot scaling.

3. Coordinate systems are described in more detail in the section "Coordinates and Scaling" on page 39.

- ***Plotting***: Draws a feature. Plotting commands draw visible features on the plot—text as well as graphic elements. Plot characteristics depend on attributes set using attribute commands or command arguments. Most of these commands are saved in the command buffer and the action usually may be undone with the `UNDO` command.

Table 4.1 on page 71 provides a summary of each **igi** command, including a brief description as well as the category, whether it is saved in the command buffer, if it may be undone with `UNDO`, and a page reference for a full command description in Chapter 4.

The commands listed in Table 2.3 deal with commands and the command buffer.

Table 2.3: Commands Dealing with Commands

Command	Action
`READ`	Read commands from a file
`INPUT`	Execute commands from a file
`WRITE`	Write commands to a file
`LIST`	Print commands
`EDITCMD`	Edit the command buffer
`DEFINE`	Define a macro
`END`	Terminate macro define mode or quit igi
`MACROS`	List macros
`PLAYBACK`	Execute commands in the buffer
`!`	Prefix to escape a command to the cl and host

Arguments

Commands are sometimes followed by positional arguments, separated from the command name and each other by white space (spaces or tabs) or a comma. They are always specified in a particular order, with no parameter keywords. Arguments may be optional. Optional arguments usually have a default value. In some cases, omitting an optional argument changes the behavior of the command.

Command Families

Many commands belong to a *family* of commands. Command families share the same syntax, but operate differently. For example, the `DRAW` command is really three commands: `DRAW`, `DDRAW`, `VDRAW`. They all draw a line, but operate in different coordinate systems. Another example is the `COLUMN` command that reads data into a plot buffer. This comprises six commands: `XCOLUMN`, `YCOLUMN`, `ECOLUMN`, `PCOLUMN`, `LCOLUMN`, and `SCOLUMN`, each filling a different plot vector buffer[4]. Command families are identified in this text with square brackets, e.g., `[XYEPLS]COLUMN`.

Commands for Working with Data Buffers

The following commands are grouped into families that work on the separate data buffers in **igi**:

- `[XYEPLS]COLUMN`
- `[XYEPLSZ]EVALUATE`
- `[XYEPLSZ]SECTION`

Commands for Working with Coordinate Systems

In addition, the following commands are grouped into families for working within the different coordinate systems:

- `DRAW, [DPV]DRAW`
- `MOVE, [DPV]MOVE`
- `RELOCATE, [DPV]RELOCATE`

Commands for Working with X,Y Plot Data

Finally, these command families are designed to work on both X and Y plot data:

- `[XY]FLIP`
- `[XY]LABEL`, `LABEL`
- `[XY]SIXTY`
- `[XY]LOGARITHM`

Complete descriptions of all commands and families are provided in Chapter 4.

Macros

A *macro* is a set of commands represented by a single command (like an abbreviation). Macro text is replaced in line, just as if the text had been typed in place of the defined macro command. Macros are defined using the `DEFINE` command followed by one argument—the name of the macro. The text of the macro is then entered as command input, but the commands

---

4. Data buffers are explained on page 35.

are not executed. Arguments are defined in the body of the macro and are replaced when the macro is invoked; the arguments are defined by positional macro arguments. If you define a macro while running **igi** interactively, the prompt changes to `macro>` indicating that subsequent text is not interpreted, but saved as macro text. Macro define mode is terminated by typing `END` at the prompt.

Macro Arguments

Arguments to macros are defined by the `&` character followed by an integer appearing in the macro text. The argument number specifies the position of the replacement argument value when the macro is invoked. The maximum argument value specifies the number of arguments to the macro.

A macro is invoked using its name, followed by the values to replace its positional arguments, if any. A macro can specify any **igi** command or even other defined macros, but it may not invoke itself (i.e., recursion is not permitted).

`BYE` is *not* equivalent to `END` for terminating a macro. Therefore, `BYE` does not terminate macro define mode, it will be stored as part of the macro text, and it will be expanded into the command stream when the macro is invoked.

Figure 2.1 lists commands that define and invoke a macro named “simple” to scale and draw a plot.

Figure 2.1: Creating and Using a Macro

```
igi> define simple
macro> data &1
macro> xcolumn &2; ycolumn &3
macro> limits; box; connect
macro> end
igi> simple igi.dat 1 2
igi> erase
igi> simple igi.dat 3 4
```

“DEFINE” Begins Macro Definition

“END” Terminates Macro Definition

“&” is Positional Argument

“;” Allows Multiple Commands on One Line

Invoking the Macro

Command Buffer

igi maintains a *command buffer* or memory of commands. This lets you execute commands either by *playback* or by saving the commands in a file (a *script*) for later execution. Not all commands are stored. In general, only those that directly influence plotted results are saved. Commands such as `HELP` or `SHOW` are not saved since they would clutter the output.

The buffer can be used in several ways:

- ***Playback*** command buffer using `PLAYBACK` command.
- ***List buffer*** (display commands) with `LIST` command.
- ***Edit buffer*** with `EDITCMD` command.
- ***Write buffer*** to file with `WRITE` command.
- ***Read command*** from file into buffer with `READ` command.
- ***Execute*** commands from file with `INPUT` command.

Commands are stored in their full, unabbreviated extent, capitalized, one command per line, regardless of how they were typed or read from a command file. String arguments (to `LABEL`, for example) are delimited by double quotes. The buffer operates differently for different classes of commands. In general, commands dealing with data I/O, data manipulation, plot parameters, and drawing commands are saved while those dealing with commands themselves are not. All of the graphics output commands and parameter (plot attribute) manipulation commands are saved in the command buffer. Command buffer manipulation commands are *not* saved. (See the complete description in Chapter 4 to determine if a particular command is saved.)

In cursor mode, a `RELOCATE` (`MOVE`) command is stored for each cursor position read, and commands are saved in the command buffer when entered in **:** mode. Macro invocations rather than the expanded text are stored in the command buffer.

Another command buffer contains a single command at a time. Only commands resulting in plot instructions are saved in this buffer. The `UNDO` command uses this buffer to erase the part of the plot just drawn. Not all devices support erasing.

Modes

Command interaction takes place in three modes:

- *Command* mode is the normal mode in which commands are entered and actions take place; it is indicated by the `igi>` prompt.
- *Macro define* mode enables macros to be created. Commands are not interpreted in macro mode. Macro mode is entered with the `DEFINE` command, ends with the `END` command, and is indicated by the `macro>` prompt.
- *Cursor* mode permits coordinates to be specified graphically with the cursor. Cursor mode is entered using the `CURSES` command and is indicated by the activity of the graphics cursor.

Undoing a Command

The action of certain commands may be *undone* under some circumstances. Some graphics devices support drawing clear lines, that is, with the background color rather than the foreground color normally used to draw. The **igi** `UNDO` command will do this on supported devices.

Input Plot Data

igi can use any plot data format supported by IRAF, including:

- ***Text (ASCII) tables:*** including comments (`#`) and blank lines.
- ***Binary STSDAS tables***: columns specified by name.
- ***FITS tables:*** columns specified by name, also uses row selectors for tables which have arrays as each element of a column (i.e., 3-D tables).
- ***IRAF images:*** including image section and FITS images with extension specification.

Subsets of input data files may be specified, such as a range of rows in a text file or table, or a section or of an image file. If an input image file contains world coordinate information, it may be used to construct the independent variable axis. The data read subsequently may be modified using expression evaluation syntax (see "Expression Evaluation" on page 46).

Any data type (integer, double precision, etc.) may be in the external file but when read will be converted to single-precision floating point values.

Points with either coordinate having the value `indef` will be excluded from autoscaling, and will not be plotted.

Text files used as input consist of columns of values separated by white space (spaces or tabs). Rows of input are restricted to a single line of the input file. Column values need not line up row to row.

Any blank line or line beginning with a `#` is ignored when counting the lines in the file and may be used as a comment.

FITS tables can now contain arrays as elements of a column, which are called *3-D tables*. These tables can be read by **igi** using one of two mechanisms: appending a row selector to the input data name or specifying a particular row number when defining the data column to be read. In the latter case, the array in the specified row of the column will be read in as an input array and treated as a normal data array within **igi**.

Internal Data Buffers

Six buffers are predefined for plotting data. Data from any recognized format may be read into any buffer, but they are treated somewhat differently. The buffers are used to hold the following data types, with the buffer designation used for the commands given in brackets:

- ***X and Y plot coordinates*** [X Y].
- ***Errors*** [E].
- ***Point style codes*** [P].
- ***Upper or lower limit flags*** [L].
- ***Scratch buffer*** [S].

These buffers can be manipulated by the commands listed in “Command Families” on page 30. The `DATA` command specifies the name of the table file to read. Columns from either format are read into the buffers using the separate explicit `[XYEPLS]COLUMN` commands. The data to be read from ASCII tables are specified by the column number appended to a “c”, e.g., the third column would be specified as `c3`. Data from STSDAS or FITS tables are specified by column name. Note that table column names are case-sensitive. The same commands are used to read text and table files; the file format itself determines the command syntax and semantics.

IRAF Images

igi recognizes the IRAF data formats shown in Table 2.4.

Table 2.4: IRAF Image Types Used by igi

Format	Type	Extension
Original IRAF	OIF	`.imh`
FITS	FITS	`.fit,.fits`
STSDAS (GEIS) including groups	STF	`.hhh`
PROS (ROSAT x-ray events)	QPOE	`.qp`

Image data are handled differently from text and binary tables. The command `[XYEPLS]SECTION` opens an image file and reads data into one of the **igi** buffers. Neither the `DATA` nor `COLUMN` commands is used. An *image section* may specify some portion of the image to read. In addition, an optional qualifier argument after the image name specifies how to treat multi-dimensional images. The qualifier must be an integer. If the qualifier is zero, the image section is read as a single one-dimensional vector regardless of the dimensionality of the data. If the argument is non-zero, its value specifies the axis on which to project the data to form a single vector. If positive, **igi** will average the data perpendicular to the specified axis. If negative, **igi** will sum instead of average. For example:

```
igi> YSECTION myimage 2
```

will average the columns (axis 2) in `myimage` to form a single vector to fill the Y plot buffer. Images of any data type may be read. `indef` values will be ignored except when projecting a multi-dimensional image section.

Greyscale Images

A separate buffer accepts true two-dimensional, greyscale images, sometimes known as *rasters* or *pixmaps*. The `ZSECTION` command fills the Z buffer from an image section. As with the other `SECTION` commands, it is possible to read an image section from any image format recognized by IRAF, but the Z buffer retains information about the dimensions of the raster. In addition, there are other commands that deal with the manipulation and display of this buffer, distinct from the line-plotting commands.

Implementation Highlights

In summary, **igi** incorporates a custom command line interpreter to parse commands and arguments, thus defining a “mini-language” within—but distinct from—the IRAF command language (cl). It recognizes all IRAF data types, using IRAF’s general file I/O facilities (fio). All graphics output is through the IRAF graphics I/O (gio) and PostScript kernel (psikern), as needed. A user can operate on internal data vectors with arbitrary expressions. Fonts are implemented using a custom host-independent subset of the Hershey fonts.

All graphics instructions in **igi** are implemented using the IRAF SPP graphics I/O (gio) library procedures. Extensive use is made of gio’s multiple world-coordinate system (mwcs) capabilities. Additional features include multiple I/O streams on a stack for command input, internal *memory files* for stored commands, and pushback to the input stream to implement user-definable macros.

Summary of igi Features:

- Custom command line interpreter.
- Implements macros using pushback to input stream.
- Uses **evexpr()** for expression evaluation.
- Makes use of internal memory files.
- Multiple I/O streams on a stack for `INPUT` etc.
- Spawns cl tasks such as `help` and `page`.
- Uses IRAF gio (graphics) and mwcs (world coordinate system) capabilities.
- Uses PostScript kernel (psikern) to provide flexibility in formatting the output text.
- Makes uses of fitsio and other IRAF interfaces to work with any data type recognized by IRAF, including FITS tables and images.
- Additional capabilities depend on kernel and device implementation. Color, greyscale, and filled areas work with the PostScript kernel and PostScript devices.

CHAPTER 3:

Features of igi

In This Chapter...

This chapter explains most of the features of **igi** and describes its commands. In addition, complete information on font selection, character sets, coordinate systems and other reference material associated with **igi** commands will be provided.

Coordinates and Scaling

Specifying location of text and graphics on a page within igi may be done using any of three coordinate systems:

- World (User) Coordinates: WC
- Virtual Page Coordinates: VPC
- Normalized Device Coordinates: NDC
- Physical Coordinates: Inches

These coordinate systems each represent the page in a different way, and the drawing commands discussed in "Command Families" on page 30 are designed to take advantage of these systems.

Coordinate Definitions

World Coordinates

Coordinates for most commands (pen position, cursor readback, etc.) are specified in *world coordinates*, or WC, also known as *user coordinates*. These coordinates are based on the input data scaled to the page. Commands that use the WC refer to positions relative to the limits of the data being displayed.

Normalized Device Coordinates

Normalized device coordinates refer to the entire page as having a range of 0:1 in both the X and Y dimension. This serves as the basis for the placement and specification of all the sections to defined for the plot on the page. In addition, a transformation between NDC and actual physical size in inches can be specified using the `PHYSICAL` command.

Virtual Page Coordinates

A *virtual page* may be specified as a portion of the physical device drawing area and spans a range of 0:1 in both X and Y. Using the `VPAGE` command, or in physical units (inches) using the `PHYSICAL` command, the boundary for the virtual page can be set in terms of normalized device coordinates (NDC).

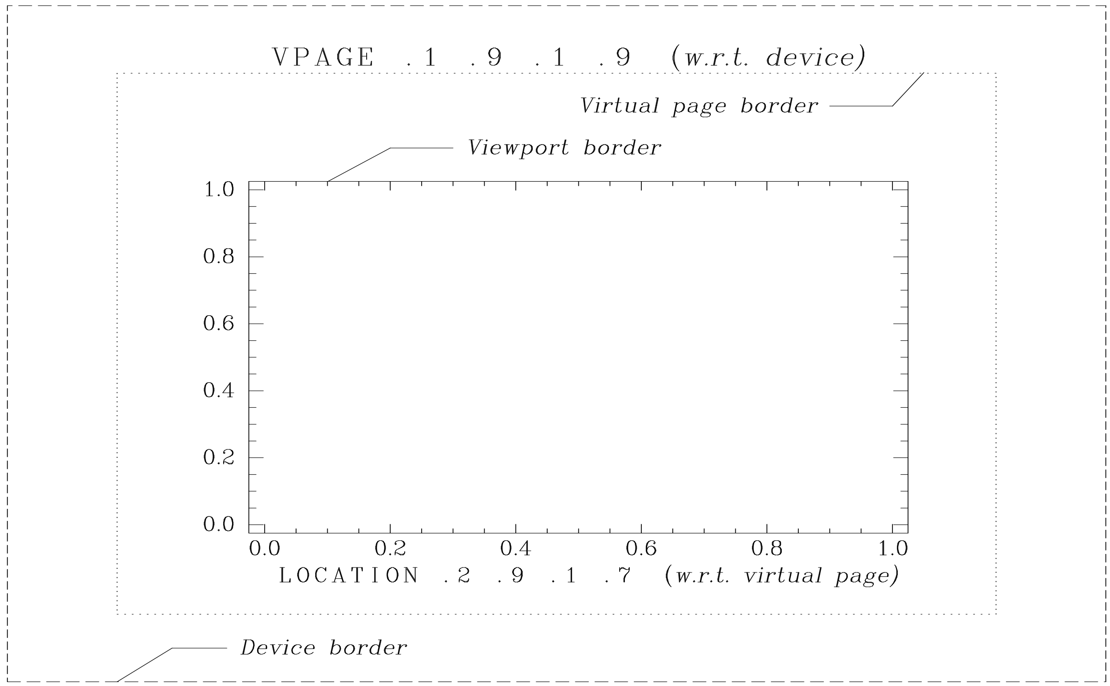
Plot Viewport

The *viewport*, i.e., the axes of a plot, may fall anywhere within this page. It can be specified as a fraction of the virtual page using the `LOCATION` command.

Table 3.1: Commands Specifying a Coordinate System Transformation

Coordinate Systems	Command
WC $\leftrightarrow$ VPC	`LIMITS`
VPC $\leftrightarrow$ NDC	`VPAGE`
NDC $\leftrightarrow$ Inches	`PHYSICAL`

Figure 3.1: Coordinate System Overview



Specifying Windows on the Page

Another way to specify a virtual page is with the `WINDOW` command. This lets the display space be *tiled* into as many as 15 adjacent *windows* or *panes* whose edges do not overlap. This is done by implicitly setting the **igi** virtual page coordinates. You need only specify the number of panes in the windows as some number horizontally and vertically. The `WINDOW` command selects one of the panes as the current virtual page. To select another window in the current tiling, specify *only* the pane number *without* re-specifying the tiling. (It is possible, of course, to reset the tiling as well, in which case, you may get overlapping windows.)

Figure 3.2 shows some examples of the `WINDOW` command. See Appendix A for the **igi** source for this example. Note that there are two different tilings in this example.

Figure 3.3 shows an example of drawing a grid in inches.

Figure 3.2: Virtual Pages and Windows

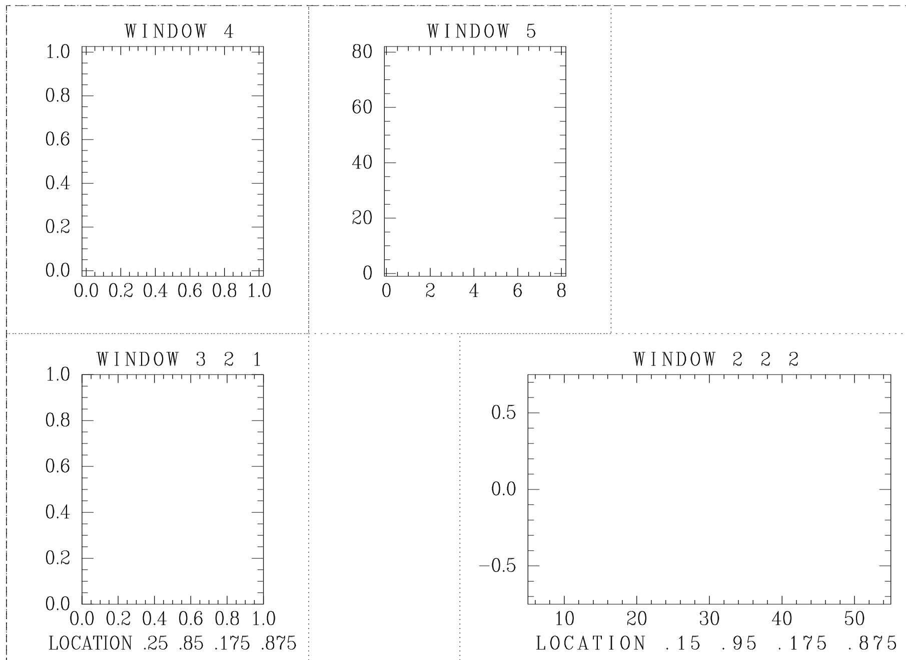
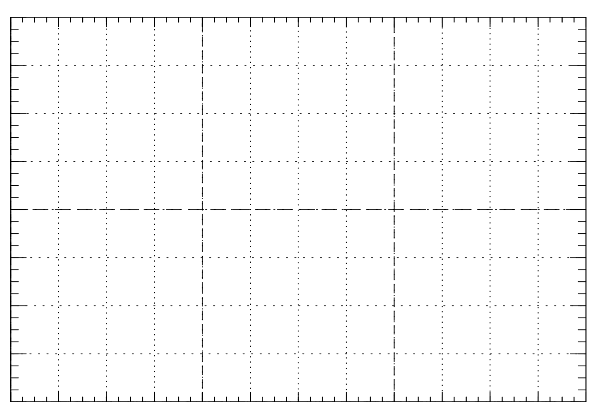


Figure 3.3: Absolute Scaling with PHYSICAL

Commands to Draw Grid

```
LOCATION 0 1 0 1                       <- Viewport on Full Vpage
PHYSICAL 0 3 0 2                       <- Make Grid 3 x 2 Inches
LIMITS 0 3 0 2                         <- Scale Mateches Vpage in Inches
TICKSIZE .125 1 .125 1                 <- Minor 1/8", Major 1" Tick Spacing
BOX 0 0                                <- Draw Axes with no Tick Labels
LTYPE DASHED; GRID                     <- Draw Grid with Dashed Lines
TICKSIZE .0625                         <- Minor 1/16" Major 1/4"
BOX 0 0 ; LTYPE DOTTED ; GRID          <- Draw Finer Grid with Dotted Lines
```

Resulting Grid



The plot scale for a specific set of axes is determined by the user coordinates associated with the viewport. These values are set using the `LIMITS` command. This command associates WC (user coordinate) values with each edge of the viewport and defines the transformation between the user and device coordinates. The `LIMITS` command can be used to explicitly specify the coordinate scale, or autoscale based on the current X and Y plot vector data values. When autoscaling, the edges of the world coordinates will be at the minimum and maximum data value of the X and Y arrays and, therefore, one or more plotted data points will fall on an axis. To add a border between the plotted curve or points and the axes, either explicitly specify the edges of the user coordinates with the `LIMITS` command, or use the `MARGIN` command to automatically include an equal border between the edges of the data and the axes. The default margin is 2.5% of the width of the viewport (axes). However, the `MARGIN` command has an optional argument to specify the border as a fraction of the size of the viewport.

You can use world coordinates from an input image as the X values that correspond to Y values input from the image. This can be done with the `YSECTION` command if the appropriate parameters exist in the image header. By default, these coordinates will not be used and you must supply the appropriate independent variable as the X values. However, if you use the `IMGWCS` command and the image contains the appropriate transformation parameters, then `YSECTION` will automatically fill in the X plot buffer with the coordinate values. Note that, in general, this works only for inherently one-dimensional data, e.g., spectra. For 2-D data such as images, coordinates are rather meaningless for an arbitrary 1-D section of the image. You will get an *identity* vector if the coordinate parameters are not present in the image. That is, the X values will be the element number of the extracted pixels. This is not exactly the same result as not using the WCS, since the coordinate transformation is applied to the image section. That is, the returned coordinates are the coordinates with respect to the original full image, not the extracted piece. For example, if you use:

```
igi> imgwcs
igi> ysection dev$pix[10,50,1]
```

the range of the X values will be 10 to 50 rather than 1 to 41 if you do not use `IMGWCS` and do not otherwise fill in the X vector. Even if the image does not contain a WCS transformation, `YSECTION` will provide an X vector that corresponds to the pixel numbers in the full input image.

The commands that require a plot position (move and draw) have versions for each of the coordinate systems:

Table 3.1: Pen Positioning Commands in the Coordinate Systems

Coordinates		Pen Up Move	Pen Down
World (User)	WC	`RELOCATE` `MOVE`	`DRAW`
Normalized Device	NDC	`DRELOCATE` `DMOVE`	`DDRAW`
Virtual Page	VPC	`VRELOCATE` `VMOVE`	`VDRAW`
Viewport	PC	`PRELOCATE` `PMOVE`	`PDRAW`

The current X and Y (logical pen) position is maintained in user (world) coordinates. Several commands change this position and others depend on its value.

Drawing

The usual order of events when drawing plots is:

1. Read data.
2. Scale: determine or set the limits of the data.
3. Draw axes.
4. Draw the curve or points.
5. Annotate with text.

The following sections describe these steps in greater detail with examples.

Reading Data

Commands for handling data are listed in Table 3.2. Use `DATA` to specify the text or binary table to read. `[XYEPLS]COLUMN` reads data from a text or table column into a plot buffer. Read an IRAF image section with `[XYZEPLS]SECTION`. Use `IMGWCS` to fill in the X plot buffer with the world coordinates of the image.

Table 3.2: Commands to Read, Write, or Modify Data

Command	Description
`[XYEPLS]COLUMN`	Read a text or binary table column into a vector
`DATA`	Specify the text or binary table file name to read
`DLIST`	Print the data values
`[XYZEPLS]EVALUATE`	Perform arbitrary vector arithmetic operations on data
`IMGWCS`	Fill independent variable with image world coordinates
`LINES`	Restrict the range of table rows to read
`[XY]LOGARITHM`	Compute the log of a data buffer
`[XYZEPLS]SECTION`	Read an image section into a vector

Reading Data from Tables

Tables within IRAF can take on a number of different forms, including STSDAS tables, FITS tables, and FITS 3-D tables. STSDAS tables and normal FITS tables can be easily read into **igi** by specifying the column names in the `XCOLUMN` and `YCOLUMN` commands. However, FITS 3-D tables—tables which contain arrays as elements of a column—require special treatment in order to extract individual arrays for plotting.

A couple of different mechanisms within **igi** accommodate 3-D tables: an extra row parameter for the `[XY]COLUMN` commands, and the use of row selectors in specifying the file name for the `DATA` command. These two mechanisms can be used exclusive of each other or in conjunction depending on the nature of the data and plot.

Row Selector Usage with the DATA Command

This method lets you specify the exact row to be plotted by using the row selector syntax when specifying the name of the data file. The row selector syntax specifies how a row or range of rows (or columns) can be selected for use from any table. The general row selector syntax is:

`table.fits[`*`extn`*`][r:`*`row`*`=`*`min#`*`:`*`max#`*`]`

In the above row, we specify that rows *`min#`* to *`max#`* be read from extension *`extn`* of the FITS table `table.fits`. Since 3-D tables contain arrays as each row element of a column, only a single row can be read in at a time. A row selector can then be appended to the filename in the `DATA` command to specify the row number within the columns from which to extract the array.

Row Parameter Usage with the COLUMN Command

A simpler approach, but perhaps not as flexible, uses an optional row number parameter with the `[XY]COLUMN` commands. If the row number is specified with the first `COLUMN` command read by **igi**, then it is assumed that a 3-D table will be accessed and array input data will come only from the specified row.

▽ If the row number is specified for only *one* of the `COLUMN` commands, it must be the *first* such command in the script. In this case, it applies to both `[XY]COLUMN` commands. If the first `COLUMN` command does *not* contain a row number and no row selector was specified, then only the first element from each column will be plotted against each other, regardless of what the subsequent column commands had specified.

Each command, however, could have a row number specified, and not necessarily the same row number, without causing any errors. If different rows were specified from different columns, they would be plotted against each other to the extent that they have the same number of elements.

Combining Both Methods

If a row selector is specified in the `DATA` command that spans a range of rows, individual rows can be plotted by referencing them with the extra row parameter each time the `COLUMN` commands are used. In this case, the row parameter references the row number *relative* to the range specified in the row selector. For example, if a row selector of `[r:sporder=45:60]` was specified in the `DATA` command, spectral order 46 would be plotted by specifying:

```
XCOLUMN wavelength 2
```

not

```
XCOLUMN wavelength 46
```

Expression Evaluation

Data present in the plot buffers may be modified using the `[XYEPLS]EVALUATE` or `[XY]LOGARITHM` commands. The normal range of Fortran style operators and functions are available, including:

- ***Basic math:*** addition, subtraction, multiplication, division, exponentiation, log10, log, sqrt
- ***Conditionals and Booleans:*** ?, :, <, >, =, ==, !=, <=, >=, &, |, !

- ***Trigonometric Functions***: cos, sin, tan, acos, asin, atan, atan2
- ***Statistical functions:*** min, max, mod, abs

Do not use expression evaluators on `indef` values or **igi** may crash. These values are not recognized as `indef` by the evaluator, but are treated as normal data values.

Variables representing the internal buffer arrays are recognized in expressions and referred to by a single character, i.e., X, Y, E, P, L, and S. You may also refer to the element number using the variable `r` and the number of elements in the vector by using `n`. Note that the result of the expression replaces an existing vector. There is no assignment operator—the command name specifies the destination vector in the assignment. The following commands would give the same result, taking the logarithm of the Y vector:

```
igi> YEVALUATE log10(y)
```

or

```
igi> YLOGARITHM
```

Scaling

The plot scale is set by the `LIMITS` command. Explicitly specifying four arguments sets the world coordinates of the edges of the viewport. In essence, the `LIMITS` command maps the coordinates of the plot, either the X and Y ranges or the number of pixels in each dimension of an image, into relative positions on the page. The relative positions specified in terms of virtual page coordinates (coordinates which simply range from 0 to 1 across the plotting area). Therefore, the plot or image is scaled to fit within the specified (or default) plotting area regardless of the output device. Furthermore, once this mapping has been done by `LIMITS`, positions within the plot can be referred to by coordinates specific to the plot ranges or image size. For example, if you have an 800 x 800 image displayed on the page, the center is at (X,Y) = (400,400), rather than a relative page position such as (0.625, 0.375).

If there are no arguments, `LIMITS` automatically scales (*autoscales*) the plot to the data in the Y coordinate buffer if there are no values in the X buffer or the X and Y buffers, if both exist.

Table 3.3 describes several commands that can be used to work with scaling.

Table 3.3: Commands to Set or Modify Scaling

Command	Description
XYFLIP	Flip axis horizontally or vertically
LIMITS	Set plot scale (WC to VPC)
LOCATION	Place viewport (axes) on virtual page
MARGIN	Adjust scale to allow space between data and axes
PHYSICAL	Place virtual page in inches
VPAGE	Place virtual page on device
WINDOW	Tile device with virtual pages

Log-Scaling the Data

Sometimes a data set can not readily be plotted using a linear scale for the axes. In these cases, you can operate on the data in the X and Y buffers using the [XY]LOGARITHM commands. These commands will take the log of the data read into the data buffers. The command TICKSIZE then controls whether linear or log axes are used for the actual plot. Figure 3.4 shows an example of log-scaling and plotting data.

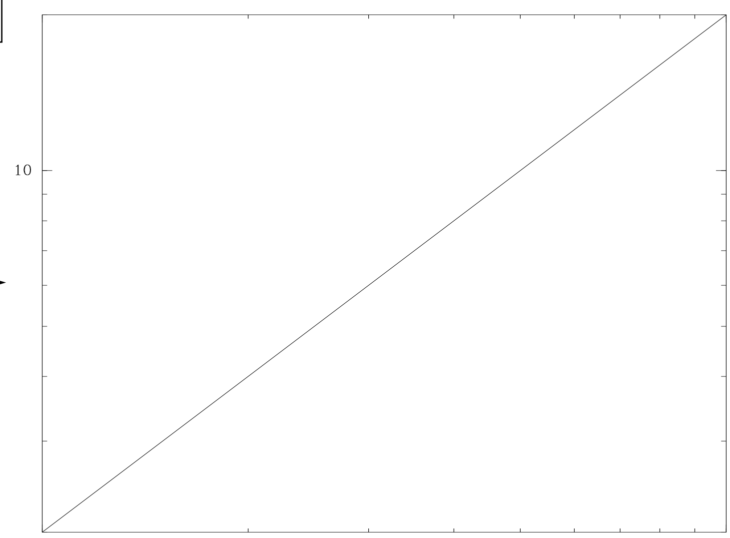
Figure 3.4: Log Scaling Data

Script to Scale Data

```
igi> data igi.dat
igi> xcolumn 1; ycolumn 2
igi> xlogarithm; ylogarithm
igi> limits
igi> ticksize -1 0 -1 0
igi> box; connect
```

XLOGARITHM and YLOGARITHM Scale the Data

Resulting Log-Log Plot



Data Curves

Data points can be plotted individually—one by one—using MOVE and DRAW or the DOT command to plot a centered symbol at the current pen position.

Alternatively, a curve can be plotted from the data in the Y (and optionally the X) buffer using one command. Data in the plot buffers can be plotted in several styles. A complex plot can be built using several different curve styles. Data points can be connected by straight lines or with *stepped* horizontal segments connected by vertical lines. In addition, a bar graph[1] may be drawn, with the vertical segments dropped to the bottom axis. Symmetrical or asymmetrical error bars, upper or lower limit symbols, or a centered symbol may be drawn at each data coordinate.

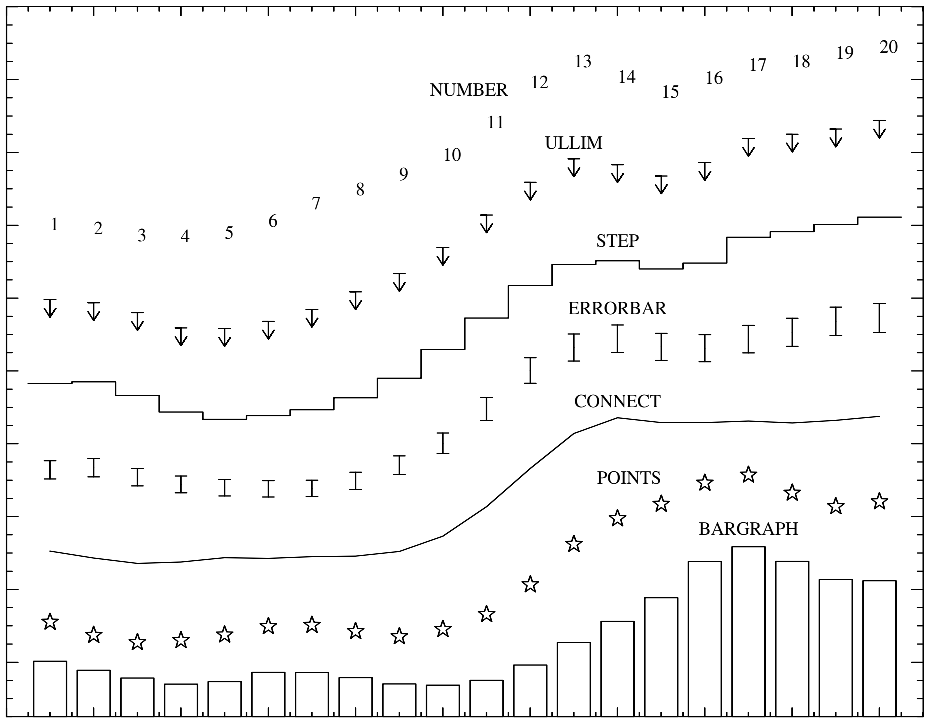
Table 3.4 lists commands that can be used to draw curves and points. Figure 3.5 shows examples of the output produced by these commands.

Table 3.4: Commands that Draw Curves or Data Points

Command	Description
BARGRAPH	Plot bar graph
CONNECT	Connect data points
DOT	Draw a marker at current pen position
ELLIPSE	Draw an ellipse at current pen position
ERRORBAR	Draw error bars
HISTOGRAM	Plot bar graph
NUMBER	Draw the index of each point
POINTS	Draw unconnected markers at each data position
ULLIM	Draw upper or lower limit symbols

1. Sometimes, confusingly referred to as a histogram-style plot.

Figure 3.5: Types of Plotted Curves and the Commands that Produce Them



Error Bars

Sometimes error data exist for the data to be plotted and you may want to plot the data and superimpose symmetrical error bars. Figure 3.6 shows a script that plots error bars for Y data. Note that the error values are read with the `ECOLUMN` command and you need to call `ERRORBAR` twice to get both halves of the symbols. `ETYPE` specifies the style of error marker (in this case, a standard bar with a tick at the end). The two calls to `ERRORBAR` specify that the error bars should be drawn in both the Y and –Y directions.

Figure 3.6: Drawing Error Bars

```
igi> data tdat
igi> ycolumn 2
igi> ecolumn 3
igi> limits
igi> etype 1
igi> errorbar  2
igi> errorbar -2
```

ECOLUMN Reads Error Values

ERRORBAR Must be Called Twice

Simple Plotting

The most basic plotting functions are *move* and *draw* providing complete control of the *pen*. The **igi** `MOVE` and `DRAW` commands do just this. **igi** provides separate move and draw commands for each coordinate system. Table 3.5 lists the move and draw plotting primitive commands for each coordinate system. Note that there are two equivalent aliases for the `MOVE` command.

Table 3.5: Coordinate Systems

System	Pen Up Move	Pen Down
WC	`RELOCATE MOVE`	`DRAW`
NDC	`DRELOCATE DMOVE`	`DDRAW`
VPC	`VRELOCATE VMOVE`	`VDRAW`

The commands in Figure 3.7 will draw a border around the edge of the device (window or page).

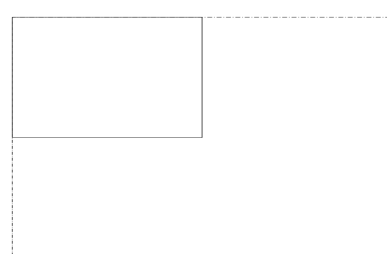
Figure 3.7: Draw a Box in NDC

```
igi> DMOVE 0 0
igi> DDRAW 1 0
igi> DDRAW 1 1
igi> DDRAW 0 1
igi> DDRAW 0 0
```

The example in Figure 3.8 sets a virtual page to the upper left of the screen and draws the border around the virtual page. This example demonstrates simple moves and draws in the various coordinate systems.

Figure 3.8: Commands and Plot of a Border in VPC

```
VPAGE 0 0.5 0.5 1
VMOVE 0 0
VDRAW 1 0
VDRAW 1 1
VDRAW 0 1
VDRAW 0 0
```



Attributes of Plotting Primitives

Several commands set *global parameters* that control the results of other commands. These commands, parameters, the attributes that they control are briefly listed in Table 3.6 and are further described below.

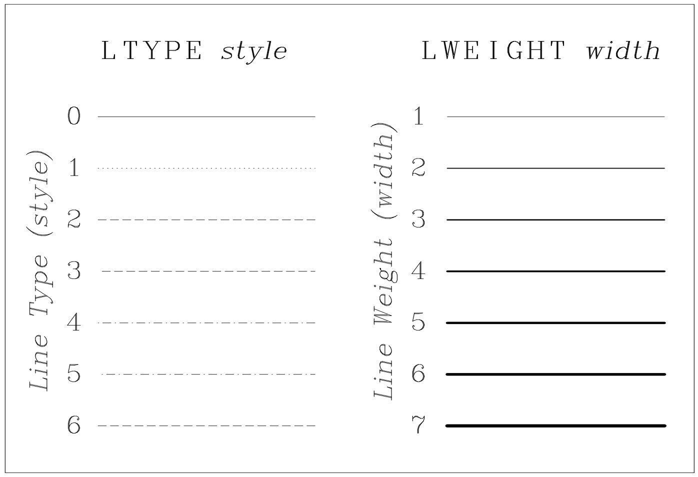
Table 3.6: Commands for Setting Global Attributes

Command	Parameter	Applies to Commands
`EXPAND`	Size	`DOT ELLIPSE ERRORBAR LABEL POINTS PUTLABEL ULLIM`
`ANGLE`	Orientation (rotation)	`DOT ELLIPSE LABEL POINTS PUTLABEL`
`ETYPE`	Error bar type	`ERRORBAR`
`LTYPE`	Line style	`CONNECT DRAW GRID HISTOGRAM STEP`
`LWEIGHT`	Line width	`AXIS  BOX CONNECT DOT DRAW ELLIPSE HISTOGRAM LABEL POINTS PUTLABEL STEP`
`PTYPE`	Marker style	`DOT POINTS`
`JUSTIFY`	Justification	`LABEL`
`FILLPAT`	Fill area pattern	`DOT, POINTS, BARGRAPH`
`COLOR`	Drawing color	All drawing commands including text

- `EXPAND` specifies the size of text, markers, etc. The value of the argument is relative to a default size, based on the fundamental character size for each device (defined in the IRAF `graphcap`[2] file).
- `LTYPE` selects one of the available line styles. This may differ somewhat from device to device, but in general, most devices will support the common styles: *solid*, *dashed*, *dotted*, and *dot-dash*. In some cases, there may also be a *clear line* style, which erases rather than draws. See Figure 3.9 for examples of line styles.
- `LWEIGHT` specifies the line width for all subsequent lines relative to the fundamental line width. This applies to all drawn entities including curves, symbols, and text. See Figure 3.9 for examples of different line weights.
- `ANGLE` controls rotation and is specified in degrees relative to the +X (horizontal) axis.

2. The means of specifying attributes of terminal and printer graphics devices recognized by IRAF.

Figure 3.9: Line Styles and Widths



- PTYPE is the point style. See Figure 3.10 for examples of the PTYPE command demonstrating the different point marker styles.
- COLOR depends on the graphics device and the kernel used to drive the device. The color is specified as an index into a table of possible colors. See Figure 3.11 for an example of available colors.
- FILLPAT selects the fill pattern—this is device- and kernel-dependent. Fill patterns apply to bar graphs and symbols plotted with DOT and POINT. Examples of the fill patterns can be found in Figure 3.10.

Figure 3.10: Point Types and Fill Patterns

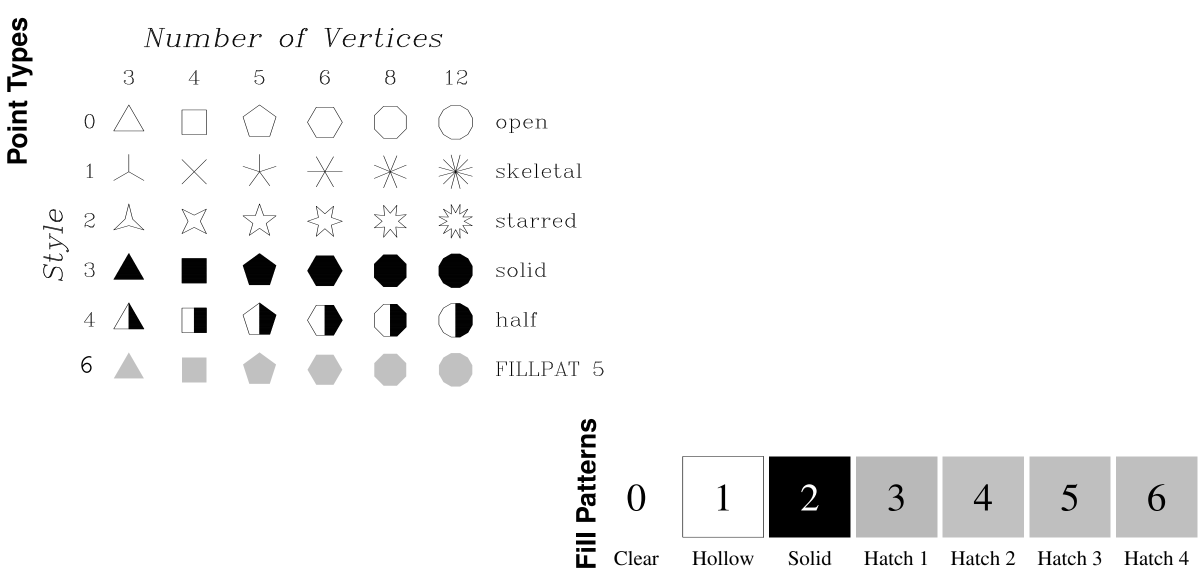
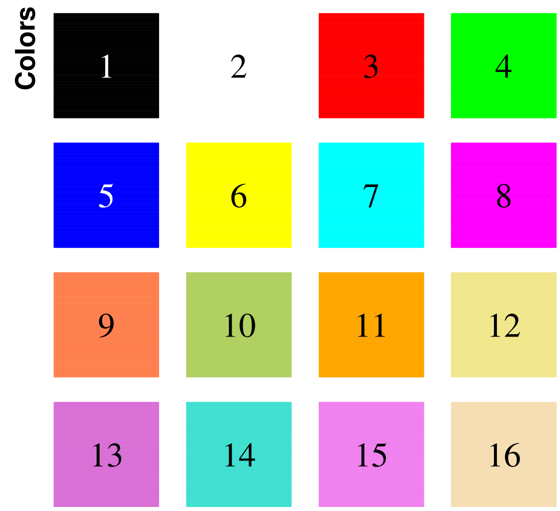


Figure 3.11: COLOR Values

Index	Color
1	Black
2	White
3	Red
4	Green
5	Blue
6	Yellow
7	Cyan
8	Magenta
9	Coral
10	Maroon
11	Orange
12	Khaki
13	Orchid
14	Turquoise
15	Violet
16	Wheat



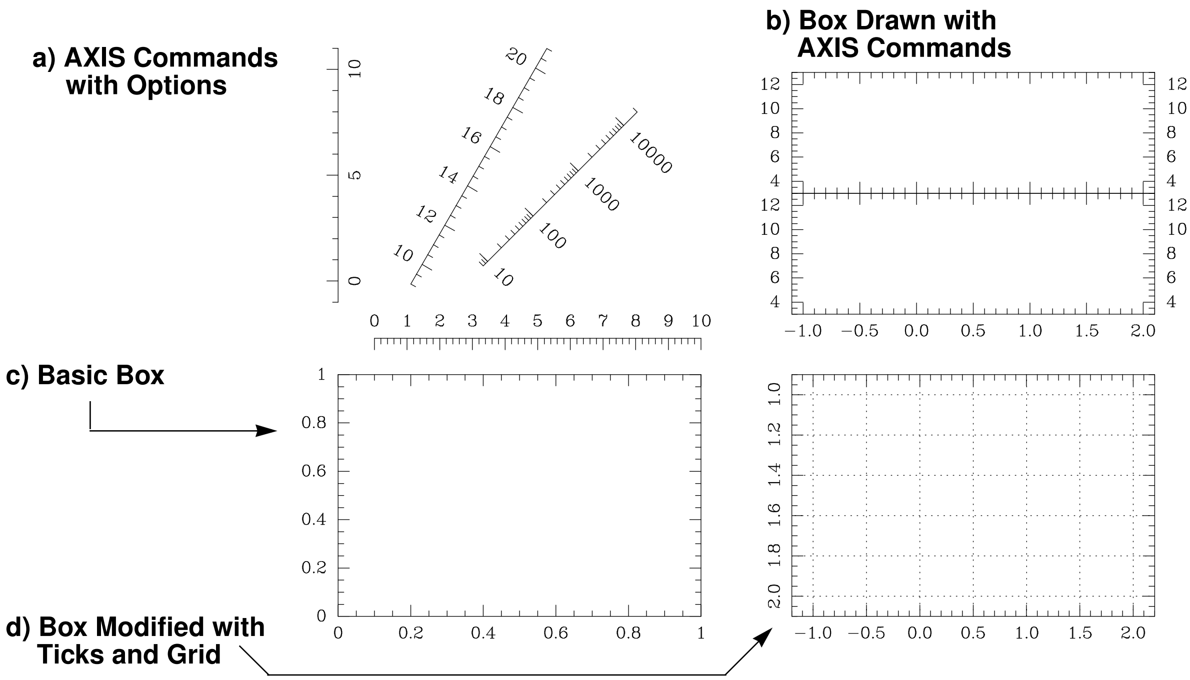
Axes

Several commands let you plot data axes and control their appearance. These commands are listed in Table 3.7. Two commands draw axes: `AXIS` and `BOX`. `AXIS` is lower level and quite general while `BOX` is more convenient but has fewer options. `BOX` automatically draws four axes at the current viewport boundaries using the current world coordinates. The four plots in Figure 3.12 show examples of various styles of axes. Each panel is explained in more detail, as are the `AXIS` and `BOX` commands.

Table 3.7: Axis and Labeling Commands

Command	Result
`AXIS`	Primitive axis—command specifies all attributes
`BOX`	Complete axes with labels
`GRID`	Connect major ticks
`TICKSIZE`	Major, minor tick spacing, Logarithmic labels
`NOTATION`	Change tick label exponential format limits
`XLABEL`	Draw X (horizontal) axis label (title)
`YLABEL`	Draw Y (vertical) axis label (title)
`TITLE`	Draw plot title

Figure 3.12: Examples of Axes



The top two panels in Figure 3.12 illustrate uses of the `AXIS` command, while the bottom two illustrate uses of the `BOX` command.

- Panel **a** (top left) demonstrates various `AXIS` commands and their options. The panel is drawn using the commands in Figure 3.13.

Figure 3.13: Individual `AXIS` Commands

```
ANGLE 0
AXIS 0 10 0.2 1 .1 .1 .8 1 1
ANGLE 90
AXIS −1 11 1 5 .1 .2 .7 1 0
ANGLE 60
AXIS 9 21 0 0 .25 .25 .75 2 1
ANGLE 45
AXIS 8 2e4 −1 0 .4 .3 .6 2 0
```

- Panel **b** has a full set of axes drawn individually using `AXIS` since `BOX` does not draw tick labels on the right axis.

- Panel **c** is drawn by the `BOX` command using only the default parameters—no options. Note the orientation and format of the tick labels.

- Panel **d** is drawn by the commands in Figure 3.14. The TICKSIZE command specifies the *spacing* between minor and major ticks in X and Y in WC. The arguments to the BOX command specify that the tick labels should be drawn parallel to both axes. The remaining two commands fill in the grid between the major ticks using dotted lines.

Figure 3.14: Attributes of BOX Command

```
LIMITS -1.2 2.2 2.1 0.9
TICKSIZE .1 .5 .05 .2
BOX -1 -1
LTYPE dotted
GRID
```

BOX: Complete Borders with Axes

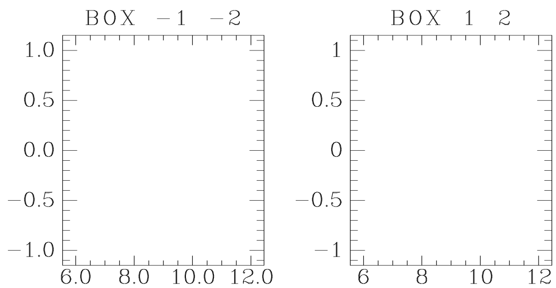
BOX does not label any axis but it does label the major ticks along the bottom and left axes. The XLABEL and YLABEL commands allow any text string label to be placed below the bottom horizontal (X) axis and to the left of the left vertical (Y) axis, respectively. To draw any other style of axis label, use the LABEL command. There are two string buffers maintained for the X and Y labels. These are filled by the [XY]COLUMN and [XY]SECTION commands with the file names used as input data. If the argument to [XY]LABEL is a null string, then **igi** will use the label strings. Otherwise, it will use the text string specified in the argument.

Optional arguments to the BOX command specify how to draw the axis labels. There are two additional commands that alter the way in which the axes are drawn: TICKSIZE and NOTATION.

TICKSIZE specifies the spacing between major and minor ticks and optionally specifies logarithmic axes. NOTATION specifies the range of data values within which tick labels will be written in exponential notation.

Figure 3.15 shows the difference between positive and negative arguments to BOX resulting in having leading and trailing zeros plotted or suppressed.

Figure 3.15: Leading and Trailing Zeros in Axis Labels with BOX

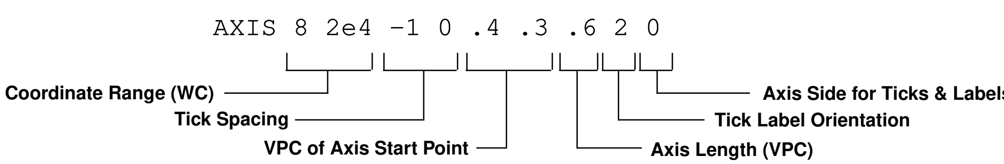


AXIS: Single, Configurable Axis

`AXIS` is a lower-level command that draws an arbitrary axis. It does not take into account the current viewport and world coordinates. Arguments to `AXIS` specify the position of the axis, its range of data values, the format of the labels and spacing of ticks. The current angle set by the `ANGLE` command determines the orientation of the axis. See Table 4.5 on page 80 for the interpretation of the label argument and Table 4.3 for an interpretation of the clock argument. The `ANGLE` command must be used separately to specify the angle of the axis.

Figure 3.16 shows the last `AXIS` command from the script displayed in Figure 3.13. The nine parameters to this command are explained below.

Figure 3.16: AXIS Parameters



- The first two arguments specify the range of data coordinates (WC) at the ends of the axis.
- The next two arguments specify the spacing between major and minor ticks. The negative value specifies logarithmic tick labeling.
- The next two are the virtual page coordinates of the starting end of axis.
- The next is the length of the axis in VPC.
- The next specifies whether the tick labels will be omitted, or plotted parallel or perpendicular to the axis.
- The last specifies on which side of the axis the ticks and labels will appear.

To fill in a grid of lines between major tick marks on axes drawn by the BOX command, use the GRID command. It uses the current line type (see LTYPE) and width (see LWEIGHT). You must use BOX before GRID to compute the tick spacing. TICKSIZE specifies the spacing in world coordinates of major and minor ticks for axes drawn by BOX. It also specifies whether to label axes linearly or logarithmically. FMTICK specifies the print format for tick labels as a Fortran-format string. This overrides exponential notation in tick labels. NOTATION changes the range of data beyond which ticks are labeled in exponential format. XLABEL draws a text label below the X (horizontal) axis. This might have been filled in by XCOLUMN as a table column name. YLABEL draws a text label left of the Y (vertical) axis. This might have been filled in by YCOLUMN as a table column name or using IMGWCS and YSECTION from the image file name.

Text Annotation

igi has flexible text drawing facilities, specifically for labeling axes and plot titles as well as for arbitrary text. Some text-handling functions are handled by **igi** commands which set global parameters, while others are handled using escape sequences within text strings. The escape sequences are preceded by the \ character, for example, a Greek character can be specified by the \g escape sequence.

Text-handling commands and escape sequences are discussed more in the next few pages, followed by a section explaining how to use fonts available internally on PostScript printers.

Text-Handling Commands

Table 3.8 list the commands that produce or modify text drawn by **igi**. The first set draw text and the second set modify its appearance. Table 3.9 lists the allowed values of the arguments to the JUSTIFY and PUTLABEL commands and the resulting justification of the text. Figure 3.17 shows examples of various text attributes such as the justification and the text escapes.

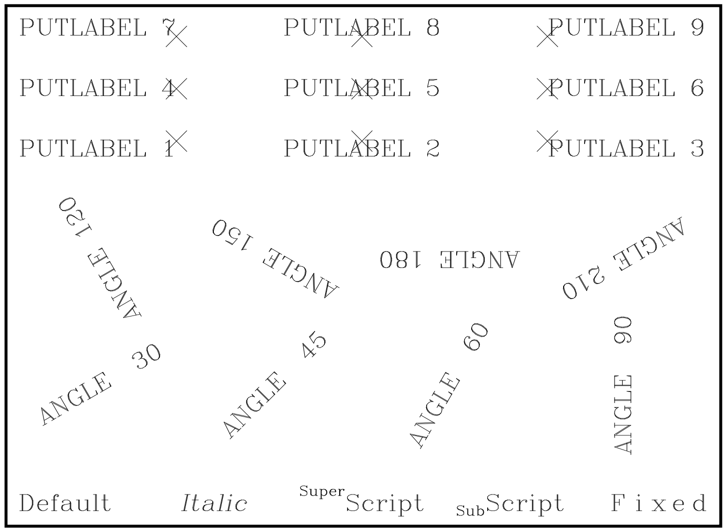
Table 3.8: Text-Handling Commands

Command	**Operation**
	Text Drawing Commands
`LABEL`	Draw text with current justification
`PUTLABEL`	Draw text with specified justification
`XLABEL`	Specify X axis label
`YLABEL`	Specify Y axis label
`TITLE`	Specify plot title
	Text Appearance Commands
`ANGLE`	Orientation
`EXPAND`	Size
`FONTSET`	Select font set
`PSFONT`	Specify PostScript font used in label(s)
`JUSTIFY`	Justification
`LWEIGHT`	Boldness (line width)

Table 3.9: Position Arguments to JUSTIFY and PUTLABEL

	Horizontal		
Vertical	**Left**	**Center**	**Right**
Above	7	8	9
Center	4	5	6
Below	1	2	3

Figure 3.17: Text Drawing Commands and Attributes



Fonts and Escape Sequences

Most attributes of plotted text are changed using *escape sequences* within the text itself. The backslash character (\) is used as the escape for single character sequences. A single \ changes the next character *only.* A double escape (\\) changes the attribute for the remainder of the string, or until it's changed again. Pairs of italics or proportional spacing escapes thus effectively toggle. Table 3.10 describes some of the text escape sequences.

Figure 3.18 shows all of the characters available in any text using embedded escapes with **igi** software fonts. Figure 3.19 shows characters available in the IRAF gio fonts. Roman is the default font and the drawn character matches the typed characters. In other fonts, the drawn character may be different from that typed. The small character in the upper right corner of each character cell shows the letter to type to have the symbol appear. Roman, Greek, and Script are proportional-spaced fonts. The *type* font is a monospaced (fixed-width) and is drawn one third the size of others. That is, to match text drawn with `EXPAND 1` using `\t` would require `EXPAND 3`.

Table 3.10: Text Escape Codes

igi Escape Code	gio Escape Code	Result
`\r`	`\fR`	Roman font
`\g`	`\fG`	Greek font
`\s`		Script font
`\t`		Small, monospaced, sans-serif font
`\i`	`\fI`	Toggle italics
	`\fB`	Bold
	`\fP`	Use PostScript font from PSFONT
`\f`		Toggle proportional spacing
`\u`		Superscript
`\d`		Subscript
`\b`		Backspace
`\e`		End string

Figure 3.18: igi Software Fonts and Embedded Text Escapes

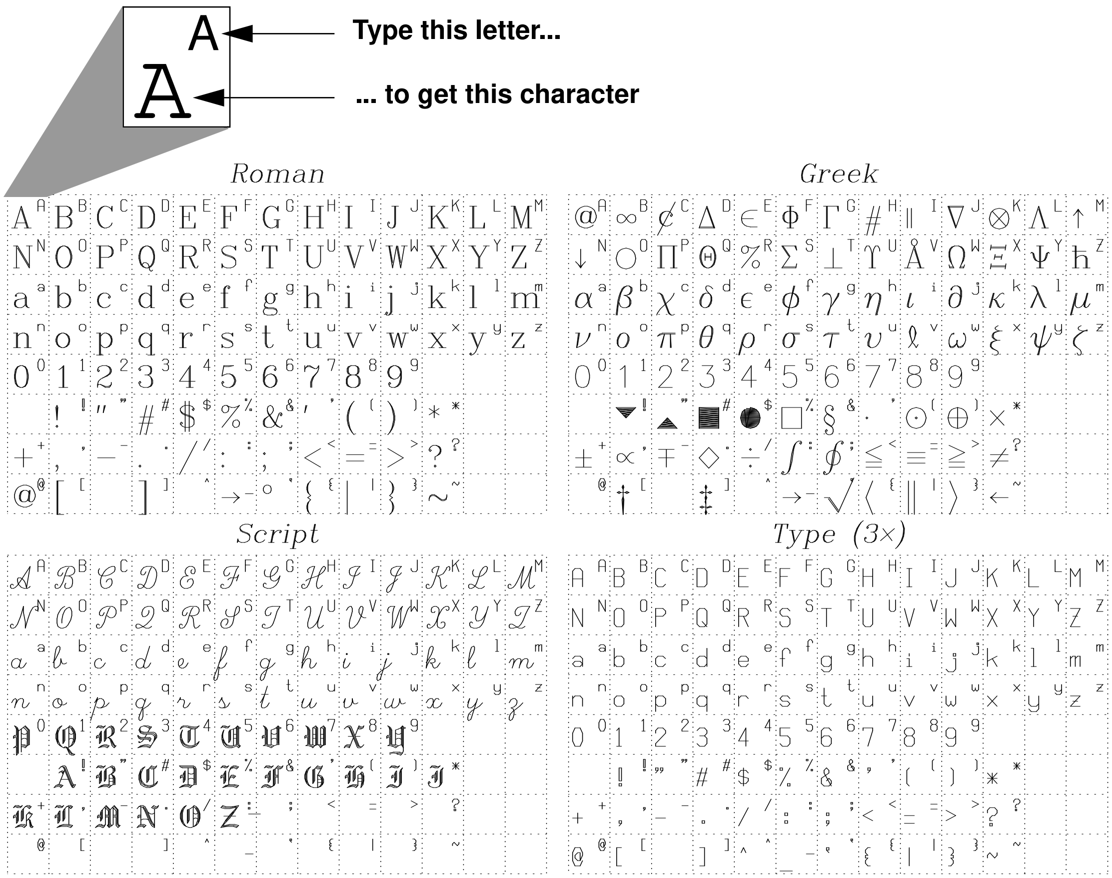
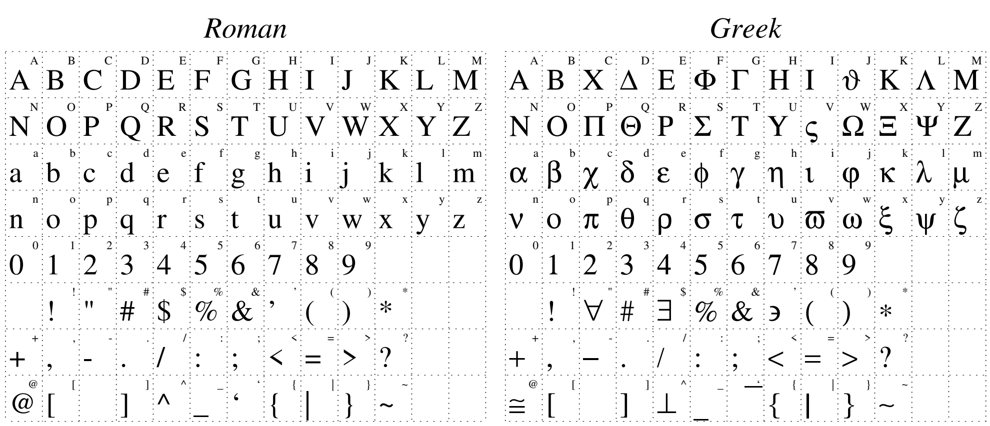


Figure 3.19: Hardware (gio) Fonts



PostScript Font Usage

The PostScript kernel allows **igi** to take advantage of whatever fonts are available on the output printer. The `PSFONT` command is used to specify a font known to the printer (which may be different from those known internally to **igi**) for use in text label within a plot. When `FONTSET` is set to `hard`, the `PSFONT` command specifies the new font that will be used for those characters prefixed with the escape code `\fP` in any subsequent labels. The `PSFONT` command can be called any number of times within a script allowing great variety of font usage within a plot. The PostScript font escape code remains active for all characters following it in the label until reset using `\fR`. As with any other text, all the usual formatting commands, such as `EXPAND`, `ANGLE` and `JUSTIFY`, will work as expected on the special PostScript characters. Figure 3.20 is an example of using **igi** commands to work with PostScript fonts.

Figure 3.20: Using PostScript Fonts

igi Command Script

```
FONTSET hard
PSFONT ZapfDingbats
LIMITS 0 1 0 1
RELOCATE 0.2 0.5
PUTLABEL 6 This is a \fPSTAR\fR
```

PSFONT Loads Font

Font is Switched Using Escape Sequence

PostScript Output

This is a ❋✳✡✬

Each printer behaves differently if it does not recognize a font that was specified, with some using a default font in place of the unrecognized font and others not creating any output at all. It is up to the user, in this case, to be sure that those fonts specified in the **igi** script are recognized by the printer they are using.

Image Display

Some graphics devices are able to display greyscale images, and **igi** has basic image rendering (display) capabilities to take advantage of this capability. This overlaps partly with the capabilities for dealing with line (vector) plots.

The usual sequence of events in image display with the appropriate commands is:

1. Read the data (`ZSECTION`).
2. Map the display brightness (`ZRANGE`).
3. If desired, apply a custom look-up table (`SAOCMAP`).
4. Scale the viewport (`LIMITS`).
5. Display the image (`PIXMAP`).
6. Supply any world coordinates as labels, if desired (`WCSLAB`).

Table 3.11 summarizes the commands used for image display, and an example is given in Figure 1.9.

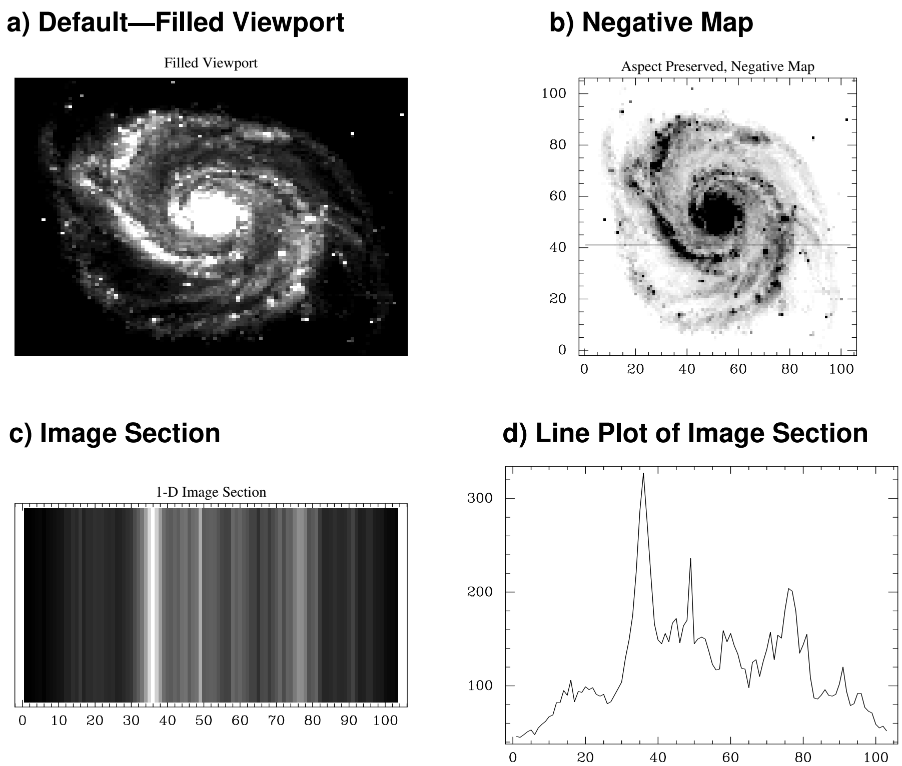
By default, the displayed image will fill the viewport as currently defined. This may be specified as for other plotting with the `VPAGE`, `LOCATION` and other scaling commands. This will, in general distort the image since the aspect ratio of the viewport and image may not match. To force the viewport to match the image, use the `FITPIX` command. This is analogous to `LOCATION` to set the edges of the viewport (plot axes) on the virtual page, but it adjusts the specified viewport to match the aspect ratio of the image, based on the number of pixels in each dimension, on the current graphics device.

Table 3.11: Image Display Commands

Command	Result
ZSECTION	Read 2-D greyscale data into Z buffer
ZRANGE	Scale greyscale to display brightness
SAOCMAP	Specify an SAOimage colormap to use
FITPIX	Set viewport to match aspect ratio
LIMITS	Scale the WCS based on image dimensions
MARGIN	Include a border between axis and image
PIXMAP	Render the image on the display
WCSLAB	Labels axes based on World Coordinates found in image header

The four panels making up Figure 3.21 show some of **igi**'s image display capabilities. Panel **a** shows **igi**'s default action when asked to display an image—to fill the viewport with a positive representation of the data. Panel **b** uses the `FITPIX` command to preserve the aspect ratio of the image, and uses `ZRANGE` to map brighter pixels to darker values (and vice versa)—resulting in a negative gray map. Panel **c** shows the result of an image section consisting of a single line of the same image displayed in panels **a** and **b**. The displayed image is widened to fill the viewport. Panel **d** is a line plot of the same image section. The horizontal line plotted in **b** is the same line represented in **c** and **d**.

Figure 3.21: Displaying Images



Custom Color Map Usage

The SAOCMAP command lets you read in a custom color map table produced by SAOimage (or written in the same format). This will produce a color image on a color printer; on a non-color printer, it will use a non-linear greyscale transformation. This command must be given anytime between the ZSECTION and the PIXMAP commands in the script.

To create a custom color map using SAOimage, the image is first displayed in SAOimage using IRAF display tasks. An appropriate color map is chosen and adjusted to produce the desired view. The color map is then written out for use in the SAOCMAP command. (The file format is described on page 110, in case the SAOimage task is unavailable). By setting ZRANGE identical to the values used when displaying the image in SAOimage and using the same color map, **igi** will reproduce exactly the same output as viewed on the screen, including color output of the image if a color transformation was specified in the color map file.

Image Manipulation with ZSECTION

The `ZSECTION` command provides two optional parameters:

- On-the-fly rotation by ± 90˚.
- On-the-fly block averaging.

Although limited in scope, these parameters can significantly speed up the processing of the image by reducing the image size and manipulating it directly in memory. The syntax for using the optional parameters is:

```
ZSECTION file.hhh [pixel rotation]
```

where *`pixel`* specifies the number of pixels to average or sum in both dimensions and *`rotation`* is either 90 for a 90˚ counter-clockwise rotation or –90 for a 90˚ clockwise rotation. If a negative integer is given for the pixel value, it will perform a block-sum instead of an average. The block-averaging function can be important when dealing with images that have more pixels than a 1024 x 1024 image. Even a simple 2 x 2 block average will reduce the size of the output image by a factor of 4 and thereby speed up the processing and printing time without significantly reducing the quality of the display.

Image Sections

Assuming the image named `specimw` contains one line having wavelength values and `specimf` has other intensity values, the following commands would plot the spectrum as intensities versus wavelength:

Figure 3.22: Reading and Plotting Image Sections

```
igi> xsection specimw.hhh[*,1]
igi> ysection specimf.hhh[*,2]
igi> limits
igi> box
igi> connect
```

Windows

The flexibility in specifying sections of the page for plotting using `VPAGE` or `LOCATION` not only applies to creating one plot, but also allows the user to place more than one plot on the page at a time. There are a couple of ways to specify how the output page should be divided for multiple plots: use the `WINDOW` command and specify a separate, explicit `LOCATION` for each plot.

WINDOW Command

The `WINDOW` command splits the page into regions by specifying how many plots should go across the page and how many should go up the page. These plots are then referred to in later calls by a logical number. In Figure 3.23, two curves are plotted in separate windows (different axis grids): raw data in one window, and the log of the data in another. In this example, two windows were specified with them going side-by-side on the page. The first `WINDOW` command in Figure 3.23 sets all this up with the following parameters:

- The first value, 2, specifies how many windows will be created in the X direction.
- The second value, 1, specified the number of windows in the Y direction.
- The final value, 1, sets which window will be used for the first plot.

The second `WINDOW` command simply defines what window will be used for the next plot. This command makes it easy to set up a regular grid of plots without complex calculations of page sections.

Figure 3.23: Plots in Windows

```
igi> window 2 1 1
igi> limits; box; connect
igi> window 2
igi> yevaluate log10(y)
igi> limits; box; connect
```

Stacking Curves

The **igi** script below shows another way to stack multiple curves, each with its own set of axes. The result is in Figure 3.24. It’s a bit tricky to get the axes to abut *and* draw tick labels on the bottom X axis only. In the method shown here, you would need to compute the coordinates of each viewport (axes), which makes it awkward to change the number of curves. `WINDOW` is more automatic, but you must choose between being able to label the X axis or abutting the axes. The difference is between defining multiple viewports within a single virtual page or a single viewport in multiple virtual pages.

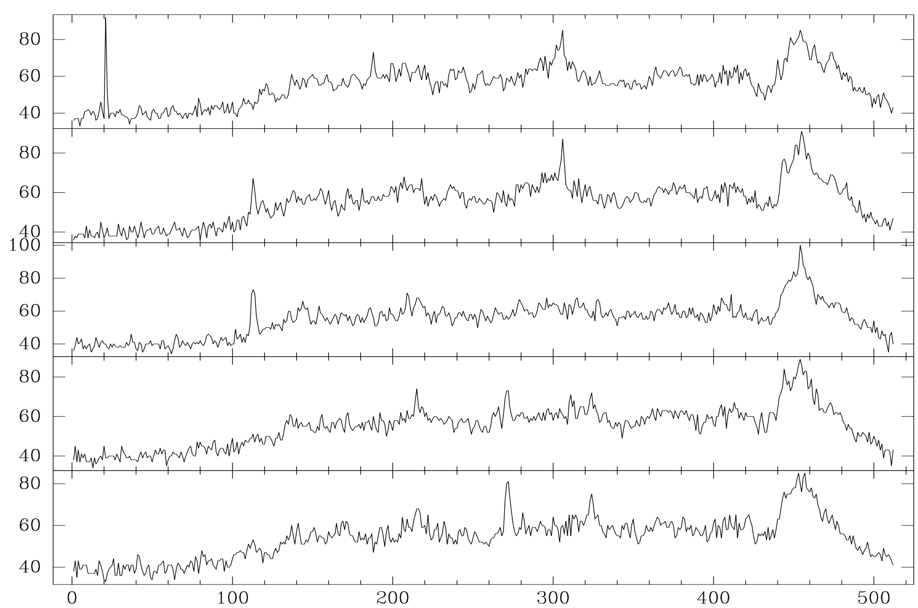
Figure 3.24: Multiple Curves Plotted in Separate Frames

Script to Stack Plots

```
DEFINE STACK                          Macro to draw axes and curve
  LOCATION .075 .98 &1 &2             Specify the window
  LIMITS;  MARGIN                     Scale the plot
  BOX &3 2                            Draw the axes
  CONNECT                             Draw the curve
  TICKSIZE 20 100 20 20               Specify tick spacing
END

YSECTION "dev$pix[*,1]"
STACK .05 .23 1
YSECTION "dev$pix[*,3]"
STACK .23 .41 0
YSECTION "dev$pix[*,5]"
STACK .41 .59 0
YSECTION "dev$pix[*,7]"
STACK .59 .77 0
YSECTION "dev$pix[*,9]"
STACK .77 .95 0
```

Resulting Plot



Interactive Graphics Cursor

All capabilities of the IRAF gio[3] graphics cursor are available from **igi** for interacting with a plot. These features include zooming, panning and

3. Graphics I/O, the graphics facility common to IRAF tasks.

scrolling of the plot screen, printing the plot, writing the graphics to a file or reading in stored graphics metacode and so forth. Cursor interaction may be initiated with the **igi** command `CURSES`.

After a plot has been produced, the graphics metacode commands remain in the cl graphics buffer[4] after **igi** terminates, until it is explicitly erased with the cl task `gflush`, or graphics are opened again. Cursor interaction is still available from the cl to interact with the plot using `=gcur`, in order to obtain hard copies using the `:.snap` cursor command, for example.

In addition to the IRAF gio cursor capability, all **igi** commands are available in cursor mode using the colon command capability. Upon typing **:** when the graphics cursor is displayed, the terminal returns to text mode with the **:** displayed as a prompt. At this prompt, any valid **igi** command may be typed. All of the logical pen movement commands (`RELOCATE`, `DRAW`, etc.) will use the current cursor position regardless of the coordinate system or any arguments input on the command, therefore, these coordinate arguments should be omitted.

See the IRAF help for cursors by typing `help cursors` in the cl. The specific operation of graphics mode depends on the graphics device being used.

4. The cl controls interactive graphics and uses a buffer to store plots. This buffer can be manipulated to some extent. Type `help gio` for more information.

CHAPTER 4:
Command Reference

This chapter fully documents every individual **igi** command including any arguments default values and so forth. Common concepts and an overview of the capabilities are described in previous chapters. Some brief examples are presented here. More extensive examples exist in other sections. In particular, the appendix on page 119 provides the **igi** commands used to draw the various figures demonstrating concepts in this document.

Table 4.1 lists each command alphabetically, along with the page on which the full description appears, a brief description and category, whether it is saved in the command buffer, if it may be undone with `UNDO`, and a page reference for the full description.

Table 4.1: Summary of igi Commands.

Command	Page	Class[a]	Saved	Undo	Description
`!`	76	E[b]			Escape a command to the cl
`?`	77	E			Page the help summary menu
`^`	77	E			Re-execute a previous command
`ANGLE`	77	A[c]	•		Set the marker and text rotation angle
`APROPOS`	78	E			List commands associated with a keyword
`AXIS`	78	P[d]	•	•	Draw and label an arbitrary axis
`BARGRAPH`	79	P	•	•	Draw a bar graph
`BOX`	80	P	•	•	Draw and label the axes
`BYE`	81	E			Terminate igi (alias for *END*)
`COLOR`	81	P	•		Set the drawing color
`CONNECT`	84	P	•	•	Draw a (polyline) curve connecting the data coordinates
`CURSES`	84	E			Start interactive cursor mode
`DATA`	85	D[e]	•		Specify the input data file

Table 4.1: Summary of igi Commands. (Continued)

Command	Page	Class[a]	Saved	Undo	Description
DDRAW	86	P	•	•	Pen down move (draw) in NDC
DEFINE	75	E			Enter macro text
DLIST	86	D			Print the data values
DMOVE	103	P	•		Pen up move in NDC (alias for *DRELOCATE*)
DOT	86	P	•	•	Draw a single marker
DRAW	86	P	•	•	Pen down move (draw) in WC
DRELOCATE	109	P	•		Pen up move in NDC
ECOLUMN	82	D	•		Read errors from a column of the data file
EDITCMD	87	E			Edit the command buffer or macro text
EEVALUATE	91	D	•		Operate on error values
ELLIPSE	87	P	•	•	Draw an ellipse
END	88	E			Exit igi or terminate macro define (end the current mode)
ERASE	88	P	•		Erase the screen
ERRORBAR	89	P	•	•	Draw error bars
ESECTION	111	D	•		Read an image section into the error data buffer
ETYPE	90	A	•		Change the error bar style
EXPAND	92	A	•		Set the marker and text size
FILLPAT	92	A			Select a filled-area pattern
FITPIX	93	S			Set a viewport to match an image’s aspect
FMTICK	94	A	•		Specify the axis tick label format
FONTSET	95	A			Select igi or gio fonts
GRID	96	P	•	•	Draw lines between major axis ticks
HELP	96	E			Print help
HISTOGRAM	96	P	•	•	Draw a histogram plot (bar graph)
ID	97	P	•	•	Write identification label
IMGWCS	97	S[f]	•		Toggle using image WCS for X values
INPUT	97	E	•		Execute commands from a file
JUSTIFY	97	A	•		Set text justification
LABEL	98	P	•	•	Draw text at current position

Table 4.1: Summary of igi Commands. (Continued)

Command	Page	Class[a]	Saved	Undo	Description
LCOLUMN	82	D	•		Read limits data from a column of the data file
LEVALUATE	91	D	•		Operate on limit arrays
LIMITS	99	S	•		Set the plot scale
LINES	99	D	•		Specify the range of input rows to read
LIST	100	E			List the command buffer or macro text
LOCATION	100	S	•		Set the viewport on the virtual page
LSECTION	111	D	•		Read an image section into the limits data buffer
LTYPE	101	A	•		Set the line style (hardware dependent)
LWEIGHT	102	A	•		Set the line width (hardware dependent)
MACROS	102	E	•		List defined macros
MARGIN	102	S	•		Rescale to include a border between curve and axes
MENU	103	E			Page the help summary
MINMAX	103	P			Print the data range
MOVE	103	P	•		Pen up move (alias for *RELOCATE*)
NOTATION	104	A	•		Set the format for axis labels
NUMBER	105	P			Draw numerals representing index of each coordinate
PAGECMD	105	E			Page the command buffer or macro text
PCOLUMN	82	D	•		Read point specifications from the input data file
PDRAW	86	P	•	•	Pen down move (draw) in PC (Physical coordinates)
PEVALUATE	91	D	•		Operate on point code values
PHYSICAL	105	S	•		Set the virtual page in physical device coordinates
PIXMAP	106	P	•		Draw (render) an image (pixmap)
PLAYBACK	106	E			Execute the commands in the command buffer
PMOVE	103	P	•		Pen up move in PC (alias for *DRELOCATE*)
POINTS	106	P	•	•	Draw a polymarker at input data coordinates
POLYGON	106	P	•	•	Draw a filled polygon using the input data coordinates
PRELOCATE	109	P	•		Pen up move in PC
PSECTION	111	D	•		Read an image section into the point-type data buffer
PSFONT	107	A			Select PostScript font to use for special characters

Table 4.1: Summary of igi Commands. (Continued)

Command	Page	Class[a]	Saved	Undo	Description
PTYPE	107	A	•		Specify the point marker type
PUTLABEL	108	P	•	•	Draw a label with a specified justification
READ	108	D	•		Read commands into the command buffer without execution
RELOCATE	109	P	•		Pen up move in WC
RESET	110	A	•		Return settable plot and data parameters to defaults
SAOCMAP	110	D			Specify a gray or color map for display
SCOLUMN	82	D	•		Read scratch data from a column of the data file
SEVALUATE	91	D	•		Operate on scratch array values
SHOW	113	E			Page the plot limits and attributes
SSECTION	111	D	•		Read an image section into the scratch data buffer
STEP	113	P	•	•	Draw a stepped curve (histogram without vertical lines)
TICKSIZE	114	A	•		Set the axis tick spacing
TITLE	114	P	•	•	Specify the plot title
ULLIM	114	P	•	•	Draw upper or lower limit symbols
UNDO	115	E			Selectively erase the last plot command
VDRAW	86	P	•	•	Pen down move (draw) in VPC (virtual page coordinates)
VERSION	115	E	•	•	Show **igi** version and date
VMOVE	103	P	•		Pen up move in VPC (alias for *VRELOCATE*)
VPAGE	115	S	•		Set the virtual page in normalized coordinates
VRELOCATE	109	P	•		Pen up move in VPC
WCSLAB	115	P	•	•	Label the viewport in WCS coordinates
WINDOW	116	S	•		Divide screen into panes
WRITE	117	E			Write the command buffer to a file
XCOLUMN	82	D	•		Read X coordinates from a column of the data file
XEVALUATE	91	D	•		Operate on X data values
XFLIP	94	S	•		Flip X axis
XLABEL	98	P	•	•	Specify the X axis label
XLOGARITHM	101	D	•		Take common log of X data

Table 4.1: Summary of igi Commands. (Continued)

Command	Page	Class[a]	Saved	Undo	Description
XSECTION	111	D	•		Read X coordinates from an image section
XSIXTY	113	P	•	•	Specify the use of sexagesimal numbering for the X axis
YCOLUMN	82	D	•		Read Y coordinates from a column of the data file
YEVALUATE	91	D	•		Operate on Y data values
YFLIP	94	S	•		Flip Y axis
YLABEL	98	P	•	•	Specify the Y axis label
YLOGARITHM	101	D	•		Take common log of Y data
YSECTION	111	D	•		Read Y coordinates from an image section
YSIXTY	113	P	•	•	Specify the use of sexagesimal numbering for the Y axis
ZEVALUATE	91	D	•		Operate on image pixel values
ZRANGE	117	S			Map image values display brightness
ZSECTION	111	D			Read Z values from an image section

a. See “Command Classes” on page 29.
b. Execution: modify action of igi
c. Attribute: modify characteristics of plotting elements
d. Plotting: draw a graphic element on the plot, including text
e. Data: input or modify data in igi buffers
f. Scaling: modify plot scale

Plotting

Saved

Undo

Commands and command families are listed alphabetically here. For a family of commands (see “Command Families” on page 30), a description of every individual command in the family is provided in that one section, with sub-sections describing the unique characteristic of each individual command where necessary. The heading for each command lists any arguments in the order in which they must appear. A template command is listed for each possible set of arguments for every command. Symbols (such as those to the left) appear with each command description indicating the category of the command, if it is saved in the command buffer, and if a plotted result may be erased, provided the device supports it.

!

Execution

!command

This is not strictly a command and therefore does not really have arguments. The `!` is a prefix to text that indicates that what follows should be passed to the IRAF cl and interpreted there. Input text following the `!` up to the end of the line is passed to the cl for interpretation. The cl environment at the time **igi** was invoked is in effect. For example: `!dir` executes the IRAF `directory` command to list the files in the current directory using the IRAF virtual file names. Note that cl commands are *case sensitive*.

The command `!cl` starts a new cl process complete with the cl prompt and with the same environment as when **igi** started. To terminate this process and return to **igi**, type `bye` at the cl prompt.

Table 4.1: Escaping Host Commands

Host OS	Command	Termination
cl	`!cl`	`bye`
Unix	`!!csh`	`exit`
VMS	`!!!`	`logout`

Any text preceded by `!!` is executed as a host (e.g., VMS, Unix, etc.) command. The first `!` escapes the command to the cl, and since the remaining text starts with another `!`, this escapes the command to the host. Any valid host command will be interpreted. The environment at the time the cl and **igi** were invoked is in effect. For example, the command `!!DIR` on VMS (`!!ls` on Unix) will list the current default directory using the host file names. The command `!!!` on VMS or `!!csh` on Unix starts a new host process. To terminate this process and return to **igi**, type `logout` on VMS and `exit` on Unix. Table 4.1 summarizes the commands to enter and exit host processes.

?

Execution

? [command]

Page the help summary (menu). Only a list of the commands and their meaning is listed. To get the extended text, use the `HELP` command. See `APROPOS` and `MENU`.

^

Execution

^
^ n

Execute a previous command. If no argument follows the `^` character, execute the last command. The optional argument must be an integer, and if present, specifies the sequence number of the command in the command buffer to execute. See `LIST`, `EDITCMD`, and `PAGECMD`. Note that not all commands are saved for reuse. See the description of each command or the summary table to determine if a command is saved.

ANGLE

Attribute **Saved**

ANGLE
ANGLE angle

The orientation for markers and text is specified by the floating point parameter in *degrees*, counterclockwise from the horizontal (+X axis). If no argument is specified, the value of the currently set angle will be listed. The commands `LABEL`, `PUTLABEL`, `AXIS`, `POINTS`, `DOT`, and ellipse use the current angle. Figure 3.17 on page 60 shows how this affects text strings.

APROPOS

Execution

APROPOS keyword

List a brief description of the commands associated with the specified keyword. The required argument keyword is a string that may be searched. Figure 4.1 shows the result of the command: `APROPOS move`.

See Table 4.1 for the command summary and `HELP` for the full help text.

Figure 4.1: Sample `APROPOS` Output

```
DDRAW         Pen down move (draw) in NDC
DMOVE         Pen up move in NDC (alias for DRELOCATE)
DRAW          Pen down move (draw) in WC
DRELOCATE     Pen up move in NDC
MOVE          Pen up move (alias for RELOCATE)
RELOCATE      Pen up move in WC
VDRAW         Pen down move (draw) in VPC
VMOVE         Pen up move in VPC (alias for VRELOCATE)
VRELOCATE     Pen up move in VPC
```

AXIS

Plotting **Saved** **Undo**

AXIS w1 w2 minor major x y len label clock

Draw an arbitrary axis with tick marks and tick labels. The labels are scaled from `w1` to `w2` in data coordinates (WCS) starting at (`x`,`y`) and length `len` in virtual page coordinates (VPC). If `major` > 0, use that for the spacing of major ticks. If `minor` < 0, draw a logarithmic axis; if `minor` > 0, try to use that for the spacing of minor ticks; if `minor = 0`, let **igi** decide where to put minor ticks. If `label = 0`, don't draw labels; if `label = 1`, draw labels parallel to the axis; if `label = 2`, draw labels perpendicular to the axis. If `clock = 1` draw ticks clockwise with respect to the axis, counterclockwise if `clock = 0`. Table Table 4.1 summarizes the arguments. Tables Table 4.5 on page 80 and Table 4.3 on page 79 list possible argument values for the `clock` and `label` arguments. The current angle specifies the position angle of the axis with respect to horizontal (see "!" on page 76). The current tick label format specifies how the tick labels will be written (see `FMTICK` on page 94). The current line width specifies how the axis will be drawn (see `LWEIGHT` on page 102). See Figure 3.12 for examples of using `AXIS`. See `BOX`, below, for drawing default axes at the current plot scale.

Table 4.1: Label Arguments

Argument	Type	Description
`w1`	real	WC at beginning of axis
`w2`	real	WC at end of axis
`minor`	real	Spacing of minor ticks; < 0 ⇒ *logarithmic*
`major`	real	Spacing of major ticks in WC
`x, y`	real	Position of axis (at w1) in VPC
`len`	real	Size of axis in VPC
`label`	integer	Orientation of tick labels
`clock`	integer	Location of tick labels relative to axis

Table 4.2: Labeling Parameters in the `AXIS` Command

Minor	Label	Clock
`< 0`	Logarithmic	
`-2`	Tick labels with zeros ⊥ axis	
`-1`	Tick labels with zeros ‖ axis	
`0`	No label	Counterclockwise
`1`	Tick labels ‖ axis	Clockwise
`2`	Tick labels ⊥ axis	

Table 4.3: Orientation Arguments for Labels

Value	Interpretation
0	Ticks and labels counterclockwise
1	Ticks and labels clockwise

BARGRAPH

Plotting **Saved** **Undo**

Draw a bar graph (sometimes confusingly referred to as a histogram style plot) through the current *X* and *Y* data. If no X data exist, the horizontal axis will be scaled on the pixel numbers. See `LTYPE` and `LWEIGHT` for specifying the style of the curve to draw. See `CONNECT`, `STEP`, and `POINTS` for different curve styles

BOX

Plotting | Saved | Undo

BOX
BOX xlabel
BOX xlabel ylabel

Draw and label all four axes using the current page, viewport, and window transformations and any axis label or title specified. The optional label arguments specify the X and Y tick labels as shown in Table 4.5. Positive values of `xlabel` and `ylabel` result in non-significant leading and trailing zeros to be omitted from tick labels. Negative values force leading and trailing zeros in certain circumstances, such as a leading zero left of a decimal point for a value less than 1 or a single zero following a decimal. See Figure 3.15 on page 57 for an example.

Table 4.4: Label Commands

Argument	Default	Meaning
`xlabel`	1	Tick labels parallel to axis
`ylabel`	2	Tick labels perpendicular to axis

See `AXIS` for drawing a single arbitrary axis, `XYLABEL` for annotating the axes with a text label, `LIMITS` for setting the plot scale, and `MARGIN` for adjusting the scale to include a border between curves and axes.

Table 4.5: Label Arguments

Value	Interpretation
-2	Tick labels perpendicular to axis, zeros shown
-1	Tick labels parallel to axis, zeros shown
0	No tick labels
1	Tick labels parallel to axis, zeros suppressed
2	Tick labels perpendicular to axis, zeros suppressed

The current tick label format specifies how the tick labels will be written (see `FMTICK`). The current line width specifies how the axis will be drawn (see `LWEIGHT`).

BYE

Execution

Terminate **igi**. This is an alias for `END`, except in macro define mode (see `DEFINE`).

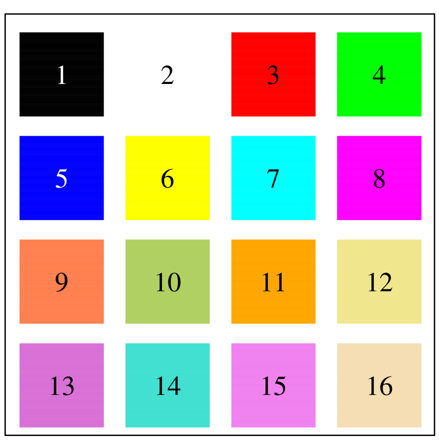
COLOR

Attribute **Saved**

COLOR index

Set the color index for all subsequent drawing. The argument, the color index, is a positive integer. All drawing operations—lines, text, symbols, curves, axes—will be rendered in the selected color. The default color index is 1, rendered as black (or the foreground color) on most devices. The rendered color depends on the device used and whether the IRAF kernel supports it. (Currently, only the psikern IRAF kernel supports color for use with color PostScript devices.) Note that colors may be rendered on a monochrome device as shades of gray or dithered with some halftone pattern.

Figure 4.1: Default Colors Available Using COLOR



Index	Color
`1`	Black
`2`	White
`3`	Red
`4`	Green
`5`	Blue
`6`	Yellow
`7`	Cyan
`8`	Magenta
`9`	Coral
`10`	Maroon
`11`	Orange
`12`	Khaki
`13`	Orchid
`14`	Turquoise
`15`	Violet
`16`	Wheat

COLUMN

Data | Saved

XCOLUMN column
YCOLUMN column
ECOLUMN column
PCOLUMN column
LCOLUMN column
SCOLUMN column

Read a column of data from the file specified by the `DATA` command. The command prefix indicates which data buffer to fill, as shown in Table 4.6

Table 4.6: Commands, Data Sources, and Buffers

Command	Data to be Read	Buffer to be Filled
`XCOLUMN`	X coordinates	X buffer
`YCOLUMN`	Y coordinates	Y buffer
`ECOLUMN`	error	E buffer
`PCOLUMN`	point marker styles	P buffer
`LCOLUMN`	limits	L buffer
`SCOLUMN`	scratch	S buffer

If the input data is a text (ASCII) file, column must be an integer specifying the column number. If it is an STSDAS table, column must be a string specifying the column name. Note that case in binary table column names is significant in the sense that the case is maintained in the table. However, the software that reads the tables ignores case, treating upper and lower case column names the same. Column names are not minimum-matched, however, so they may not be abbreviated. See `DATA` to specify the input file (table) name. See `[XYEPLS]SECTION` to read an IRAF image into a buffer.

ECOLUMN

Read error values from the column in the current data file specified by the column parameter. See `ETYPE` to specify the style of error bars and `ERRORBAR` for drawing them.

LCOLUMN

Read a column of limits data from the file specified by the data command. See `ULLIM` for use of the limits data in drawing upper or lower limit symbols.

PCOLUMN

Read point specifications from the input data file (table). The floating point data values are assumed to represent a coded marker style and size for each data value. The units digit specifies the marker style by the same code as `PTYPE`. See Figure 4.11 on page 107 for the marker type codes. Any value larger than 4 is read as 4 (half-filled symbols).

The tens and up digits specify the number of vertices in the marker, and the decimal portion represents a multiplier for the global text and marker size. The global text and marker angle determines the angle of all point markers. For example, if the value in the points column were 52.5 the marker would be a five pointed star half the size of the current marker and text size. If no decimal value is present, the marker size is the global text and marker size.

SCOLUMN

Read a column of scratch data from the file specified by the data command. See `EVALUATE` for use of the scratch data in manipulating the data buffers.

XCOLUMN

Read X coordinate data from the specified column in the current input data file. If no X data are specified, the Y data are plotted against row number.

YCOLUMN

Read Y coordinate data from the specified column in the current input data file. If no X data are specified, the Y data are plotted against row number.

CONNECT

Plotting | Saved | Undo

CONNECT

Draw a (polyline) curve in the current line style (see `LTYPE`) connecting the data coordinates. See `POINTS` for drawing markers without connection.

CURSES

Execution

CURSES
CURSES file

Start cursor mode and read back the cursor position. This command invokes the IRAF gio cursor facility (see `!help cursor`). If the argument file exists, it is the name of a file to which **igi** writes the coordinates of positions selected with the cursor. If no argument exists, the positions are written only to `STDOUT` (the terminal); the coordinates are not saved in a file. To terminate cursor mode, type `q` or `e` (in *lower case*) or `CTRL-D`. Any other lower case character will list the cursor position and redisplay the cursor. Most upper case and special characters are recognized by IRAF cursor mode.

All **igi** commands are available in cursor mode using the "colon command" capability. Upon typing **:** when the graphics cursor is displayed, the terminal returns to text mode with the **:** displayed as a prompt. At this prompt, any valid **igi** command may be typed. Commands expecting input coordinates (`DVRELOCATE` and `DVDRAW`) will use the current cursor position regardless of any arguments input on the command, therefore, these coordinates may be omitted.

The `CURSES` command is not stored in the command buffer, but a `RELOCATE` command with the last cursor position is stored on each cursor read or **:** command. Therefore, on `PLAYBACK`, no cursor interaction takes place but any commands relying on the interactively specified cursor position execute appropriately.

Type `!help cursor` from **igi** (`help cursor` from the cl) for help on the IRAF cursor capabilities. Note that the specific operation of interacting with the graphics cursor depends on the terminal used.

DATA

Data **Saved**

DATA filename

Specify the input data file. This file may be either an ASCII list (text) file whose contents are numerical values arranged in columns or an STSDAS binary table with named columns. If no file name is specified, the current data file and type is listed on the standard output. The filename can also contain row selector in order to access FITS tables which contain arrays as elements of a column. Information about the row selector syntax is available by typing `help selector` within IRAF/STSDAS.

See `[XYEPLS]COLUMN` to read an arbitrary column into one of the predefined **igi** arrays. See `[XYEPLS]SECTION` to read an IRAF image into a buffer.

DEFINE

Execution

DEFINE macro

Enter macro text. The argument is the name of the macro. A defined macro may be invoked as any other command, possibly with arguments. `DEFINE` causes **igi** to enter macro define mode, in which commands are not interpreted (except `END`) but stored as is. The prompt `macro>` indicates macro define mode. To terminate the macro definition, type `END`. The macro is invoked by typing its name. The name is not case sensitive. It may not be abbreviated. When the macro is invoked, the text is expanded and interpreted just as typed text.

Optional macro arguments are positional and are defined by the character `&` and an integer. Defined arguments are replaced by their values typed on the command line in the order specified by their definition. The order in which the arguments appear in the macro text is unimportant, but the order in which the argument values appear on the invoked macro must match the definitions. The highest numbered argument definition specifies how many argument values must appear in the macro call.

If a macro is defined more than once with the same name, the new text and argument definition will supercede existing text. Macro text may be listed (`LIST`), edited (`EDITCMD`), or saved in a text file (`WRITE`). The `MACROS` command lists the currently defined macro names and the number of their arguments.

DLIST

Execution

DLIST

Page the data values (if any) currently in the X, Y, error, and point code buffers on the text screen.

DOT

Plotting | Saved | Undo

DOT

Draw a single marker of the current style, size, and angle at the current location.

See `PTYPE`, `EXPAND`, and `ANGLE` to specify the marker style. To draw a single circle or ellipse, see the `ELLIPSE` command. To draw markers at the data coordinates in the plot buffers, use the `POINTS` command. See Figure 4.11 on page 107 for the marker type codes.

DRAW

Plotting | Saved | Undo

DRAW xpos ypos
DDRAW xpos ypos
PDRAW xpos ypos
VDRAW xpos ypos

Draw a line in the current line style from the current position to the coordinates specified by the two floating point parameters. The ending point of the line becomes the new current position. If any `DRAW` command is used in cursor mode (:`DRAW`, :`DDRAW`, :`PDRAW`, or :`VDRAW`), `xpos` and `ypos` are ignored and should be omitted.

Each `DRAW` command works within a different coordinate system and requires that coordinate system to be used for the input positions (`xpos` and `ypos`). The coordinate system used for each command are given in Table 4.7..

Table 4.7: Draw Commands and Coordinate Systems

Command	Coordinate System
`DRAW`	World Coordinates (WC)
`DDRAW`	Normalized device coordinates (NDC)
`PDRAW`	Physical coordinates (PC)
`VDRAW`	Virtual page coordinates (VPC)

See the `RELOCATE` (or `MOVE`) command families to specify the initial position in the desired coordinate system.

EDITCMD

Execution

EDITCMD macro

Edit the command buffer or macro text. An optional argument specifies the name of a defined macro. If no argument is specified, then edit the command buffer. `EDITCMD` uses the cl `edit` command and therefore uses the editor specified by the cl `editor` environment variable.

ELLIPSE

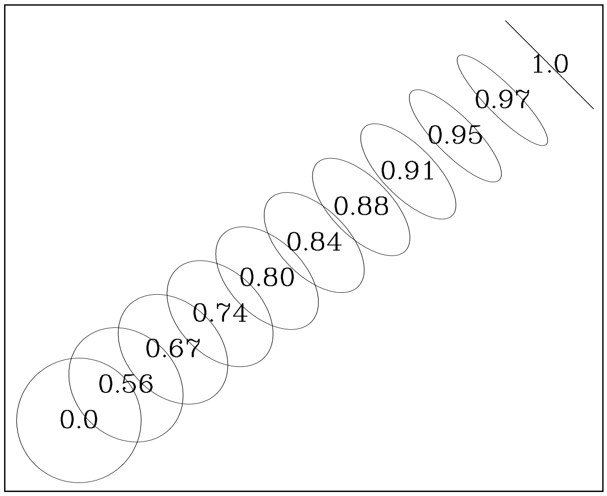
Plotting | Saved | Undo

ELLIPSE eccentricity

Draw an ellipse with specified eccentricity. The major axis is the current size multiplied by the default point marker size. The center of the ellipse is at the current location, and it is rotated counterclockwise from the horizontal by the current angle. Note that a circle is an ellipse with zero eccentricity.

Figure 4.2 shows examples of `ELLIPSE`.

Figure 4.2: Examples of ELLIPSE



END

Execution

End the current command mode. In normal interactive command mode, `END` terminates **igi**. `END` also terminates macro define mode to resume interpreting commands.

ERASE

Plotting

Erase the entire screen. All of the parameters retain their values. To reset data and plot attribute parameters to defaults, use `RESET`. Note that it is not possible to erase only part of the screen, except to `UNDO` a previous command, if the hardware supports the erase line type.

ERRORBAR

Plotting | Saved | Undo

ERRORBAR direction

Draw error bars at the positions specified by current X and Y coordinate buffers. The integer argument direction specifies the direction of the error bars:

Table 4.8: Error Bar Position Arguments

Code	Axis	Direction
-2	-Y	Down
-1	-X	Left
1	+X	Right
2	+Y	Up
3	-X	Left
4	-Y	Down

ERRORBAR must be used twice to draw a full error bar spanning the data coordinate. For symmetrical errors, use the same data column and change the direction argument. For asymmetrical errors, read new data into the errors buffer and respecify the direction argument.

The current text and point size determines the size of the ticks at the end of each error bar; use EXPAND to change the size. Change the default error bar style with ETYPE.

See DATA and ECOLUMN to read error data and EEVALUATE to operate on them.

ETYPE

Attribute | Saved

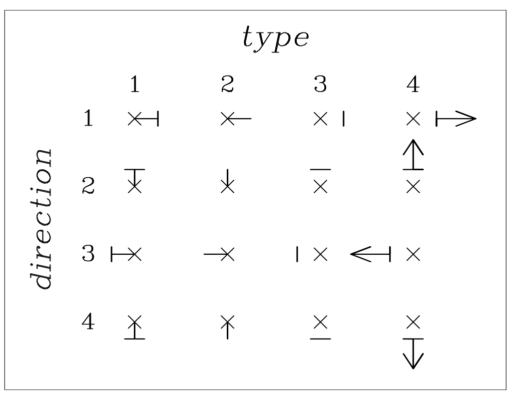
ETYPE style

Change the error bar style. The argument specifies which style to select: Figure 3.5 on page 50 shows an example of commands to draw error bars. The × symbol shows the plotted coordinate value and the error bar is offset by the value of the errors column in the direction specified in the argument to `ERRORBAR`. See the `ERRORBAR` command for drawing the error bars. See the `ULLIM` command for another way to draw upper or lower limit symbols.

Table 4.9: Error Bar Styles

Code	Error Bar Style
1	Standard error bars with a bar and a tick at the end
2	Bar only, no tick at the end
3	Tick only, no bar
4	Upper or lower limit arrow (pointing away from data coordinate)

Figure 4.3: Error Bar Styles



EVALUATE

Data | Saved

XEVALUATE expression
YEVALUATE expression
ZEVALUATE expression
EEVALUATE expression
LEVALUATE expression
PEVALUATE expression
SEVALUATE expression

Replace the data vector in the appropriate buffer by the result of the expression in the string argument. The buffers that can be operated on are shown in Table 4.10 along with the commands used to read in the data to the buffer.

Table 4.10: EVALUATE Commands and Data Types

Command	Data Read by...	Data (Buffer) Type
`EEVALUATE`	`ECOLUMN`	Error array
`LEVALUATE`	`LCOLUMN`	Limits array
`PEVALUATE`	`PCOLUMN`	Point marker code array
`SEVALUATE`	`SCOLUMN`	Scratch data
`XEVALUATE`	`XCOLUMN or XSECTION`	X data array
`YEVALUATE`	`YCOLUMN or YSECTION`	Y data array
`ZEVALUATE`	`ZSECTION`	Image pixel values

Typical expression syntax may be used to specify operators: +, -, *, /, **, and common functions: sqrt, sin, cos, log, log10, exp, etc. See "Expression Evaluation" on page 46 for a more complete listing of the functions that can be used to operate on the data. The current value of any of the four plot data vectors is specified by the single (case-insensitive) character: `x`, `y`, `e`, or `p`; the element number may be specified by `r`, and the number of elements in the vector by `n`. For example,

```
EEVALUATE log10(r**2)
```

will replace each error value by the log of the square of the element number ($e_i = \log i^2$), while

```
YEVALUATE log10(y)
```

will take the logarithm of the Y data values, equivalent to using the `YLOGARITHM` command. The same can be done for the X data using `XEVALUATE`.

For image data, the command

```
ZEVALUATE log10(z+10)
```

will add 10 to the value of each pixel and then take the log of the image.

An array may operate on itself.

`XYEVALUATE` may be used in place of `XYLOGARITHM`.

EXPAND

Attribute **Saved**

EXPAND
EXPAND size

Set the marker and text size. The optional parameter specifies the size as a multiple of the default size, 0.0125 times the smallest dimension of the device. Note that this base size is constant, regardless of the window, virtual page, or viewport. That is, characters and points will not be scaled to the size of the viewport as a fraction of the full page, but will be drawn the same size on a full page plot as on a plot in a small window. So if you wish to retain the relative size of objects on a pane of a windowed screen, you must manually reset the size using `EXPAND`. The argument is always the final size as a multiple of the base size, *not* the previously specified size.

If no argument is specified, the current setting of the size is printed, with the resulting character and point sizes in NDC units.

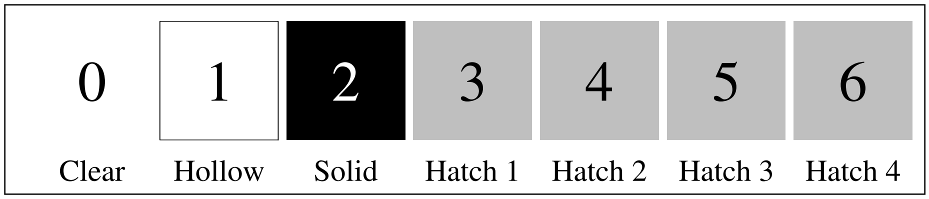
FILLPAT

FILLPAT pattern

Specify the fill pattern for hollow symbols and bar graphs. The integer argument is an index for the device-dependent pattern. A value of zero indicates to fill with a clear pattern, or to erase any existing graphics. A value of one indicates a hollow pattern with an outline only.

If no argument is specified, the current setting of the pattern index is printed. Patterns are shown in Figure 4.4.

Figure 4.4: Filled-Area Patterns



FITPIX

FITPIX
FITPIX left right bottom top

Specify the location of the viewport (plot axes) and force the shape of the viewport to be the same as the pixmap raster image input by `ZSECTION`. The resulting viewport will, in general, not actually span the edges specified by the command arguments. The bottom right corner is fixed, but the right or top edge will be changed to adjust the aspect ratio of the viewport to match the aspect ratio of the image as rendered on the output device.

Note that this is device-dependent in the sense that different devices have different aspect ratios and the image, while having the same aspect ratio, may not occupy exactly the same location on the page. Both device and image pixels are assumed to be square. The computed viewport depends on the shape of the device, the input image, and the edges specified to `FITPIX`.

For example, the following commands on a “landscape” format device would result in a viewport that was narrower than the one specified.

```
igi> zsection myimage[1:100,1:200] # Read a "portrait" image
igi> fitpix .25 .75 .25 .75 # Use centered, half-size display
igi> limits; zrange # Autoscale
igi> pixmap # Display
```

The arguments specify the edges of the viewport as a fraction of the virtual page. If `FITPIX` is used with no arguments, the edges of the current viewport are listed. If `FITPIX` is not used, the default is to fill the viewport, which in general matches the virtual page, except for margins outside the axes for tick labels. This is useful to smear (widen) a narrow image section, such as a spectrum, to cover the viewport. `LOCATION` or `PHYSICAL` may be used to specify a viewport in this case. If the first (only) argument is INDEF, then `FITPIX` will use the currently specified viewport but adjust the top right corner to match the aspect ratio of the input image.

Note that the virtual page itself may be a subset of the display area. See `PHYSICAL`, `VPAGE`, and `WINDOW` to specify the virtual page.

FLIP

Scaling **Saved**

XFLIP
YFLIP

Flip an axis.That is, reverse the direction of the X or Y axis. Results in plotting data and axis labels increasing right to left or top to bottom instead of the default left to right or bottom to top. This must be used before the axes are drawn (`BOX`) for the plot to scale to match the data. This is equivalent to specifying `LIMITS` with the lower and upper limits reversed, but operates on the currently set limits, whether set manually or automatically. This command does not modify the values in the data buffers.

FMTICK

Attribute **Saved**

FMTICK format

Specify the print format for axis tick labels. The optional argument format is a string containing a Fortran or SPP[1] print format specifier. Note that in general, axis labels are floating point values. By default, **igi** tries to format the labels logically, including superscripts for exponential notation on large numbers. If there is no argument, reset to the default formatting. Note that overriding the default precludes formatting using superscripts for exponential notation.

See `AXIS` and `BOX` for drawing axes and tick labels.

1. See the *SPP Reference Manual*, copies are available from the STScI Help Desk.

FONTSET

Attribute **Saved**

FONTSET
FONTSET fonts
FONTSET fonts mono

Select the set of fonts to use in drawing text. The argument is a string specifying **igi** or gio fonts. If no argument is specified, `FONTSET` shows the current font set in use. The choices for the `fonts` argument are `soft` or `igi`, which tells **igi** to use the software outline (igi) fonts or `hard` or `gio` which specifies hardware (gio) fonts. In the former case, the text will appear similar regardless of the device used to display or print the plot. In the latter case, the appearance of the fonts depends on the device. This is primarily intended to support the PostScript fonts resident in a laser printer or those fonts used by a PostScript interpreter, which are supported using the PostScript graphics kernel, `psikern`

If mono is set, non-proportional fonts will be used, such as Courier, until reset with another call to FONTSET without the mono parameter specified.

Figure 4.5: Differences Between igi and gio Fonts

a) igi Fonts Default *Italic* $^{\text{Super}}$Script $_{\text{Sub}}$Script F i x e d

b) PostScript TImes (gio) Normal *Italic* **Bold** Γρεεκ Roman

c) PostScript Helvetica (gio) Normal *Italic* **Bold** Γρεεκ Roman

Figure 4.5 shows examples of the font sets. In line **a**, the **igi** fonts were used with **igi** escape sequences. In lines **b** and **c**, gio fonts were used with the PostScript kernel. Note that this kernel permits specifying the PostScript fonts to use for the various gio fonts. The Times family of fonts was used for line **b** and Helvetica for line **c**.

In both cases, the `EXPAND` command is used to specify the size, `ANGLE` the rotation, `COLOR` sets the hardware(kernel)-specific color, and `JUSTIFY` the justification. `LWEIGHT` may be used to change the line width used to draw **igi** characters, but does not effect gio characters.

Embedded control characters specify further attributes of the plotted text. Note that there are different escapes for the font sets.

Italics (`\\i`) and proportional spacing (`\\f`) behave as toggles for **igi** fonts, enclosing a (sub)string in matching escapes sets the attribute for that string only. Superscript and subscript each undoes the action of the other, to write a substring as a super(sub)script, enclose it in matching pairs of `\\u`...`\\d` or `\\d`...`\\u`.

GRID

Plotting **Saved** **Undo**

Draw lines connecting major tick marks on the axes drawn by the `BOX` command. Note that `BOX` must have been used first to compute the tick spacing. `GRID` uses the current value of the line style (solid, dotted, etc.) and the line width in drawing the grid. See `LTYPE` to set the line type and `LWEIGHT` to set the line width.

HELP

Execution

HELP
HELP [command]

Page the help text. If the optional argument is present, a description of that command is listed. See `!help help` for a description of the IRAF `help` command. The optional argument may also be used to list any major section of the **igi** help text. For example, to list the general description of **igi**, you would type:

```
HELP description
```

See `APROPOS` for listing commands by keyword and `?` for a list of the commands.

HISTOGRAM

Plotting **Saved** **Undo**

Draw a histogram style plot (bar graph) through the current X and Y data. If no X data exist, the horizontal axis will be scaled on the pixel numbers. *Note that the command name refers to the curve style, not to actual binning of data.* See `LTYPE` and `LWEIGHT` for specifying the style of the curve to draw. See `CONNECT`, `STEP`, and `POINTS` for different curve styles

ID

Plotting | Saved | Undo

Draw a standard identification text string vertically to the right of the right Y (vertical) axis. The label includes the user name, time, date, etc., for example:

```
igi STScI/IRAF V2.5 LEVAY@scivax Fri 10:25:50 15-Apr-88
```

IMGWCS

Data | Saved

Specify whether to use world coordinates from an image as the `X` plot buffer (see the `YSECTION` command). This command operates as a toggle, By default `SECTION` will *not* fill in the X buffer when reading the `Y` buffer. If the `IMGWCS` command is used, the next use of `YSECTION` will replace the contents of the X buffer by computed values based on the world coordinate information in the image. Using `IMGWCS` again will toggle this state. Note that this command together with `YSECTION` will modify the contents of the plot buffer.

INPUT

Execution

INPUT filename

Execute commands from a file. The commands are executed as they are read. Only the `INPUT` command is written to the command buffer, not the individual commands. Use the `READ` command to input the commands into the buffer without execution.

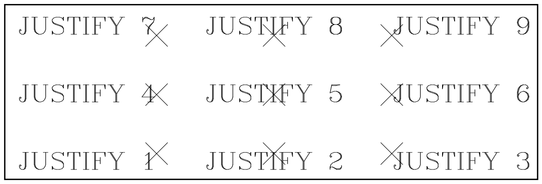
JUSTIFY

Attribute | Saved

JUSTIFY justification

Set the justification for text drawn using the `LABEL` command. See `PUTLABEL` for more information. Figure 4.6 shows the results for various argument values. The × symbol indicates the coordinate of the pen when using `LABEL`.

Figure 4.6: Text Justification Specified with JUSITY and PUTLABEL



LABEL

Plotting | Saved | Undo

LABEL string
XLABEL
XLABEL string
YLABEL
YLABEL string

Draw text in the parameter `string` at the current position or label the appropriate axis. The string is assumed to terminate at the end of the line, or may be enclosed in single or double quotes to enter more than one command on the line. The size of the text may be specified with the `EXPAND` command, the orientation may be set using the `ANGLE` command, and the justification may be specified with the `JUSTIFY` command.

Roman, Greek, Script, and small Type fonts may be selected within any label using the `\r`, `\g`, `\s`, or `\t` escapes. Any font may be italicized by using `\i` escape, and any font may be printed in fixed rather than proportional spacing by using the `\f` escape. A single backslash (`\`) sets the attribute for the following single character only. Two backslashes (`\\`) sets the attribute for the remainder of the string, or until it's reset by the opposite attribute.Embedded control characters specify further attributes of the plotted text:

Italics (`\\i`) and proportional spacing (`\\f`) behave as toggles. Enclosing a string in matching escapes sets the attribute for that string only. Superscript and subscript each undoes the action of the other.

To write a substring as a superscript or subscript, enclose it in matching pairs of `\\u`...`\\d` or `\\d`...`\\u`.

XLABEL

Draw text centered below the bottom X (horizontal) axis. If the optional argument is missing, **igi** uses an internal string. This is filled by the `DATA` and XCOLUMN commands. If the DATA command specifies a binary STSDAS table, then the XCOLUMN command appends the column name to the string.

YLABEL

Draw text centered below the bottom Y (vertical) axis. If the optional argument is missing, **igi** uses an internal string. This is filled by the DATA and YCOLUMN commands. If the DATA command specifies a binary STSDAS table, then the YCOLUMN command appends the column name to the string.

▽ Neither the X nor Y label will appear if there is not enough room between the bottom axis and the edge of the virtual page or device border, i.e., the bottom of the screen or page.

LIMITS

Scaling | Saved

LIMITS
LIMITS left right bottom top

Set the plot scale, i.e., the user (world) coordinates assigned to the edges of the plot (viewport). If no parameters are specified, then the limits will be set automatically based on the range of current data values. Note that in general, the current viewport is not the entire screen.

The scaling set by LIMITS may be adjusted to include a border between the data extremes and the axes using the MARGIN command. See BOX for drawing axes at the current plot scale.

LINES

Data | Saved

LINES first last

Specify the range of lines to read from the input data file using the DATA and [XYEPLS]COLUMN commands. Blank lines and comment lines in ASCII files will be ignored by this command. If no parameters are specified, then the current range will be listed. Specifying `first = INDEF` is the same as `first = 1`; `last = INDEF` will read to the last row in the input file. Specifying one argument will read only that one line. This does not apply to reading image sections (See [XYEPLS]SECTION).

LIST

Execution

LIST
LIST macro

List the command buffer. The optional argument is used to list the text of a defined macro. See MACROS to list defined macro names.

LOCATION

Scaling **Saved**

LOCATION
LOCATION left right bottom top

Specify the location of the plot axes (viewport) as a fraction of the virtual page. The default is to fill the virtual page except for margins outside the axes for tick labels. Note that the virtual page itself may be a subset of the display area. See PHYSICAL, VPAGE, and WINDOW to specify the virtual page. Table 4.11 shows the default values for the arguments to LOCATION. The SHOW command can be used to determine the currently set values of LOCATION.

Table 4.11: Viewport Location Arguments

Argument	Default
left	0.1
right	0.95
bottom	0.1
top	0.95

LOGARITHM

Data | Saved

XLOGARITHM
YLOGARITHM

Take the common logarithm of the data values in the X or Y buffer. If any value is less than or equal to zero, the result is made `indef`, causing that point to be ignored in scaling and plotting. See `EVALUATE` for an alternate way of doing this and to operate on buffers other than `X` or `Y`. You must use `[XY]LOGARITHM` for `ERROBAR` to work correctly on log data.

LTYPE

Attribute | Saved

LTYPE style

Set the line style. The integer parameter selects a hardware specific line type. Note that not all devices implement the clear or erase line type (–1). The default is solid lines (0). (See Figure 4.7.)

If no parameter is given, the current setting is listed. The default is solid.

Figure 4.7: Line Types Selected with LTYPE

6	dashed
5	dotdash
4	dotdash
3	dashed
2	dashed
1	dotted
0	solid (default)
-1	clear (erase)

LWEIGHT

Attribute **Saved**

LWEIGHT
LWEIGHT width

Set the line width. A floating point parameter specifies the factor to multiply the default hardware line width. Not all devices support selectable line width. The line width applies to any drawing command including curves and text. The default is 1.

Figure 4.8: Line Weights Specified with LWEIGHT

MACROS

Execution

Print the names of defined macros and the number of their arguments. This command has no arguments and is not saved in the command buffer.

See `DEFINE` to create a new macro.

MARGIN

Scaling **Saved**

MARGIN
MARGIN fraction

Adjust the current plot scale to provide a border between the data extremes and the axes. The optional argument specifies the size of the margin as a fraction of the current viewport. If the argument is missing, the default margin is 0.025, or 2.5% of the size of the viewport. See `LIMITS` for scaling the plot to data. MARGIN really makes sense only if it is used between the LIMITS and BOX commands. Its effect will be undone by another LIMITS command.

MENU

Execution

Print a list of the **igi** commands with a single line description. The following listing shows sample output from the MENU command.

Figure 4.9: Partial Output of the MENU Command

```
ANGLE       Set the marker and text rotation angle
APROPOS     List commands associated with a keyword
AXIS        Draw and label an arbitrary axis
BOX         Draw and label the axes
BYE         Terminate igi (alias for END)
COLOR       Set (hardware-dependent) color index
CONNECT     Draw a (polyline) curve connecting the data
CURSES      Read back cursor position (IRAF cursor mode)
DATA        Specify the input text data file
DDRAW       Pen down move (draw) in NDC
```

MINMAX

Data **Saved**

Print the range of data values in both the X and Y arrays. This command has no arguments and is not saved in the command buffer.

MOVE

Plotting **Saved**

MOVE xpos ypos
DMOVE xpos ypos
PMOVE xpos ypos
VMOVE xpos ypos

This is an alias for RELOCATE. Pen up move, i.e., change the current position to the coordinates specified by the two floating point parameters. The coordinates need to be specified in the coordinate system appropriate for the command used, as given by Table 4.13. See `RELOCATE` for more details.

NOTATION

Attribute **Saved**

NOTATION xlow xhigh ylow yhigh

Set the format for axis labels drawn by `BOX`. The parameters `xlow` and `xhigh`, `ylow` and `yhigh` specify the range of values within which axis label values will be written as floating point values and outside which they will be written in exponential notation. If `xlow = xhigh`, all values on the X axis will be written in exponential notation, and similarly for the Y axis. Exponential values are written using Fortran (SPP) syntax, `1.0E+8` for example.

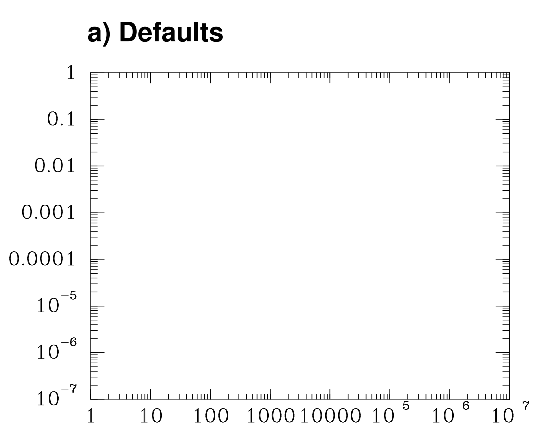
Table 4.12: Boundaries at Which Scientific Notation is Used in Labels

Argument	Default
`xlow`	10^{-4}
`xhigh`	10^{+5}
`ylow`	10^{-4}
`yhigh`	10^{+5}

Figure 4.10 shows two examples of axes. Graph **a** was drawn using the default notation. Graph **b** was drawn with the following command:

```
notation 1e−3 1e+4 1e−3 1e+4
```

Figure 4.10: Effect of Using NOTATION



NUMBER

Plotting | Saved

NUMBER

Draw the element number at the coordinates of each data point. If no X vector is defined, then use the element number as the X coordinate. If both an X and Y vector is defined, use those values as the X and Y coordinate. The current size, line width, rotation angle and text justification are used for the attributes of the string. See Figure 3.5 on page 50 for an example.

PAGECMD

Execution

PAGECMD
PAGECMD macro

Page the command buffer, i.e., invoke the IRAF `page` command via the cl to list the stored commands. The optional argument is the name of a defined macro, whose text may be paged instead of the command buffer.

PHYSICAL

Scaling | Saved

PHYSICAL
PHYSICAL left right bottom top

Set the plot location on the device in physical units (inches). This is analogous to the `VPAGE` command that sets the virtual page on the device. While `VPAGE` uses normalized device coordinates (NDC), `PHYSICAL` uses real device coordinates, i.e., inches[2]. The `LOCATION` command may be used to specify the viewport (location of the axes) in normalized coordinates on the virtual page. The `LIMITS` command defines the transformation between data, world, or user coordinates (WC) and the viewport.

Note that accurate, absolute coordinate scaling depends on the correct values specified for the particular graphics device in the IRAF `graphcap` file. In addition, specific printers might vary somewhat in their absolute scale, even from print to print.

2. Well, we can't all be dragged into the 20th century, can we?

Figure 3.3 on page 42 shows an example of using PHYSICAL to draw a grid in inches.

PIXMAP

Plotting **Saved**

Render (display) an image if data has been read into the Z buffer using the ZSECTION command. The image is displayed using the current pixel scaling specified by LIMITS, the brightness scale specified by ZRANGE in the current viewport, set by FITPIX, LOCATION, or PHYSICAL.

PLAYBACK

Execution

Execute the commands in the command buffer. Note that any attributes (point style, size, etc.) set by the already executed commands may influence the action of the playback.

POINTS

Plotting **Saved** **Undo**

Draw a marker at each data coordinate according to the code read from the data file by PCOLUMN. The floating point data values are assumed to represent the marker style and size for each data value. See PTYPE for coding the marker style. The size of the markers is specified by the current size parameter and may be set using the EXPAND command. The marker may be rotated by the current angle parameter, which may be set using the ANGLE command. See the DOT command to draw a single marker at the current pen position. See the ELLIPSE command to draw a single ellipse (or circle) at the current pen position.

POLYGON

Plotting **Saved** **Undo**

Draw a closed, filled polygon connecting the data coordinates. The COLOR command affects the color of the filled area and the FILLPAT command determines the pattern used to fill the polygon. See CONNECT to draw a curve through the points. See POINTS for drawing markers without connection. The current pen position is left on the last point.

This command is saved in the command buffer. The results of this command may be erased with UNDO.

PSFONT

Attribute **Saved**

PSFONT fontname

Specify which printer font to use in any text output which follows this command. This will only be active when FONTSET is set to ‘hard’. A new font can be specified every time this command is called. The font is specified in a text string with the escape code ‘\fP’ and will be active in the string until reset to the default font with the escape code ‘\fR’ (for Roman font).

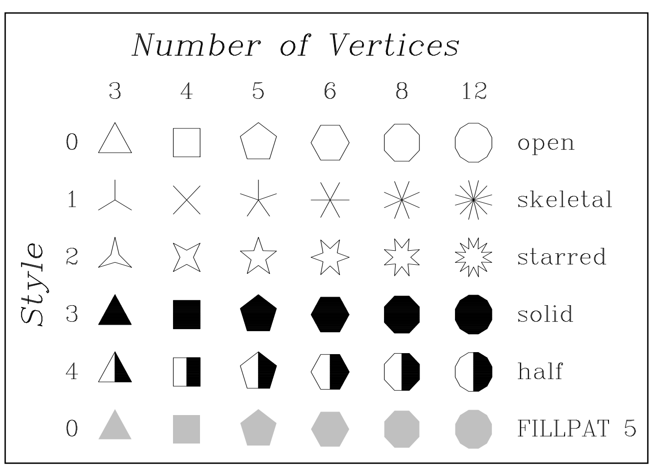
PTYPE

Attribute **Saved**

PTYPE
PTYPE vertices style

Set the marker type. The argument vertices specifies how many vertices are in the marker, and the argument style selects one of the marker styles either as an integer code or a string keyword. Figure 4.11 shows examples of various styles.

Figure 4.11: Types of Plotted Symbols Specified with PTYPE



Any value larger than 4 is read as 4 (half-filled symbol type). Alternately, the style may be specified as a string with the value given in Figure 4.11.

An open symbol will be plotted with the current fill pattern, specified using the `FILLPAT` command. The default fill pattern is hollow. Other fill patterns depend on the device and kernel.

If the number of vertices is –1, 0, or 1, then `DOT` or `POINTS` draws the smallest plottable point. If the number of vertices is less than –1, `DOT` or `POINTS` draws markers defined by the IRAF marker drawing routine `gmark` as described in the gio reference manual[3].

To draw a circle, use a large number of vertices, 25, perhaps. To draw small points, use a small size, `EXPAND .1`, for instance. See the `POINTS` command to draw markers at the data coordinates, the `DOT` command to draw a single marker at the current pen position.

PUTLABEL

Plotting | **Saved** | **Undo**

PUTLABEL justify text

Draw text in the argument string relative to the current coordinates according to the argument justify.

The size of the text may be specified with the `EXPAND` command and the angle may be set using the `ANGLE` command. Embedded control characters specify further attributes of the plotted text. See Table 3.10 on page 61) for the text escape characters and Figure 3.17 on page 60 for examples of its usage.

Also see the `LABEL` command. This command is saved in the command buffer.

READ

Execution

READ filename

Read commands into the command buffer without execution. Commands come from the command file specified by the filename parameter. See `WRITE` to make a script file from the command buffer. See

3. Type `help gio$doc/gio.hlp fi+` in the cl.

INPUT for executing commands from a script file. Also see EDITCMD, LIST, and PAGECMD.

RELOCATE

Plotting **Saved**

RELOCATE xpos ypos
DRELOCATE xpos ypos
PRELOCATE xpos ypos
VRELOCATE xpos ypos

MOVE xpos ypos
DMOVE xpos ypos
PMOVE xpos ypos
VMOVE xpos ypos

Pen up move, i.e., change the current position to the coordinates specified by the two floating point parameters. The coordinates need to be specified in the coordinate system appropriate for the command used, as given by Table 4.13. The MOVE commands, shown in the table, are simply aliases for the corresponding RELOCATE commands, and can be used interchangeably.

Table 4.13: RELOCATE Commands and Coordinate Systems

Command	Alias	Coordinate System
RELOCATE	MOVE	World Coordinates (WC)
DRELOCATE	DMOVE	Normalized device coordinates (NDC)
PRELOCATE	PMOVE	Physical coordinates (PC)
VRELOCATE	VMOVE	Virtual page coordinates (VPC)

If any RELOCATE or DRAW command is used in cursor mode (i.e., :RELOCATE, :DRELOCATE, :MOVE, or :VMOVE), xpos and ypos are ignored and should be omitted. In fact, it is unnecessary to use any move command in cursor mode. Issuing any command or request for readback resets the current position to the coordinates of the cursor.

RESET

Attribute **Saved**

Return settable plot and data parameters to defaults.

SAOCMAP

Data **Saved**

SAOCMAP cmapfile

Apply a colormap to an image rendered using `PIXMAP`. The argument is the name of a plain text file containing a colormap written by SAOimage. This consists of a functional description of a mapping between stored and displayed pixel values for the three display colors: red, green, and blue, as shown in the example below.

Figure 4.12: Sample SAOimage Colormap File

```
# SAOimage color table
# saodump - SAODUMP[1/1]
# Fri Mar 13 15:53:40 1992
PSEUDOCOLOR
RED: gamma 1.504
(0.000,0.000)(0.500,1.000)(1.000,1.000)
(0.000,0.000)(0.250,0.000)(0.750,1.000)(1.000,1.000)
GREEN: gamma 1.504
BLUE: gamma 1.504
(0.000,0.000)(0.500,0.000)(1.000,1.000)
```

Color display depends on the graphics kernel and the output device. Only the PostScript kernel (`psikern`) supports this feature.

SECTION

Data | Saved

XSECTION image project
YSECTION image project
ZSECTION image project
ESECTION image project
PSECTION image project
LSECTION image project
SSECTION image project

This family of commands reads a column of data from an IRAF (OIF, or STF) image. The image may be of any dimensionality and data type and may include an image section specification that might be a subraster of the image or may include a group number. By default, all of the image section pixels are read as a one-dimensional array into the specified **igi** vector.

It is not possible to extract an arbitrarily oriented vector from a multidimensional image. (See the **pvector** task in the **plot** package.)

`[XYEPLS]SECTION` overrides the use of the `DATA` and `[XYEPLS]COLUMN` commands in filling the data arrays. One array may be filled with an image and another by a table column using `DATA` and `[XYEPLS]COLUMN`, yet another from a text file, also using `DATA` and `[XYEPLS]COLUMN`.

Multi-dimensional images are read pixel by pixel, line by line, plane by plane, and so forth. For example, a 2-D image is read from left to right along each line from the bottom to the top. Optionally, a multi-dimensional section may be reduced to a single dimension by averaging or summing all of the image lines into a single vector the size of the lines in the image section. The optional argument project is an integer.

For example, the command:

```
igi> XSECTION myimage[1:10,1:10]
```

will read 100 pixels from the lower left corner of `myimage` into the X buffer. The command:

```
igi> XSECTION myimage 1
```

will read all of the pixels but will store the average of all of the image lines in the X buffer.

```
igi> YSECTION myimage -2
```

This command will add the columns of `myimage` and store the result in the Y plot buffer.

XYSECTION saves the input file name string with the section specification in the X or Y axis label buffer. This may be used with the XYLABEL command with no arguments to label the X or Y axis with the section.

World Coordinates and YSECTION

It is possible to use world coordinates from the input image as the X values corresponding to Y values read using YSECTION if the appropriate parameters exist in the image header. By default, these coordinates will not be used and you must supply the appropriate independent variable as the X values. However, if you use the IMGWCS command and the image contains the appropriate transformation parameters, then YSECTION will automatically fill in the X plot buffer with the coordinate values.

Note that in general this works only for inherently one-dimensional data, e.g., spectra. For 2-D data such as images, coordinates are rather meaningless for an arbitrary 1-D section of the image. You will get an *identity vector* if the coordinate parameters are not present in the image. That is, the X values will be the element number of the extracted pixels. This is not exactly the same result as not using the WCS, since the coordinate transformation is applied to the image section. That is, the returned coordinates are the coordinates with respect to the original full image, not the extracted piece. For example, if you use:

```
igi> imgwcs
igi> ysection dev$pix[10,50,1]
```

the range of the X values will be 10 to 50 rather than 1 to 41 if you do not use IMGWCS and do not otherwise fill in the X vector. Even if the image does not contain a WCS transformation, YSECTION will provide an X vector that corresponds to the pixel numbers in the full input image.

The YSECTION command, if used with IMGWCS, changes the contents of the X plot buffer (see above).

ZSECTION

Read a one- or two-dimensional image section into the Z buffer. The data are interpreted as pixels in a grayscale image rather than coordinates of discrete points.

See ZRANGE, PIXMAP, and LIMITS for scaling and displaying the data.

See WCSLAB for labelling the image using WCS coordinate information from the image header.

See ZEVALUATE for manipulating the pixel values.

See SAOCMAP for specifying a gray or color mapping for the image display.

Note that not all graphics devices support image display.

SHOW

Attribute

Print the plot limits and attributes using the IRAF page facility. This is simply informational output and does not change the plot.

SIXTY

Plotting | Saved | Undo

XSIXTY
XSIXTY digits
YSIXTY
YSIXTY digits

This command will convert the numbering of the axes to use sexagesimal numbering for the axis labels. The optional parameter `digits` specifies the number of decimal places of precision used for the labels. This command must be called prior to outputting the plot axes using the BOX command for plots.

STEP

Plotting | Saved | Undo

Draw a stepped curve, i.e., a histogram or bar graph without vertical lines, through the current data values. See Figure 3.5 on page 50 for an example.

TICKSIZE

Attribute | Saved

TICKSIZE
TICKSIZE minorx majorx minory majory

Set the intervals in WC for tick marks on the X and Y axes drawn by `BOX`. The arguments `minorx` and `minory` specify the spacing for minor (small, unlabeled) ticks on the X and Y axes; `majorx` `majory` specify the spacing for major (large, optionally labeled) ticks on the X and Y axes. If `minorx` or `minory` is less than zero, that axis will have logarithmic tick spacing with major ticks at each decade and minor ticks at each integral value between.

See `BOX` for drawing the axes and `NOTATION` for specifying labels. See `AXIS` for drawing an arbitrary axis.

TITLE

Plotting | Saved | Undo

TITLE text

Draw text centered above the top X (horizontal) axis. The text escape characters may be used to specify different fonts, superscripts and subscripts (See `LABEL`).

Note that the title will not appear if there is not enough room between the top axis and the edge of the virtual page or device border, i.e., the top of the screen or page.

ULLIM

Plotting | Saved | Undo

Draw a limit symbol (an arrow) to indicate upper or lower limits. The base of the arrow is drawn at the coordinates in the X and Y columns. If no X data are specified, the X coordinate is the element number. The direction of the arrow depends on the value in the limits column. Values less than zero indicate lower limits and result in an upward arrow. Values greater than zero indicate upper limits and result in an downward arrow. The current size determines the size of the arrow.

See the `DATA`, `XCOLUMN`, `YCOLUMN`, `LCOLUMN` commands. See the `ERRORBAR` and `ETYPE` commands for drawing error bars and an alternate way of drawing upper or lower limit symbols.

UNDO

Plotting

Selectively erase the result of the last command that produced plotted output. This command will not restore the same figure if called again.

Note that this only works if the device supports erasing vectors.

VERSION

Execution **Saved** **Undo**

Show the current **igi** version and date of its initial installation. This information is also printed in the initial prompt when starting **igi** in an interactive session, in the first line of the output from the `SHOW` command and in the text written to the plot using the `ID` command. This command will produce a text line such as:

```
Version 28 Feb 97
```

VPAGE

Scaling **Saved**

VPAGE
VPAGE left right bottom top

Specify the edges of the virtual page on the plotting area of the device as a fraction of the width and height. For example, to use only the upper left corner of the frame, use `VPAGE 0.5 1.0 0.5 1.0`. The default is the entire plot area. This does not change the relative location of the axes on this page (see `LOCATION`).

See `PHYSICAL` to specify the page in device coordinates (inches).

WCSLAB

Plotting **Saved** **Undo**

WCSLAB
WCSLAB edit

Label the currently specified viewport with WCS coordinate information, such as those from the image header. This information would include equatorial (celestial) coordinates as produced by the STSDAS **stplot.wcslab** task.

The optional command argument is a string that begins with “e” or “E”. This allows editing of the **wlpars** parameter set for specifying attributes via **eparam**.

If the `ZSECTION` command was used to read in an image section into the Z buffer, then the WCS attributes of that image are used in the labeling of the display. Otherwise, the WCS information is taken from parameters in the **wcspars** pset.

Note that the psets may be edited before entering **igi** by explicitly assigning parameter values or by using **eparam** at the cl prompt. In addition, the parameter values may be assigned or the psets edited from **igi** using the “!” escape before using the `WCSLAB` command.

> Some attributes specified in the **wlpars** pset conflict with attributes set using **igi** commands—such as text size and color. Currently, the **wlpars** parameters take precedence over any attributes set within **igi**.

WINDOW

Scaling | Saved

WINDOW nx ny pane
WINDOW pane
WINDOW nx ny
WINDOW

Divide the device plotting area into `nx` by `ny` windows (tiles or panes) and select the pane in which to plot, numbered left to right, bottom to top. For example, `WINDOW 3 3 3` will create nine panes and select the bottom right pane for subsequent plotting. If only two arguments are given, the current pane number is selected, but with a different division. If only one argument is given, a different pane is selected in the same tiling. `WINDOW` is an alternative to `PHYSICAL` or `VPAGE` to specify the virtual page on the device.

WRITE

Execution

WRITE filename

Write the command buffer to the file specified by the filename parameter.

ZRANGE

Scaling **Saved**

ZRANGE
ZRANGE min max

Specify the minimum and maximum image pixel values that map to the darkest and brightest display values. Using no arguments indicates that the task should use the full range of values in the image data. In fact, min may be greater than max, which will result in the lower-valued pixels rendered brighter than the higher-valued pixel—a negative image.

APPENDIX A: Source for Figures

In This Appendix...

This appendix includes the **igi** source scripts for several of the figures in this document. These scripts exemplify various features of **igi**. They are more extensive than the short examples provided within the text, and they demonstrate capabilities other than simple plotting. See the particular section of the document for the plot resulting from these commands. The first three scripts relate to the examples given in Chapter 1, and contain comments to explain more clearly what the scripts are doing. All scripts presented here are designed to not only demonstrate what can be done with **igi**, but how they were done as well.

A Sample Log Plot

The following script creates a simple log plot based on randomly generated numbers shown in Figure 1.1. The data used, of course, is totally fabricated and bears no intentional resemblance to any real observations or theory. This script contains comments describing the function of each command. In this example, the X data is simply the number of the value as it is read in.

Figure A.1: Source for Sample Plot

```
data rand.dat                      Read data from the file rand.dat
ycol 3; yevaluate y/(r+10)         Read column 3 into the Y buffer and operate on it
limits; margin 0.04                Scale the plot with a 4% margin
expand 1; ptype 3 4
points                             Draw data points, plotting line number vs. (y/r+10)
ecol 3                             Read errors from column 3
eev sqrt(y)/5
etype 1; errorbar 2; errorbar -2
lines 1 20; ycol 2                 Read the first 20 lines of column 2 into the Y buffer
yevaluate 1/(r+10)
limits; margin 0.04
connect
limits 1 50 -1 4                   Rescale and draw the axes
margin 0.04
ticksize 1 10 -1 0;
notation 1e-4 1e5 2e-3 2e2
expand 1.2; box                    Draw the labels (note the embedded
ltype 1; grid                      text formatting escapes)
xlabel “\\iSample Number”
ylabel “\gx [\\i\gms \gV\\u-1\\d\\i]”
title “\\iObserved versus Theoretical Data”
```

The input data contained in the file named `rand.dat` can be constructed using the IRAF command `urand` in the **utilities** package. The following cl command would produce a file suitable for input to the above **igi** script:

```
cl> urand 50 6 ndigits=3 scale=100 >rand.dat
```

A Labelled Spectrum

This script produces the spectral plot in Figure 1.2. Due to the length, and repetitive nature of the labels, some of those commands were omitted from this listing. In addition, the data files is not shown here.

Figure A.2: Source for Sample Plot

```
imgwcs                                      Use WC from data
YSECTION n6543.hhh                          Read spectrum data
LIMITS 3530 5610 -10 340                    Scale the plot
expand 0.9                                  Size for text
BOX                                         Draw the axes
CONNECT                                     Draw the curve
YLABEL "\\i Counts"                         Y axis label
xlabel "\\iWavelength [\\gV]"               X axis label
expand 1;  VMOVE .1 .96
PUTLAB 9 "\\iFOC Long Slit"                 Plot title


VMOVE .95 .96;  EXPAND 0.6
PUTLAB 7 "\\iCourtesy P. Hodge, ST ScI"
justify 6; angle 90; expand .8              Text attributes
move 3727 58; draw 3727 68                  Label spectral lines
move 3727 71; label OII
move 3869 63; draw 3869 73
move 3869 76; label NeII
move 3889 25; draw 3889 30
draw 3915 40; draw 3915 45
move 3915 48; label HeI, H\\gz\\r
```

Additional commands omitted

A Complex Image Display

Output from the paper products code consists of many pages of tables, figures, and images, all produced by **igi** scripts. The complexity of the output means that the **igi** scripts are generally quite large, relying on the results of calculations made in the paper products code itself. Therefore, only a few select sections of the **igi** script that went into creating Figure 1.3 are shown here.

Figure A.3: Portion of Script for Producing Paper Products Output

Additional Commands Omitted

```
vpage 0.26 1.0 0.01 0.99; fontset hard
location 0. 1. 0. 1.
ptype 3 0; expand 0.7
vmove 0.5931308 1.0100000
angle 60; dot
vmove -0.0100000 0.4168687
angle 30; dot
angle 0; expand 1
zsection u2440101t.c0h[1] 2 90
fitpix 0.2715 0.5 0.5 0.7285
limits; zrange 3573.7  138.0
pixmap
zsection u2440101t.c0h[2][-*,-*] 2
fitpix 0.0 0.5 0.0 0.5
limits; zrange 3573.7  138.0
pixmap
zsection u2440101t.c0h[3] 2 -90
fitpix 0.5 1.0 0. 0.5
limits; zrange 3573.7  138.0
pixmap
zsection u2440101t.c0h[4] 2
fitpix 0.5 1.0 0.5 1.0
limits; zrange 3573.7  138.0
pixmap
vmove 0. 0.5; vdraw 1.0 0.5
vmove 0.5 0.0; vdraw 0.5 1.0
zsection dev$pix[1:128,1]
zevaluate r
limits
zrange 128. 1.
location 0.050 0.450 0.870 0.900
pixmap
```

Specify Section for Images

No Margins Around Image Section

Read in PC Image, Block-Average by 2, Rotate by 90˚

Scale image

Read in WF2 Image, Flip by 180˚, Block-Average by 2

Read in WF3 Image, Block-Average by 2, Rotate by –90˚

Read in WF4 Image, Block-Average by 2

Create Greyscale Bar

(Continued...)

Figure A.3 (Continued): Portion of Script for Producing Paper Products Output

```
vmove 0.050 0.870
vdraw 0.450 0.870
vdraw 0.450 0.900
vdraw 0.050 0.900
vdraw 0.050 0.870
expand 0.700
justify 2
vmove 0.050 0.860; label '137.99'
vmove 0.450 0.860; label '3573.7'
reset                                   Reset Page Coordinates
                                        and Text Settings
vpage 0. 1. 0. 1.
location 0. 1. 0. 1.
justify 5
move 0.33 0.62
draw 0.315 0.695
expand 0.7
move 0.3125829 0.7104092                Create Compass
label 'N'
move 0.33 0.62
draw 0.272 0.601
move 0.2601384 0.5974602
label 'E'
reset
vpage 0. 1. 0. 1.
location 0. 1. 0. 1.
justify 2
move 0.2855198 0.8
draw 0.286 0.785
draw 0.514 0.785
draw 0.514   0.8                        Create Plate-Scale Bar
expand 0.7
move 0.4 0.83
label '50 arc sec'
reset; fontset hard; location 0. 1. 0. 1.
vpage 0.01 0.25 0.01 0.9
limits 0 40 35 0                        Set Text Column
move 40 0; draw 40 35
move 20 0; expand 1; justify 2
label 'u2440101t.c0h'
```

Additional Text Formatting Commands Omitted

Axes

Figure A.4 demonstrates drawing axes using the `BOX` and `AXIS` commands. See Figure 3.12 on page 55.

Figure A.4: Axis Code

```
location .1 .9 .1 .9

WINDOW 2 3 3
vpage 0.05 .45 .3 .7
AXIS 0 10 0.2 1 .1 .1 .9 1 1
ANGLE 90
AXIS -1 11 1 5 0 .2 .7 1 0
ANGLE 60
AXIS 9 21 0 0 .2 .25 .75 2 1
angle 45
AXIS 8 2e4 -1 0 .4 .3 .6 2 0

WINDOW 2 3 1
BOX

window 4
angle 0
AXIS -1.1 2.1 0.1 .5 .1 .1 .8 -1 0
AXIS -1.1 2.1 0.1 .5 .1 .9 .8 0 1
AXIS -1.1 2.1 0.1 .5 .1 .5 .8 0 0
AXIS -1.1 2.1 0.1 .5 .1 .5 .8 0 1
angle 90
AXIS 3 13 0.5 2 .1 .1 .4 2 1
AXIS 3 13 0.5 2 .9 .1 .4 2 0
AXIS 3 13 0.5 2 .1 .5 .4 2 1
AXIS 3 13 0.5 2 .9 .5 .4 2 0

WINDOW 2
limits -1.2 2.2 2.1 0.9
ticksize .1 .5 .05 .2
BOX -1 -1
LTYPE dotted
GRID
```

Curve Styles

This example demonstrates the various styles of plotted curves. See Figure 3.5 on page 50 .

Figure A.5: Code for Generating Curve Styles Example

```
LOCATION 0 1 0 1
LIMITS 0 1 0 1
LWEIGHT 2

LIMITS 0 21 50 1750
BOX 0 0

YSECTION "dev$pix[221:240,248]"
yev y*.65
BARGRAPH 0.75

PTYPE 5 2
YSECTION "dev$pix[221:240,249]"
yev y*0.95 + 52
POINTS

YSECTION "dev$pix[221:240,250]"
yev y + 250
CONNECT

esection "dev$pix[221:240,251]"
eev 1.5*sqrt(e)
YSECTION "dev$pix[221:240,251]"
yev y + 450
ETYPE 1
ERRORBAR 2
ERRORBAR -2

YSECTION "dev$pix[221:240,252]"
yev y + 650
STEP

YSECTION "dev$pix[221:240,253]"
yev y + 900
LSECTION "dev$pix[221:240,273]"
ULLIM



EXPAND 1.0
JUSTIFY 8

RELOCATE 17 540
LABEL "BARGRAPH"

RELOCATE 17 756
LABEL "POINTS"

RELOCATE 14 895
LABEL "CONNECT"

RELOCATE 14 1145
LABEL "ERRORBAR"

RELOCATE 14 1332
LABEL "STEP"

RELOCATE 13 1603
LABEL "ULLIM"

EXPAND 2.0
lweight 2
RELOCATE 0.9 1654
PUTLABEL 3 "\\iCurve Styles"
```

Marker Types

This example demonstrates the style of centered symbols drawn with the DOT and POINTS commands. See Figure 4.11 on page 107.

Figure A.6: Code for Generating Symbol Types

```
LOCATION 0 1 0 1
vpage .15 .85 .15 .85
LIMITS -1.0 7 0 7.75
LW 1.5;  EXPAND 2;  ANGLE 0
RELOCATE 3 6.5;  PUTLABEL 8 \\iNumber of Vertices
ANGLE 90
RELOCATE 0 3;  PUTLABEL 8 \\iStyle
LWEIGHT 1
expand 1.5
ANGLE 0

JUSTIFY 8
RELOCATE 1 5.75;  LABEL 3
RELOCATE 2 5.75;  LABEL 4
RELOCATE 3 5.75;  LABEL 5
RELOCATE 4 5.75;  LABEL 6
RELOCATE 5 5.75;  LABEL 12

justify 4
RELOCATE 0.5 5;  LABEL 0
RELOCATE 0.5 4;  LABEL 1
RELOCATE 0.5 3;  LABEL 2
RELOCATE 0.5 2;  LABEL 3
RELOCATE 0.5 1;  LABEL 4

justify 6
move 5.5 5; label "open"
move 5.5 4; label "skeletal"
move 5.5 3; label "starred"
move 5.5 2; label "solid"
move 5.5 1; label "half"

EXPAND 2.5
PTYPE 3 0; RELOCATE 1 5; DOT
PTYPE 3 1; RELOCATE 1 4; DOT
PTYPE 3 2; RELOCATE 1 3; DOT
PTYPE 3 3; RELOCATE 1 2; DOT
PTYPE 3 4; RELOCATE 1 1; DOT

PTYPE 4 0; RELOCATE 2 5; DOT
PTYPE 4 1; RELOCATE 2 4; DOT
PTYPE 4 2; RELOCATE 2 3; DOT
PTYPE 4 3; RELOCATE 2 2; DOT
PTYPE 4 4; RELOCATE 2 1; DOT

PTYPE 5 0; RELOCATE 3 5; DOT
PTYPE 5 1; RELOCATE 3 4; DOT
PTYPE 5 2; RELOCATE 3 3; DOT
PTYPE 5 3; RELOCATE 3 2; DOT
PTYPE 5 4; RELOCATE 3 1; DOT

PTYPE 6 0; RELOCATE 4 5; DOT
PTYPE 6 1; RELOCATE 4 4; DOT
PTYPE 6 2; RELOCATE 4 3; DOT
PTYPE 6 3; RELOCATE 4 2; DOT
PTYPE 6 4; RELOCATE 4 1; DOT

PTYPE 12 0; RELOCATE 5 5; DOT
PTYPE 12 1; RELOCATE 5 4; DOT
PTYPE 12 2; RELOCATE 5 3; DOT
PTYPE 12 3; RELOCATE 5 2; DOT
PTYPE 12 4; RELOCATE 5 1; DOT
```

Line Types

This example demonstrates the various line styles available using the `LTYPE` command and different line widths selected using `LWEIGHT`. See Figure 3.9 on page 53.

Figure A.7: Code for Generating Line Weights Example

```
LOCATION 0 1 0 1
LIMITS 0 1 0 1
VPAGE 0 1 0 1
LTYPE 0

EXPAND 2.25;  LWEIGHT 1.5
RELOCATE 0.25 0.9;  PUTLABEL 5 "\\fLTYPE\\f \\istyle"
EXPAND 2;  ANGLE 90;  LWEIGHT 1
RELOCATE 0.05 0.43;  PUTLABEL 5 \\iLine Type (style)

ANGLE 0
RELOCATE 0.1 0.76; PUTLABEL 5 0
RELOCATE 0.1 0.65; PUTLABEL 5 1
RELOCATE 0.1 0.54; PUTLABEL 5 2
RELOCATE 0.1 0.43; PUTLABEL 5 3
RELOCATE 0.1 0.32; PUTLABEL 5 4
RELOCATE 0.1 0.21; PUTLABEL 5 5
RELOCATE 0.1 0.1;  PUTLABEL 5 6

LWEIGHT 1
LTYPE 0; RELOCATE 0.135 0.76; DRAW 0.45 0.76
LTYPE 1; RELOCATE 0.135 0.65; DRAW 0.45 0.65
LTYPE 2; RELOCATE 0.135 0.54; DRAW 0.45 0.54
LTYPE 3; RELOCATE 0.135 0.43; DRAW 0.45 0.43
LTYPE 4; RELOCATE 0.135 0.32; DRAW 0.45 0.32
LTYPE 5; RELOCATE 0.135 0.21; DRAW 0.45 0.21
LTYPE 6; RELOCATE 0.135 0.1;  DRAW 0.45 0.1

LTYPE 0;  ANGLE 0;  LWEIGHT 1.5
RELOCATE 0.75 0.9;  PUTLABEL 5 "\\fLWEIGHT\\f \\iwidth"
EXPAND 2;  ANGLE 90;  LWEIGHT 1
RELOCATE 0.55 0.45;  PUTLABEL 5 \\iLine Weight (width)



ANGLE 0
RELOCATE 0.6 0.76; PUTLABEL 5 1
RELOCATE 0.6 0.65; PUTLABEL 5 2
RELOCATE 0.6 0.54; PUTLABEL 5 3
RELOCATE 0.6 0.43; PUTLABEL 5 4
RELOCATE 0.6 0.32; PUTLABEL 5 5
RELOCATE 0.6 0.21; PUTLABEL 5 6
RELOCATE 0.6 0.1;  PUTLABEL 5 7

LWEIGHT 1; RELOCATE 0.642 0.76; DRAW 0.95 0.76
LWEIGHT 2; RELOCATE 0.642 0.65; DRAW 0.95 0.65
LWEIGHT 3; RELOCATE 0.642 0.54; DRAW 0.95 0.54
LWEIGHT 4; RELOCATE 0.642 0.43; DRAW 0.95 0.43
LWEIGHT 5; RELOCATE 0.642 0.32; DRAW 0.95 0.32
LWEIGHT 6; RELOCATE 0.642 0.21; DRAW 0.95 0.21
LWEIGHT 7; RELOCATE 0.642 0.1;  DRAW 0.95 0.1
```

Coordinate Systems

Virtual Page

This example demonstrates virutal page coordinates (VPC) and the VPAGE command. See Figure 3.2 on page 42. This includes examples of defining and invoking macros.

Figure A.8: Code for Generating Virtual Page Example

```
define dvbox
    VMOVE 0 0
    VDRAW 1 0
    VDRAW 1 1
    VDRAW 0 1
    VDRAW 0 0
end

define vboxf
    ltype dotted
    dvbox
end

define dbox
    ltype dashed
    dvbox
end
dbox

lweight 1
VPAGE .1 .9 .1 .9
vboxf
dmove .5 .91
expand 1.26
putlabel 8 "\\f VPAGE .1 .9 .1 .9 \\f (\\iw.r.t. device)"

LOCATION .15 .9 .15 .8
expand 1.05
xlabel "\\f LOCATION .2 .9 .1 .7 \\f (\\iw.r.t. virtual page)"
limits 0 1 0 1
margin
BOX -1 -2


ltype solid

dmove 0.1 0
DDRAW 0.15 0.05
dDRAW 0.2 0.05
PUTLABEL 6 "\\i Device border"

MOVE 1.05 1.35
DRAW 1.0 1.25
DRAW 0.9 1.25
PUTLABEL 4 "\\iVirtual page border "

RELOCATE 0.1 1.024
DRAW 0.2 1.125
DRAW 0.3 1.125
putlabel 6 "\\i Viewport border"
```

Windowing

This example further demonstrates virtual page coordinates using the `WINDOW` command. See Figure 3.2 on page 42.

Figure A.9: Code for Generating Window Example

```
define vboxf
    ltype dotted
    VMOVE 0 0
    VDRAW 1 0
    VDRAW 1 1
    VDRAW 0 1
    VDRAW 0 0
end

define dbox
    ltype dashed
    VMOVE 0 0
    VDRAW 1 0
    VDRAW 1 1
    VDRAW 0 1
    VDRAW 0 0
end


dbox

dmove 0.9 0.9

location .25 .85 .175 .875

window 3 2 1
vboxf
box -1 -2
title "\\f WINDOW 3 2 1"
xlabel "LOCATION .25 .85 .175 .875"

window 4
vboxf
margin
box -1 -2
title "\\f WINDOW 4"

window 5
vboxf
limits .1 8 1 80
margin
box 1 2
title "\\f WINDOW 5"

window 2 2 2
ltype dotted
vboxf
location .15 .95 .175 .875
limits 5 55 -.75 .75
box 1 -2
title "\\f WINDOW 2 2 2"
xlabel "\\f LOCATION .15 .95 .175 .875"
```

APPENDIX B: Glossary

The following terms and acronyms are used in **igi**, additional generic to IRAF and STSDAS packages are defined in the glossary in the *STSDAS Users Guide.*

3-D tables: FITS tables which contain columns where each element of the column is an array.

axis: The edge of a plot having ticks and labels. The X axis typically goes horizontally across an image, the Y axis vertically. The directions along an unlimited line around which the image is symmetric.

command buffer: Memory of previously-executed commands.

error bar: Graphic representation of uncertainty in the data.

escape: To send a command from one logical level to another. In **igi**, commands can be escaped to the cl or to the host operating system.

family: Collection of commands that operate in a similar manner, but on data from different vectors (arrays).

global parameters: Attributes that affect the operation of commands.

histogram: In **igi**, a bar chart.

igi: Interactive Graphics Interpreter.

justify: To evenly space text between margins. In **igi**, it means to align any edge of a text block with the specified position.

label: Text applied to a graph, specifically tick and axis annotations.

macro: A set of **igi** commands that can be stored in memory and played back at some later point as if it were typed by the user.

metacode: Binary file containing device-independent graphics commands for generating a plot.

normalized device coordinates (NDC): The coordinates natural to the device, ranging in value from zero to one in both directions.

physical units: Real-world units of measurement, inches in **igi**.

pipe: To use the output from one task or program as the input to another task or program. The output of a task consisting of **igi** commands may be piped to the **igi** task.

polyline: A curve represented by points at particular coordinates, connected by straight line segments.

scaling: To change the represented size of an image.

script: Collection of commands stored to an ASCII file that can be played back as a macro.

spawn: To start a subprocess.

tick: A mark on a plot or graphic used to delineate change in data values.

tiling: Displaying multiple windows simultaneously, without overlapping edges.

vector: An array of data values.

viewport: The region of a page or display in which to draw a graph, coincident with the plot axes.

virtual page: A portion of the plotting area on the display or page, treated logically as a page itself.

virtual page coordinates (VPC): The normalized coordinate system whose origin is at the lower left corner of the virtual page.

world coordinates (WC): The coordinate system naturally applying to the data, pixels or wavelength, for example.

Index

S

T

U