STSDAS Version 1.3
Second Edition
October 1992

IGI Reference Manual

SPACE
TELESCOPE
SCIENCE
INSTITUTE

Science Software Branch
Science Computing and Research 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

Brad Whitmore	Acting Chief, Science Software Branch
Betty Stobie	STSDAS Programming Supervisor
Dick Shaw	STSDAS Project Scientist
Ray Williamson	System Administration and Distribution
Howard Bushouse	FOS Calibration and Analysis, Synthetic Photometry
Jonathan Eisenhamer	Graphics, Image Display, WCS Support
Phil Hodge	Table System, Fourier Analysis, FOC Calibration and Analysis
J.C. Hsu	HSP, FGS, and WF/PC Calibration and Analysis
Zolt Levay	Graphics, Image Display, Filmwriter, Documentation
Bernie Simon	Calibration Data Base, Table Editor, IRAF System Support
Nelson Zarate	FITS, IRAF System Support, Ports, Benchmarking
Cheng-Yue Zhang	WF/PC II Calibration and Analysis
Mark Stevens	Technical Writing
Fred Romelfanger	X-Windows-Based User Interface
Jinger Mo	Software Testing, Image Restoration

Written by Zoltan G. Levay
Edited October 1992

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

M9.27

Table of Contents

List of Figures

List of Tables

CHAPTER 1: Overview

The Interactive Graphics Interpreter (**igi**) is a task in STSDAS[1] for drawing high quality data plots within the environment used to analyze observations. It is closer in concept to programs such as the X-windows application `xvgr` than to structured drawing programs such as MacDraw or `xfig`.

Because it is an IRAF[2] application, it 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 any IRAF 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.

This manual supplements the on-line help text (available from the cl using the `help igi` command or from within **igi** using the `HELP` command). This chapter explains the concepts underlying **igi**. The next chapter describes how to use the task to create plots and presents some specific examples. The final chapter is an alphabetical reference to individual **igi** commands.

1. Space Telescope Science Data Analysis System, an external package running in IRAF and available from STScI.

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

Commands

Interaction with **igi** takes place through text commands. That is, keywords recognized by the interpreter represent some action. The structure of the command set is primarily verbs that perform a particular function. Thus, **igi** represents a "mini-language" distinct from the IRAF cl. Some commands have arguments to specify further qualification of the action. Arguments to the primary commands specify needed information such as a coordinates or data file names. Other commands set global parameters that modify the actions of the other commands—such as changing the marker style or line width.

The following list describes some of the 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. Also, cl syntax, e.g., filenames 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 always 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 uniquely distinguish 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 a # 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 input as a command *script*.
- ***Piping and redirection*** - Commands input to **igi** 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[3]. When the graphics

device is the user's 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 terminal used for interaction.

Typing `END` or `BYE` (or the end-of-file marker in a command file) terminates the interpreter. That is, 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**. Type `HELP` to get general information, `HELP` *`command`* for information about a specific command, or `APROPOS` *`keyword`* to list commands associated with the keyword.

Command Classes

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

- ***Execution*** - Alter subsequent **igi** command operation or show the current state. These commands are generally not saved in the command buffer.
- ***Data*** - Input or modify data in the **igi** data buffers. The `SHOW` command displays the current state of the data buffers.
- ***Attribute*** - Set or modify 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*** - Modify or set the transformation between various coordinate systems[4] 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. IRAF-specific, device-independent binary file containing plotting instructions.

4. Coordinate systems are described in more detail in the section "Coordinates and Scaling" on page 11.

- ***Plotting*** - Draw 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 3.1 on page 49 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 3: “Command Reference”.

The commands listed in Table 1.1 deal with commands and the command buffer.

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

Table 1.1: Commands Dealing with Commands.

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 and semantics, 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[5]. Command families are identified in this text with square brackets, e.g., `[XYEPLS]COLUMN`. Chapter 3: "Command Reference" describes each command and family separately.

Redirecting Command Input

igi commands are read from the standard IRAF input stream (`STDIN`). Therefore, they may be typed interactively, piped from another task, or redirected from a file directly on the command line when **igi** is started. 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 still be directed to the terminal or redirected to a plotter or a file.

Escaping to the cl

The user may *escape* commands to the enclosing environment. That is, you may type IRAF cl commands and host command from the **igi** prompt. Escape commands to the cl by preceding the command with the `!`

5. Data buffers are explained on page 9.

character. Since the cl recognizes this character as an escape to the host system, commands may by 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
```

Macros

A *macro* is text that may be represented by a single command. Macro text is replaced in line, just as if the text had been typed in place of the defined macro command. The user may define a macro using the `DEFINE` command. The argument of the `DEFINE` command is the name of the macro. The text of the macro is then entered as command input, but the commands are not executed. Arguments are defined in the body of the macro and replaced on invocation by the positional macro arguments. When 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.

```
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
```

Figure 1.1: Creating a Macro Using `DEFINE`.

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 may invoke any **igi** command or other defined macros, but 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, will be stored as part of the macro text, and will be expanded into the command stream when the macro is invoked.

Figure 1.1 lists commands that define and invoke a macro named "simple" to scale and draw a plot.

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*** 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 3: "Command Reference" 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. Note that 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
- IRAF images — including image section

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 input data buffers subsequently may be modified arbitrarily using expression evaluation syntax (see “Expression Evaluation” on page 11).

Any data type (integer, double precision, etc.) may be in the external file but when read are converted to single-precision floating point value. 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 and may be used as a comment.

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. Data in the buffers are interpreted as: X and Y plot coordinates, errors, point style codes, upper or lower limit flags, and a scratch buffer.

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 text data to be read are specified by the column number and table data 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 1.2.

Format	*Type*	*Extension*
“Old” IRAF	OIF	`.imh`
STSDAS (GEIS) including groups	STF	`.hhh`
ROSAT/pros x-ray events	QPOE	`.qp`

Table 1.2: IRAF Image Types Recognized by igi.

Image data are handled slightly differently from text and binary tables. The command `[XYEPLS]SECTION` opens an image file and reads data into one of the **igi** vectors. 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.

Grayscale Images

A separate buffer accepts true two-dimensional, grayscale images, also known as rasters, pixmaps, etc. 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.

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 such as addition, subtraction, multiplication, trigonometric functions, etc. Note that `indef` values operated on in this way may cause **igi** to crash since they are not recognized as `indef` by the evaluator but treated as normal data values.

Variables representing the internal buffer arrays are recognized in expressions and referred to by a single character. In addition, you may refer to the element number by `r` and the number of elements in the vector by `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)
igi> YLOGARITHM
```

Coordinates and Scaling

Coordinates for most commands (pen position, cursor readback, etc.) are specified in *world coordinates*, or WC, also known as *user coordinates*. A *virtual page* may be specified as a portion of the physical device drawing area by setting the edges in normalized device coordinates (NDC) using the `VPAGE` command, or in physical units (inches) using the `PHYSICAL` command. The *viewport*, i.e., the axes of a plot, may fall anywhere within this page and is specified as a fraction of the virtual page in the range 0:1 using the `LOCATION` command on which to plot is specified in *normalized device coordinates* or NDC, in the range 0:1 in both the X and Y direction.

Coordinate Systems	*Command*
WC ↔ VPC	LIMITS
VPC ↔ NDC	VPAGE
NDC ↔ Inches	PHYSICAL

Table 1.3: Commands Specifying a Coordinate System Transformation.

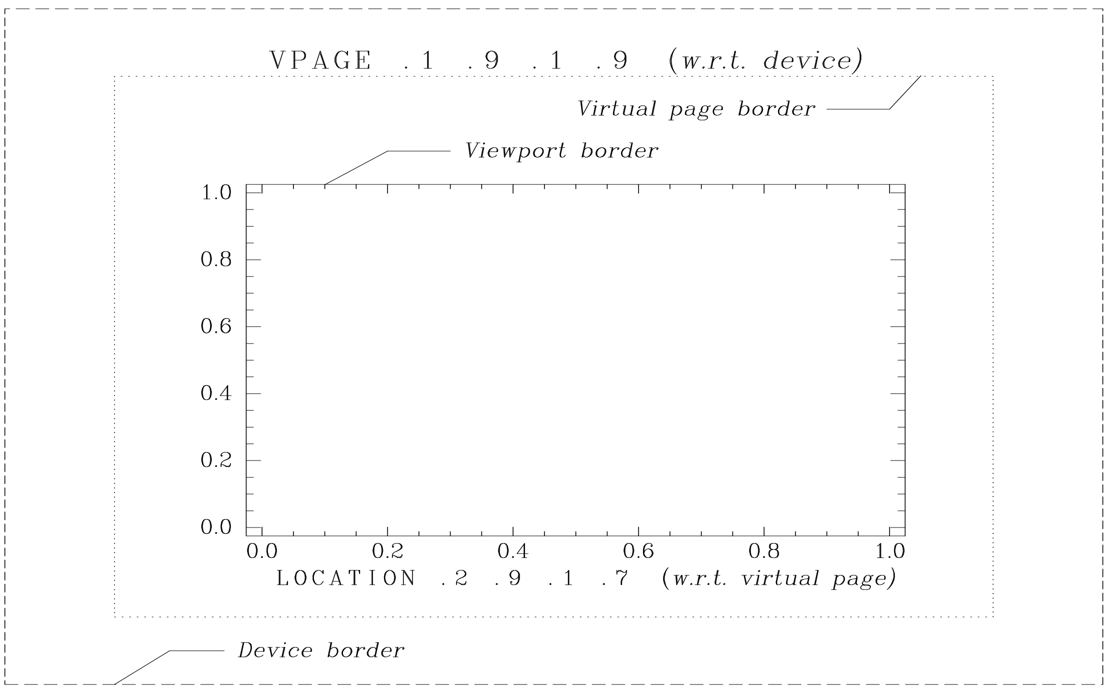


Figure 1.2: Coordinate System.

An alternate method of specifying a virtual page is with the `WINDOW` command. This permits the display space to be *tiled* into a maximum of 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. A given invocation of the `WINDOW` command selects one of the panes as the current virtual page. Another window in the current tiling may be selected by specifying only the pane number without respecifying the tiling. It is possible, of course, to reset the tiling as well. In this case, you may get overlapping windows. Figure 1.3 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.

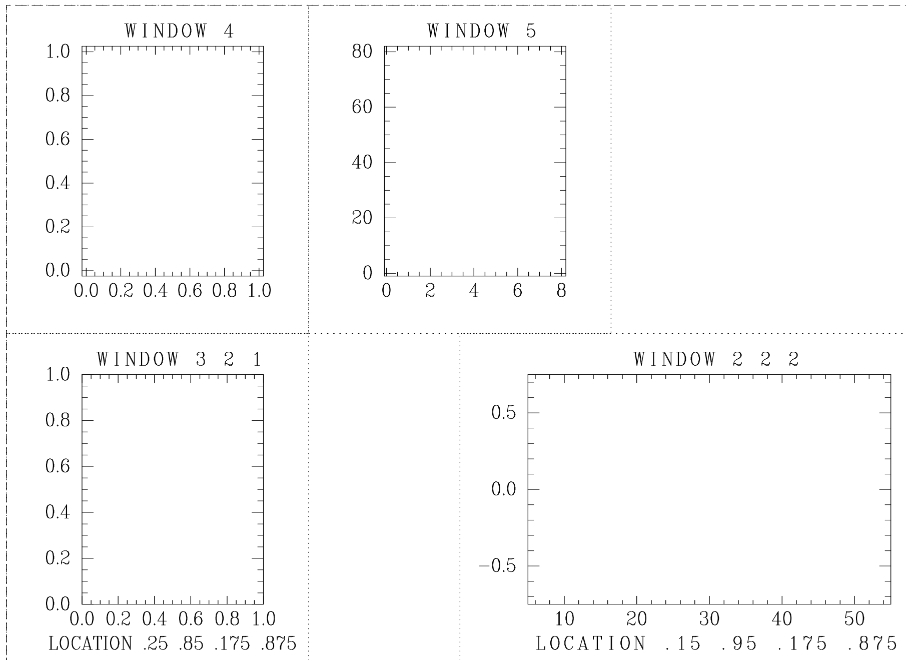


Figure 1.3: Virtual Pages and Windows.

Figure 1.5 shows an example of the results from the following commands to draw a grid in inches.

`LOCATION 0 1 0 1`	*Viewport on full vpage*
`PHYSICAL 0 3 0 2`	*Make the grid 3 x 2 inches*
`LIMITS 0 3 0 2`	*Scale matches vpage in inches*
`TICKSIZE .125 1 .125 1`	*Minor 1/8, major 1 inch tick spacing*
`BOX 0 0`	*Draw the axes with no tick labels*
`LTYPE DASHED; GRID`	*Draw the grid with dashed lines*
`TICKSIZE .0625 .25 .0625 .25`	*Minor 1/16, major 1/4 inch*
`BOX 0 0; LTYPE DOTTED; GRID`	*Draw the finer grid with dotted lines*

Example 1.4: Example of Commands for Absolute Physical Scaling.

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 thus defines the transformation between the user and device coordinates. The `LIMITS` command may be used to explicitly specify the coordinate scale, or to autoscale based on the current X and Y plot vector data values. In the latter case, 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.

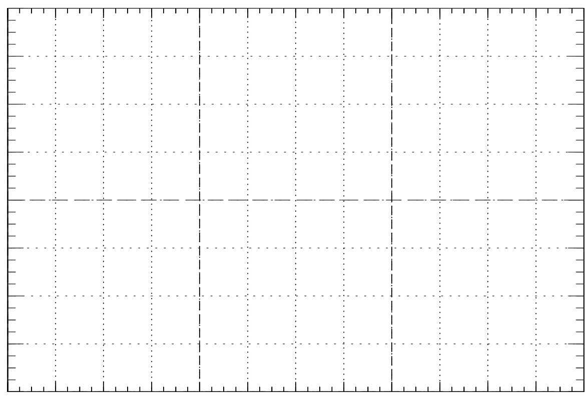


Figure 1.5: Absolute Scaling with `PHYSICAL`.

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 requiring a plot position (move and draw) have versions for each of 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`

Table 1.4: Pen Positioning Commands in the Coordinate Systems.

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.

Implementation Highlights

igi is implemented in the IRAF Subset PreProcessor (SPP) language[6] as a task in the `stplot` package of STSDAS. It incorporates a custom command line interpreter (CLI) 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). A user can operate on internal data vectors with arbitrary expressions, implemented using the `evexpr` SPP VOS

6. Syntactically, a hybrid of Fortran and C, preprocessed into Fortran.

procedure. Fonts are implemented using a custom host-independent subset of the Hershey fonts using the machine-independent integer (MII) facility of SPP.

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. Commands may be *spawned* to the cl and the host level using the spawned command facility of the IRAF SPP Virtual Operating System[7] (VOS).

Summary of igi Features:

- Custom command line interpreter (CLI) to interpret commands
- 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
- Additional capabilities depend on kernel and device implementation. Color, grayscale, filled areas work with the kernel psikern and PostScript devices.

igi's Future

While **igi** is useful in its current implementation, several enhancements are planned to make it a more complete facility. A useful and natural extension is to render multi-dimensional input data by drawing contour maps or wireframe surface plots. Changes to the command interpreter would improve the robustness and provide more programming language capabilities. Many users would find it easier to interact with **igi** if the user interface were more visual. Finally, most of the capabilities of **igi** should be available to any IRAF graphics application, not only from the cl. That is,

7. The library of procedures interfacing a task to IRAF.

we will provide a library of procedures that provide the same capabilities as most of the interactive plotting commands of **igi**.

- Multi-dimensional data rendering
  - Contour maps
  - Surface plots
  - 3-D models
- Integrate with image display
- Enhance command interpreter and macro processor (rewrite using `xyacc`)
- Improve the user interface
- Provide applications library

CHAPTER 2: Using igi

This chapter explains most features of **igi**, describes commands, and provides several examples. Because **igi** offers a separate environment and command syntax from the cl, let's take a moment to explain how to get out and get help on line.

Getting Started

How to Get In

To start **igi**, first start IRAF using the `cl` command at your system prompt, load the **stsdas** and **stplot** packages, then type `igi`.

How to Get Out

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

HELP!

There is on-line help available in the cl and within **igi**. At a cl prompt, type `help igi` for complete help (a subset of this document). 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:

```
cl> help igi section = data
```

Within **igi** the `HELP` command displays the same help text. Typing `HELP` pages the full **igi** help text. To get help on a specific command, simply type the name of the command after `help`. For example:

```
igi> help data
```

To see a list of available commands, type `MENU` or `HELP MENU`. To find commands related to a specific topic, use the `APROPOS` command, `APROPOS axis` for example.

Using igi

This chapter is intended to explain as many features as possible, describing the commands to use them. In addition, it presents many examples of using **igi**. Refer to Chapter 1 for a more abstract presentation of concepts and Chapter 3 for details of each command. Since **igi** offers a separate environment and command syntax from the cl let's take a moment to explain how to get out and get help on line.

Invoking the task

The command line examples in Table 2.1 illustrate several ways **igi** might be used

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 doplot task
`st> igi init="data mydat" <mycmd.igi`	Start with an initial command

Table 2.1: Sample igi Task Command Lines.

Note that the `append` parameter allows one to overlay additional graphics using **igi** on a plot drawn by any IRAF graphics task. The `initcmd` parameter enables a user to pass a command string for **igi** to execute on start-up; this allows the same command to be run with different initial states.

Task Parameters

All of the task parameters are *hidden* which means that their values need not be given explicitly on the command line if they have default values. In addition, if they are to be typed on a command line, they must be specified with the parameter name and value.

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

Table 2.2: igi Task Parameters.

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 may be executed by separating them with colons, 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.

The graphics device parameter, `device`, is consistent with other IRAF graphics tasks. It's default value, `stdgraph`, uses the device set in the IRAF cl environment variable named `stdgraph`. For example, if `stdgraph = xterm`, and the user is 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 file. This parameter is required if `device = "file"` but otherwise ignored. Alternately, you can redirect graphics output by using `>G file` on the command line.

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.

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.

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.

Drawing

The usual order of events when drawing plots is:

1. Read data
2. Scale
3. Draw axes
4. Draw the curve or points
5. Annotate Text

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

1. Graphics Kernel Interface; see the gio help for more explanation.

Reading Data

The commands used for dealing with data are listed in Table 2.3. 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.

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
XYLOGARITHM	Compute the log of a data buffer
[XYZEPLS]SECTION	Read an image section into a vector

Table 2.3: Commands that Input or Modify Data.

Scaling

The plot scale is set by the LIMITS command. Explicitly specifying four arguments sets the world coordinates of the edges of the viewport. 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.

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

Table 2.4: Commands to Set or Modify Scaling.

Data Curves

Data points may be plotted individually—one by one—using move and draw or the `DOT` command to plot a centered symbol at the current pen position. Or a curve may be plotted from the data in the Y and optionally the X buffer using one command. Data in the plot buffers may be plotted in several styles. A complex plot may be built using several different curve styles. Data points may be connected by straight lines or with *stepped* horizontal segments connected by vertical lines. In addition, a bar graph[2] 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.

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

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 points through data
`ULLIM`	Draw upper or lower limit symbols

Table 2.5: Commands that Draw Curves or Data Points.

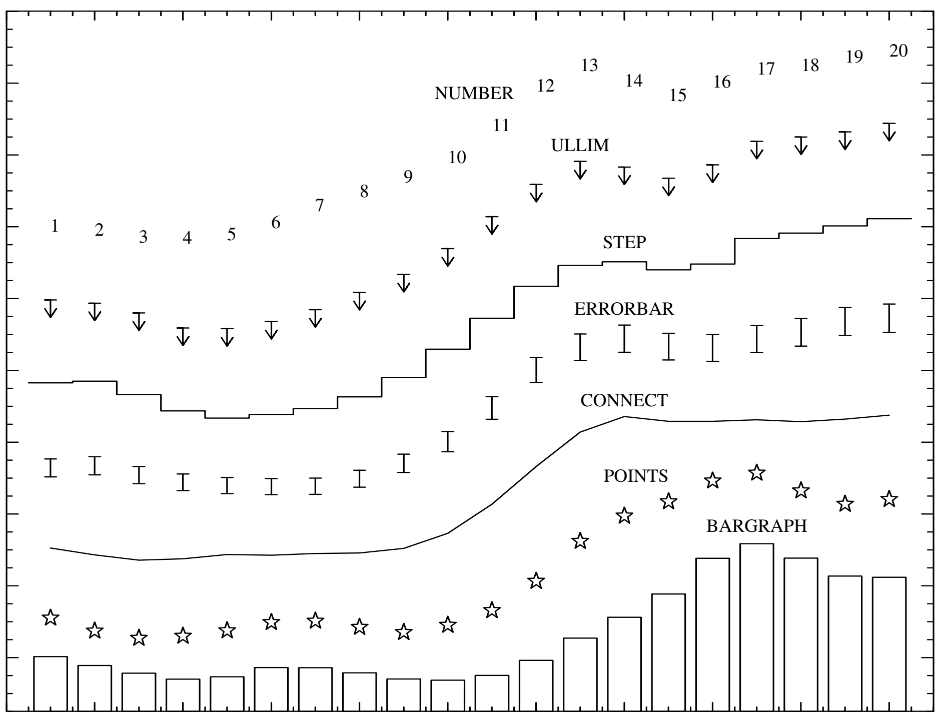


Figure 2.1: Types of Plotted Curves and the Commands that Produce Them.

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 2.6 lists the move and draw plotting primitive commands for each coordinate system. Note that there are two equivalent alias for the move command.

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

Table 2.6: Coordinate Systems.

For example the commands in Figure 2.1 will draw a border around the edge of the device (window or page): see “A Basic Plot” on page 39 for an example of annotations drawn using primitives.

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

Example 2.2: Draw a Box in NDC.

The example in Figure 2.3 sets a virtual page to the upper left of the screen and draws the border around the virtual page.

Figure 2.3 also demonstrates simple moves and draws in the various coordinate systems.

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

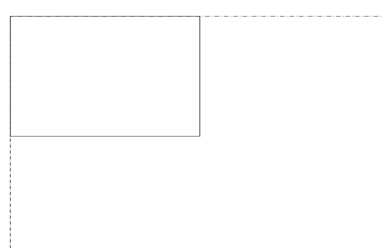


Figure 2.3: Commands and Plot of a Border in VPC.

Attributes of plotting primitives

Several commands set *global parameters* that control the results of other commands.

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

Table 2.7: Commands for Setting Global Attributes.

- `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`[3] file).
- LTYPE selects one of the available line styles. This is 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 3.8: on page 77 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: on page 78 for examples of changing the line weight.

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

- `ANGLE` is specified in degrees relative to the +X (horizontal) axis See Figure 2.5 for examples of the `PTYPE` command demonstrating the different point marker styles. Line weights and types are shown in Figure 2.4.
- `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 3.2: on page 59 for an example of available colors.
- `FILLPAT` is also device- and kernel-dependent. The argument selects one of the available fill patterns, which apply to bar graphs and symbols plotted with `DOT` and `POINTS`.

Figure 2.4: Line Styles and Widths.

Axes

Several commands let you plot data axes and control their appearance. These commands are listed in Table 2.8 on page 29. 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.

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

Table 2.8: Axis and Labeling Commands.

The four plots in Figure 2.5 show examples of various styles of axes.

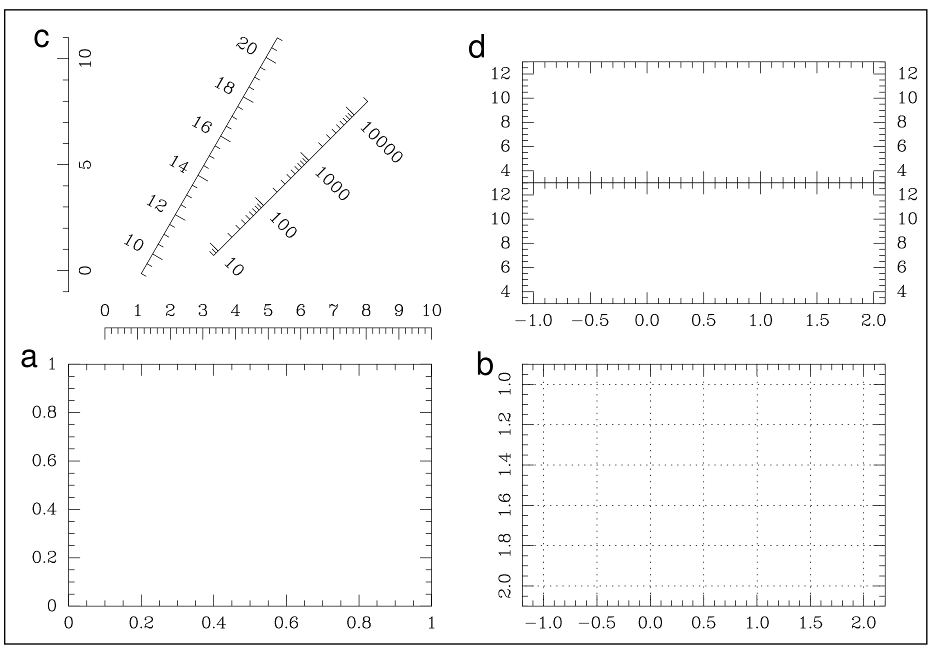


Figure 2.5: Examples of Axes

- The bottom left plot (a) is the result of the BOX command with no arguments, i.e., using the default parameters. Note the orientation and format of the tick labels.
- The bottom right plot (b) results from the commands in Figure 2.5. 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.

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

Example 2.6: Attributes of BOX Command.

- The top left plot (c) is the result of the AXIS commands in Figure 2.5 demonstrating different options.

```
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
```

Example 2.7: Individual AXIS Commands.

- The upper right plot (d) is a full set of axes drawn individually using AXIS since BOX does not draw tick labels on the right axis.

BOX*: Full plot borders with axes*

BOX does not label any axis but does label the major ticks along the bottom and left axes. The XLABEL and YLABEL commands allow an arbitrary string to be drawn 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 to draw an arbitrary string. There are two string buffers maintained for the X and Y labels. These are filled by the XYCOLUMN and XYSECTION commands with the filenames used as input data. If the argument to `XYLABEL` is a null string, then **igi** will use the label strings. Otherwise, it will use the string 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` 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 2.8 shows the difference between positive and negative arguments to `BOX` resulting in having leading and trailing zeros plotted or suppressed.

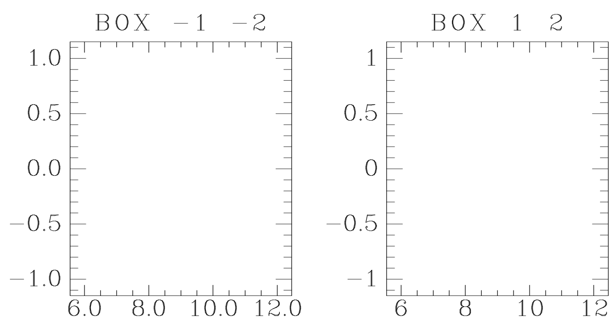


Figure 2.8: Leading and Trailing Zeros in Axis Labels with `BOX`.

`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, it's 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 3.6 on page 57 for the interpretation of the label argument and Table 3.4 for an interpretation of the clock argument.

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 2.9: Labeling Parameters in the `AXIS` Command.

The first two arguments to `AXIS` 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 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, and the last specifies on which side of the axis the ticks and labels will appear. Note the negative value for the third argument in the last command, specifying logarithmic tick labeling. The `ANGLE` command must be used separately to specify the angle of the axis.

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 arbitrary text. Some attributes of the text are specified using commands to set global parameters. These include `ANGLE` to specify how to rotate the text, `EXPAND` to specify the size, and `LWEIGHT` to set the width of lines (boldness) of text. These parameters are used by other drawing commands as well. Other attributes of the text are specified by embedded escape sequences, single characters preceded by the \ character. For example, a Greek character may be specified by the `\g` prefix. A single backslash character changes the attribute for the single following character. A second \ character changes the attribute for the remainder of the string or until it is changed again by another escape. Any font may be italicized using the `\\i` sequence, for example.

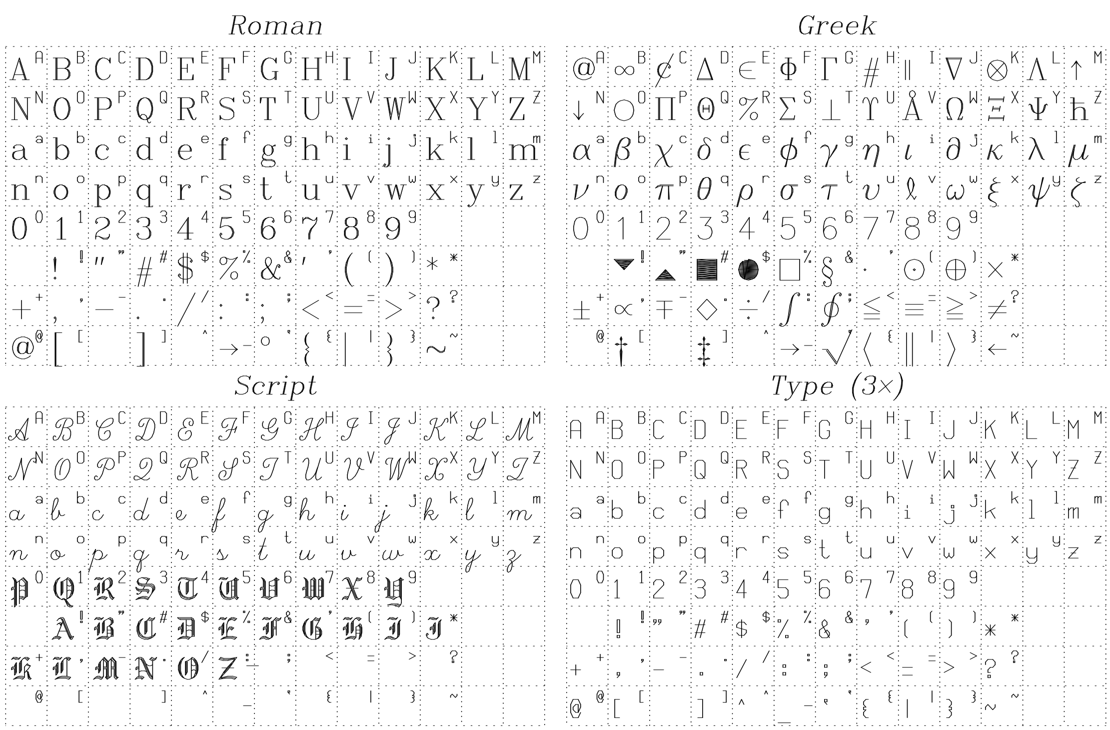


Figure 2.9: igi Fonts Available with Embedded Text Escapes.

Table 2.10 list the commands that produce or modify text drawn by **igi**. The first set draw text and the second set modify its appearance. Table 2.11 lists the allowed values of the arguments to the `JUSTIFY` and `PUTLABEL` commands and the resulting justification of the text. Figure 2.10 shows examples of various text attributes such as the justification and the text escapes.

Command	***Operation***
	Text Drawing Commands
`LABEL`	Draw text with current justification
`PUTLABEL`	Draw text with arbitrary 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
`JUSTIFY`	Justification
`LWEIGHT`	Boldness (line width)

Table 2.10: Text Commands.

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

Table 2.11: Numeric Arguments to `JUSTIFY` and `PUTLABEL`.

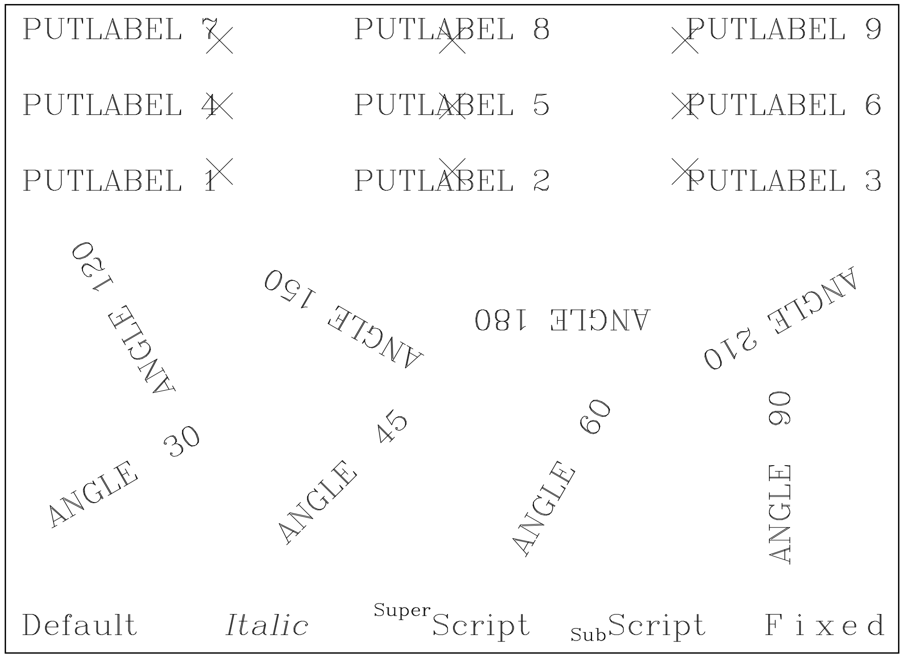


Figure 2.10: Text Drawing Commands and Attributes.

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.

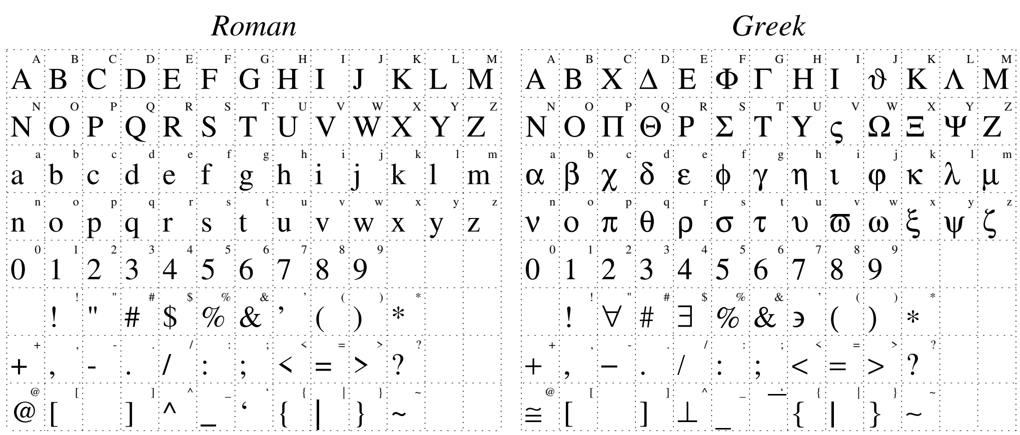


Figure 2.11: Hardware (gio) Fonts.

Figure 2.11 shows all of the characters available in any text using embedded escapes with **igi** 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`.

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
`\f`		Toggle proportional spacing
`\u`		Superscript
`\d`		Subscript
`\b`		Backspace
`\e`		End string

Table 2.12: Text Escape Codes.

Image Display

Some graphics devices are able to display grayscale images. Igi has basic image rendering (display) capabilities. 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:

- Read the data (ZSECTION)
- Map the display brightness (ZRANGE)
- Scale the viewport (LIMITS)
- Display the image (PIXMAP)

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 on the current graphics device.

Command	***Result***
ZSECTION	Read 2-D grayscale data into Z buffer
ZRANGE	Scale grayscale to display brightness
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

Table 2.13: Image Display Commands

Figure 2.12 shows some of the capabilities of the image display in igi. Graph A shows the default action, to fill the viewport with a positive representation of the data. Graph B demonstrates the FITPIX command to preserve the aspect ratio of the image. In addition, ZRANGE specified a bright display values mapped to a pixel value smaller the faint value, resulting in a negative gray map. Graph C shows the result of an image section consisting of a single line of the same image displayed in graphs A and B. The displayed image is widened to fill the viewport. Graph D is a line plot of the same image section. The horizontal line plotted in Graph B is the same line represented in Graphs C and D.

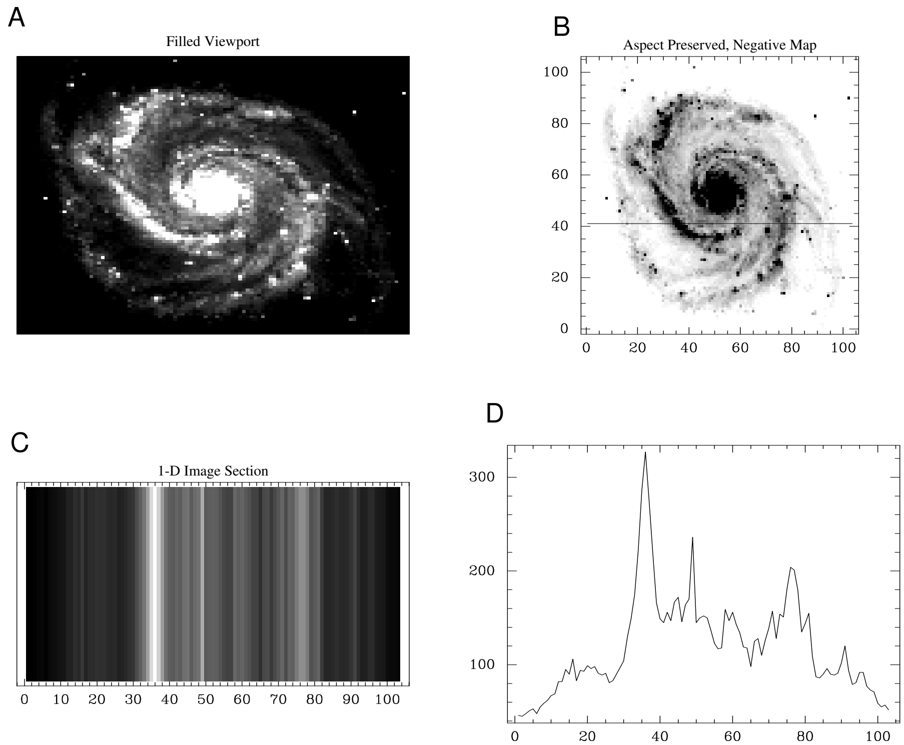


Figure 2.12: Examples of Displaying Images

Interactive Graphics Cursor

All of the existing capabilities of the IRAF gio[4] graphics cursor is available from **igi** for interacting with a plot. These features include zooming, panning and 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[5] 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

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

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

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.

Examples

This section presents various fairly simple examples of the kinds of things that can be done with **igi** and a few different ways it may be used. Where possible, the results of an example are also shown in a figure. In some cases, it is not possible to show the data used, when reading an IRAF image, for example.

A Basic Plot

The following **igi** commands produce the relatively simple plot shown in Figure 2.13. The data files are not shown here and some repetitive commands are omitted. In producing the example, some cursor interaction was used to position the identifications. This demonstrates several common commands.

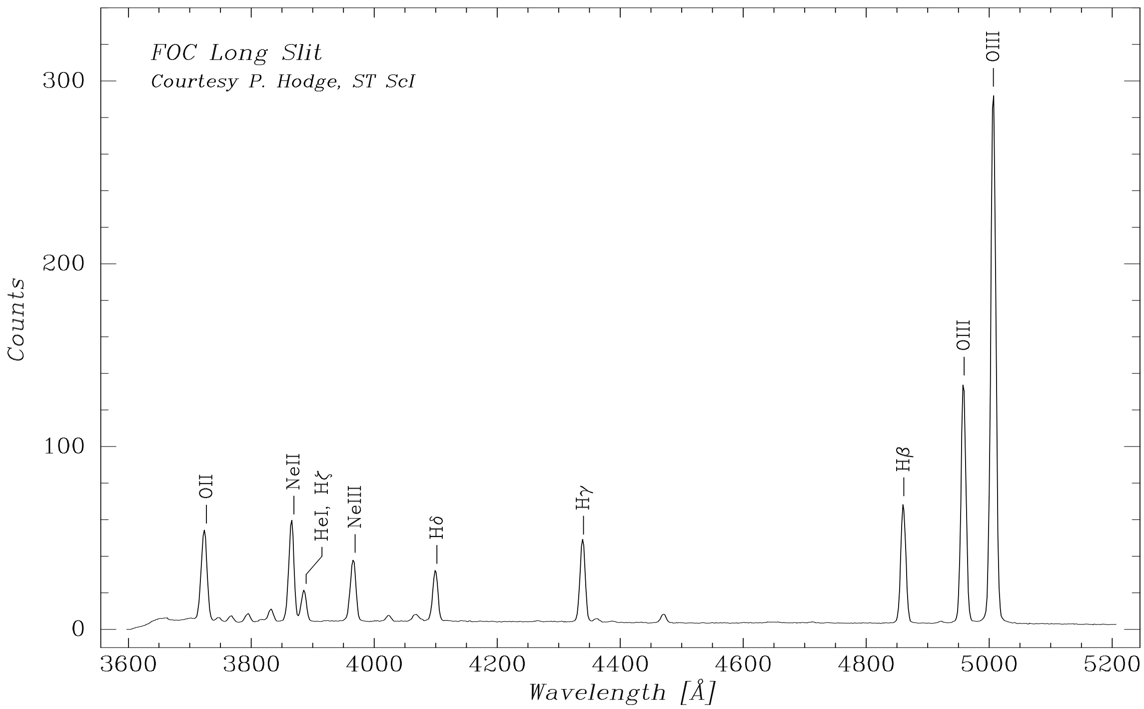


Figure 2.13: Basic igi Example.

```
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 .5 .96
PUTLAB 8 "\\iNGC 6543"
VMOVE .95 .96;  EXPAND 0.6
PUTLAB 7 "\\iCourtesy P. Hodge"
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
```

Additional commands omitted

Example 2.14: Source for Sample Plot.

Simple Plot

The following are examples of **igi** interactions. That is, the commands typed interactively after starting up the **igi** task. The prompt (`igi>`) is written by **igi** and the remainder of the line is what would be typed by the user. These are all fairly simple examples that demonstrate some of the basic means of drawing plots. Some of the more common kinds of things are demonstrated, with some of the more subtle tricks pointed out.

Read data from a text file, scale the plot, draw the axes, and draw the curve. Note the multiple commands on a line, separated by semicolons. Spaces after the semicolon is optional.

```
igi> data igi.dat
igi> xcolumn 1;ycolumn 2
igi> limits
igi> box;connect
```

Example 2.15: Reading Data into igi.

Log Scaling

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

Example 2.16: Log Scaling.

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

Example 2.17: Change to a Log-log Scale.

Binary Table Data

Read data from a table, scale the plot, draw the axes, and draw the curve as the Y elements versus row number:

```
igi> data table;  ycolumn ycol
```

Macro

Define a macro named `simple` to scale and draw a plot and invoke the 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
```

Example 2.18: Example of Defining a Macro.

Non-trivial Text

Print a vertical label (in italics with a special character) centered at the current pen position twice the default character size.

```
igi> expand 2; angle 90; justify 5
igi> label \\iWavelength (\gVngstroms)
```

Error Bars

Draw a curve and superimpose symmetrical error bars in the 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.

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

Example 2.19: Drawing Error Bars.

Windows

Plot two curves in separate windows (different axis grids), the data as read in one, and the log of the data in another:

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

Example 2.20: Plots in Windows.

Image Sections

Assuming the image named `spectrum` contains two lines, one having wavelength values and the other intensities, the following commands would plot the spectrum as intensities versus wavelength:

```
igi> xsection specim[*,1]
igi> ysection specim[*,2]
igi> limits
igi> box
igi> connect
```

Example 2.21: Reading and Plotting Image Sections.

Axis Labels

Label the above plot with the input file names:

```
igi> xlabel
igi> ylabel
```

World Coordinates

Use coordinate information in an input image to fill in the X values:

```
igi> imgwcs
igi> ysection spect
```

Complete Figure

The following is a more complete example of an **igi** command file. It includes some comments describing what's going on. Note that the data are totally fabricated random numbers and bear no intentional resemblance to any real observations or theory. The results of this example are shown in Figure 2.23.

```
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
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"
```

Example 2.22: Source for Sample Plot.

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
```

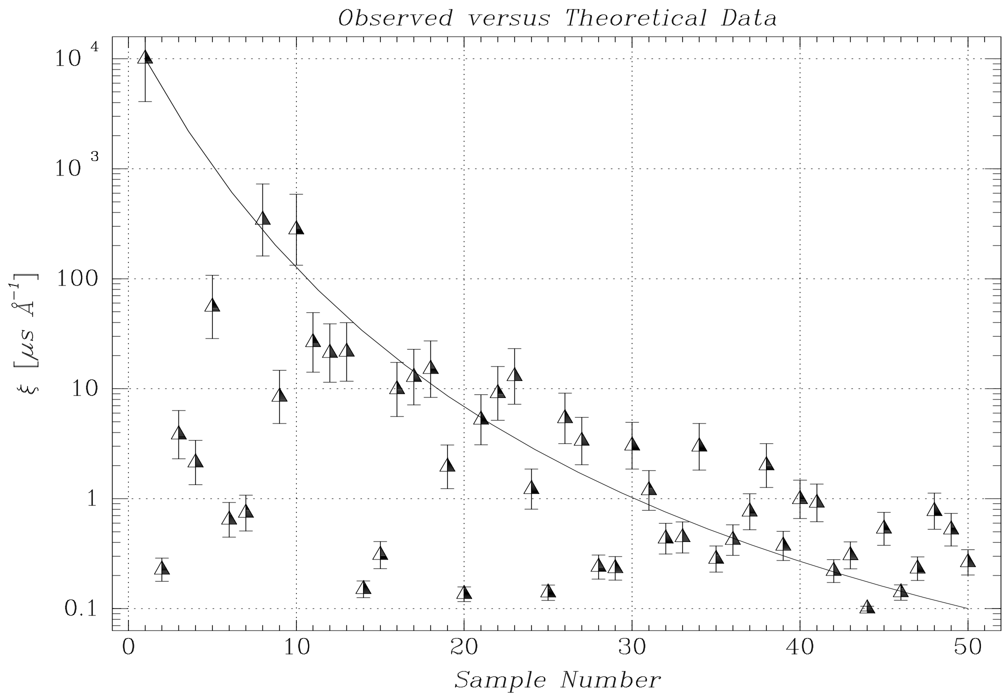


Figure 2.23: Sample igi Plot.

Stacking Curves

The **igi** script below shows one way to stack multiple curves, each with its own set of axes. The result is in Figure 2.25. It's a bit tricky to get the axes to abut *and* draw tick labels on the bottom X axis only. An alternate way of stacking plots is to use `WINDOW` specifying one pane horizontally and more than one vertically, `WINDOW 1 6 1` for example. In the method shown here, one must 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.

```
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
```

Example 2.24: Code for Stacked Curves Example.

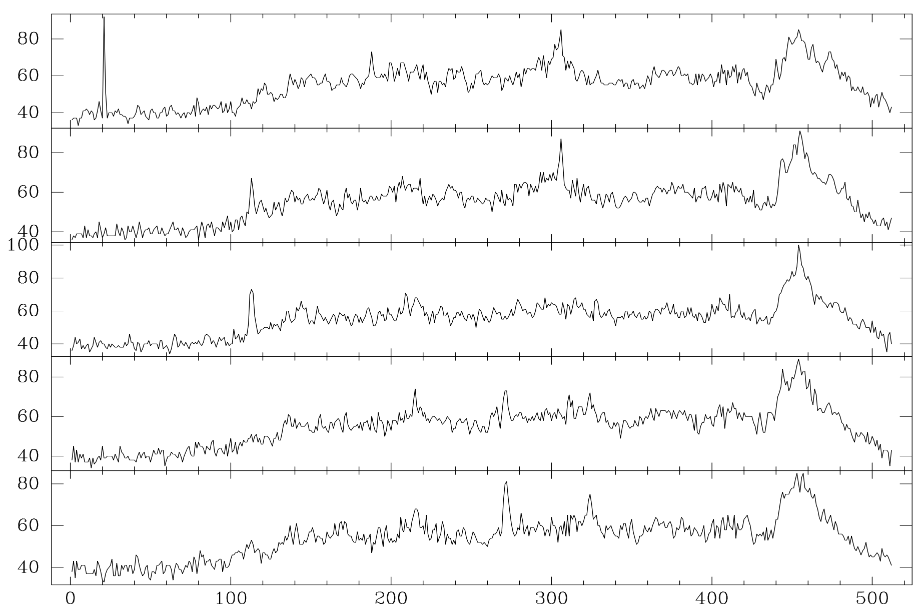


Figure 2.25: Multiple Curves Plotted in Separate Frames.

CHAPTER 3:

Command Reference

This chapter fully documents each **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 provides the **igi** commands used to draw the various figures demonstrating concepts in this document.

Table 3.1 lists each command, 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.

Command	*Page*	*Class*[a]	*Saved*	*Undo*	*Description*
!	53	E[b]			Escape a command to the cl
?	54	E			Page the help summary menu
^	54	E			Re-execute a previous command
ANGLE	38	A[c]	•		Set the marker and text rotation angle
APROPOS	38	E			List commands associated with a keyword
AXIS	38	P[d]	•	•	Draw and label an arbitrary axis
BARGRAPH	73	P	•	•	Draw a bar graph
BOX	57	P	•	•	Draw and label the axes
BYE	58	E			Terminate igi (alias for *END)*
COLOR		P	•		Set the drawing color
CONNECT	60	P	•	•	Draw a (polyline) curve connecting the data coordinates
CURSES	60	E			Start interactive cursor mode

Table 3.1: Summary of igi Commands.

Command	*Page*	*Class[a]*	*Saved*	*Undo*	*Description*
DATA	61	D[e]	•		Specify the input data file
DDRAW	61	P	•	•	Pen down move (draw) in NDC
DLIST	62	D			Print the data values
DMOVE	62	P	•		Pen up move in NDC (alias for *DRELOCATE*)
DOT	62	P	•	•	Draw a single marker
DRAW	63	P	•	•	Pen down move (draw) in WC
DRELOCATE	63	P	•		Pen up move in NDC
ECOLUMN	63	D	•		Read errors from a column of the data file
EDITCMD	64	E			Edit the command buffer or macro text
EEVALUATE	64	D	•		Operate on error values
ELLIPSE	64	P	•	•	Draw an ellipse
END	65	E			Exit igi or terminate macro define (end the current mode)
ERASE	65	P	•		Erase the screen
ERRORBAR	66	P	•	•	Draw error bars
ETYPE	66	A	•		Change the error bar style
EXPAND	68	A	•		Set the marker and text size
FILLPAT		A			Select a filled-area pattern
FITPIX		S			Set a viewport to match an image's aspect
FMTICK	71	A	•		Specify the axis tick label format
FONTSET		A			Select igi or gio fonts
GRID	72	P	•	•	Draw lines between major axis ticks
HELP	73	E			Print help
HISTOGRAM	73	P	•	•	Draw a histogram plot (bar graph)
ID	73	P	•	•	Write identification label
IMGWCS	73	S[f]	•		Toggle using image WCS for X values
INPUT	74	E	•		Execute commands from a file
JUSTIFY	74	A	•		Set text justification
LABEL	74	P	•	•	Draw text at current position
LCOLUMN	75	D	•		Read limits data from a column of the data file
LIMITS	75	S	•		Set the plot scale
LINES	76	D	•		Specify the range of input rows to read
LIST	76	E			List the command buffer or macro text
LOCATION	76	S	•		Set the viewport on the virtual page

Table 3.1: Summary of igi Commands.

Command	*Page*	*Class*[a]	*Saved*	*Undo*	*Description*
LTYPE	77	A	•		Set the line style (hardware dependent)
LWEIGHT	78	A	•		Set the line width (hardware dependent)
MACROS	79	E	•		List defined macros
MARGIN	79	S	•		Rescale to include a border between curve and axes
MENU	79	E			Page the help summary
MINMAX	80	P			Print the data range
MOVE	80	P	•		Pen up move (alias for *RELOCATE*)
NOTATION	80	A	•		Set the format for axis labels
NUMBER		P			Draw numerals representing index of each coordinate
PAGECMD	82	E			Page the command buffer or macro text
PCOLUMN	82	D	•		Read point specifications from the input data file
PEVALUATE	82	D	•		Operate on point code values
PIXMAP		P	•		Draw (render) an image (pixmap)
PHYSICAL	83	S	•		Set the virtual page in physical device coordinates
PLAYBACK	83	E			Execute the commands in the command buffer
POINTS	84	P	•	•	Draw a polymarker at input data coordinates
PTYPE	84	A	•		Specify the point marker type
PUTLABEL	85	P	•	•	Draw a label with a specified justification
READ	85	D	•		Read commands into the command buffer without execution
RELOCATE	86	P	•		Pen up move in WC
RESET	86	A	•		Return settable plot and data parameters to defaults
SAOCMAP		D			Specify a gray or color map for display
SCOLUMN	87	D	•		Read scratch data from a column of the data file
SECTION	88	D	•		Read an image section for a plot buffer
SHOW	65	E			Page the plot limits and attributes
STEP	66	P	•	•	Draw a stepped curve (histogram without vertical lines)
TICKSIZE	66	A	•		Set the axis tick spacing
TITLE	90	P	•	•	Specify the plot title
ULLIM	91	P	•	•	Draw upper or lower limit symbols
UNDO	91	E			Selectively erase the last plot command
VDRAW	91	P	•	•	Pen down move (draw) in VPC
VMOVE	92	P	•		Pen up move in VPC (alias for *VRELOCATE*)
VPAGE	92	S	•		Set the virtual page in normalized coordinates

Table 3.1: Summary of igi Commands.

Command	Page	Class[a]	Saved	Undo	Description
VRELOCATE	92	P	•		Pen up move in VPC
WINDOW	93	S	•		Divide screen into panes
WRITE	93	E			Write the command buffer to a file
XCOLUMN	93	D	•		Read X coordinates from a column of the data file
XEVALUATE	96	D	•		Operate on X data values
XFLIP	94	S	•		Flip X axis
XLABEL	94	P	•	•	Specify the X axis label
XLOGARITHM	95	D	•		Take common log of X data
XSECTION	98	D	•		Read X coordinates from an image section
YCOLUMN	96	D	•		Read Y coordinates from a column of the data file
YEVALUATE	96	D	•		Operate on Y data values
YFLIP	97	S	•		Flip Y axis
YLABEL	97	P	•	•	Specify the Y axis label
YLOGARITHM	97	D	•		Take common log of Y data
YSECTION	98	D	•		Read Y coordinates from an image section
ZRANGE		S			Map image values display brightness
ZSECTION		D			Read Z values from an image section

Table 3.1: Summary of igi Commands.

a. See “Command Classes” on page 3.
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 are listed alphabetically here without regard to similarity of function. In some cases, a family of commands (see “Command Families” on page 5) is described in addition to each individual in the family. 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 as such 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.

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

Table 3.2: Escaping Host Commands.

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 3.2 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.

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 3.1 shows the result of the command: `APROPOS move`.

See Table 3.1 for the command summary and `HELP` for the full help text.

```
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
```

Figure 3.1: Sample `APROPOS` Output.

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 3.3 summarizes the arguments. Tables Table 3.6, "Label Arguments.," on page 57 and Table 3.4, "Orientation Arguments for Labels.," on page 56 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 53). The current tick label format specifies how the tick labels will be written (see `FMTICK` on page 71). The current line width specifies how the axis will be drawn (see `LWEIGHT` on page 78). See Figure 2.5 for examples of using `AXIS`.

Argument	*Type*	*Description*
`w1`	real	WC at beginning of axis
`w2`	real	WC at end of axis
`minor`	real	Spacing of minor ticks; $< 0 \Rightarrow$ *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 3.3: Label Arguments.

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

Table 3.4: Orientation Arguments for Labels.

See `BOX`, below, for drawing default axes at the current plot scale.

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 3.6. 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 2.8 on page 31 for an example.

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

Table 3.5: Label Commands.

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.

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

Table 3.6: Label Arguments.

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`).

COLUMN

Data *Saved*

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

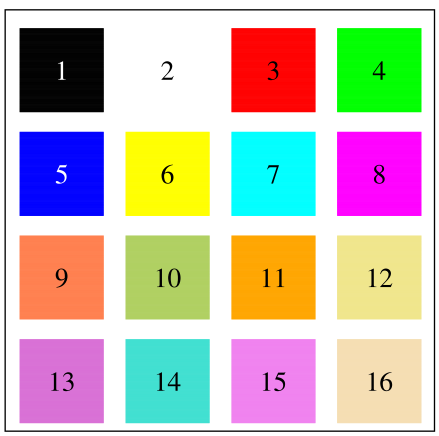
Read a column of data from the file specified by the `DATA` command. The command prefix indicates which data buffer to fill: X coordinates, Y coordinates, errors, point marker styles, limits, or scratch. If the input data is a text (ASCII list) file, column must be an integer specifying the column number. 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.

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.)



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

Figure 3.2: The Default Colors Available using `COLOR`.

Note that colors may be rendered on a monochrome device as shades of gray or dithered with some halftone pattern.

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.

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.

DDRAW

Plotting *Saved* *Undo*

DDRAW xpos ypos

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

See `DRAW` to draw in WC and `VDRAW` to draw in VPC.

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.

DMOVE

Plotting *Saved*

DMOVE xpos ypos
DRELOCATE xpos ypos

Pen up move, i.e., change the current position to the normalized device coordinates (NDC) specified by the two floating point parameters. If `:DRELOCATE` is used in cursor mode, `xpos` and `ypos` are ignored and may 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. This is an alias for `DRELOCATE`.

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 3.12 on page 84 for the marker type codes.

DRAW

Plotting | *Saved* | *Undo*

DRAW xpos ypos

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

See `DVRELOCATE` (or `DVMOVE`) to specify the initial position and see `DVDRAW` to specify the position in other coordinates.

DRELOCATE

Plotting | *Saved*

DRELOCATE xpos ypos
DMOVE xpos ypos

Pen up move, i.e., change the current position to the normalized device coordinates (NDC) specified by the two floating point parameters. If `:DRELOCATE` is used in cursor mode, `xpos` and `ypos` are ignored and may 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.

ECOLUMN

Data | *Saved*

ECOLUMN column

Read error values from the column in the current data file specified by the column parameter. If the input data is a text (ASCII list) 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 table column names are case-sensitive and may not be abbreviated.

See DATA to specify the input file (table) name. See ETYPE to specify the style of error bars and ERRORBAR for drawing them.

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.

EEVALUATE

Data | Saved

EEVALUATE expression

Replace the errors data vector by the result of the expression in the string argument. Typical expression syntax may be used to specify operators: +, -, *, /, **, and common functions: sqrt, sin, cos, log, log10, exp, etc. 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 ($e_i = \log i^2$). An array may operate on itself.

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.

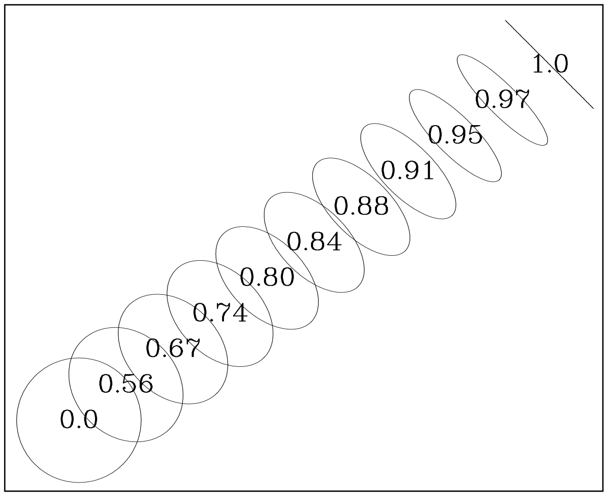


Figure 3.3: Examples of `ELLIPSE`.

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:

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

Table 3.7: Error Bar Position Arguments.

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 2.13 on page 43 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.

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)

Table 3.8: Error Bar Styles.

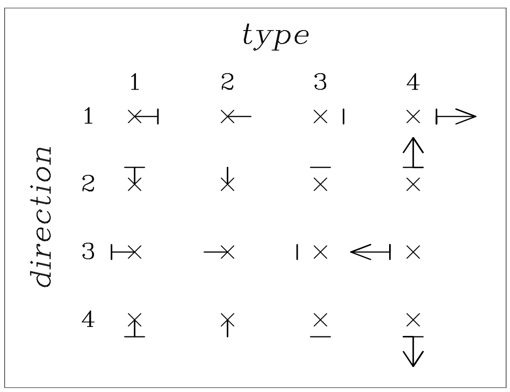


Figure 3.4: Error Bar Styles.

See the ERRORBAR command for drawing the error bars. See the ULLIM command for another way to draw upper or lower limit symbols.

EVALUATE

Data *Saved*

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

Replace a plot data vector by the result of the expression in the string argument. Typical expression syntax may be used to specify operators: +, -, *, /, **, and common functions: sqrt, sin, cos, log, log10, exp, etc. Note that the trig functions expect arguments in *degrees*. The current value of any of the four plot vectors is specified by the single 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, `XEVALUATE log10(r**2)` will replace each X value by the logarithm of the square of the element number. 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.

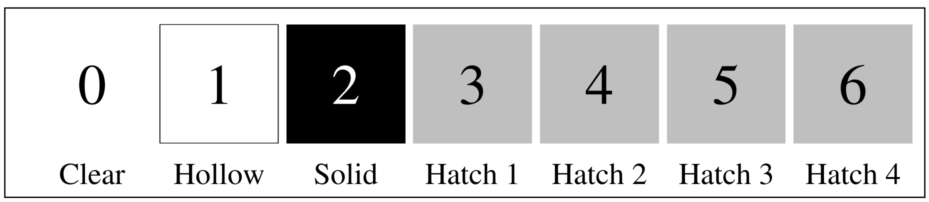


Figure 3.5: Filled-Area Patterns.

If no argument is specified, the current setting of the pattern index is printed.

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,

```
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
```

on a "landscape" format device would result in a viewport that was narrower than the one specified.

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.

FONTSET

Attribute *Saved*

FONTSET
FONTSET fonts

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`.

1. See the *SPP Reference Manual*, copies are available from the STSDAS Group at STScI.

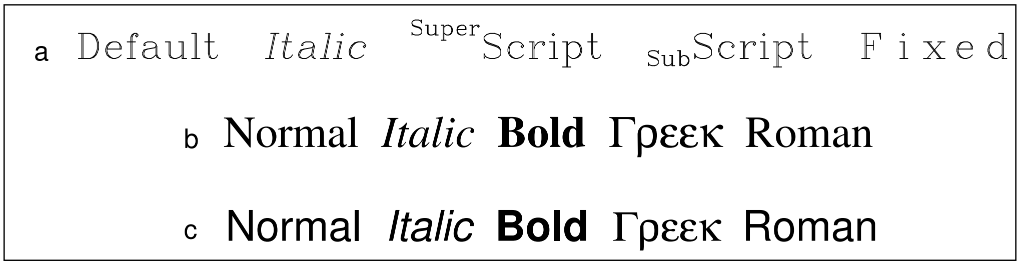


Figure 3.6: Differences Between igi and gio Fonts.

Figure 3.6 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 3.7 shows the results for various argument values. The $\times$ symbol indicates the coordinate of the pen when using `LABEL`.

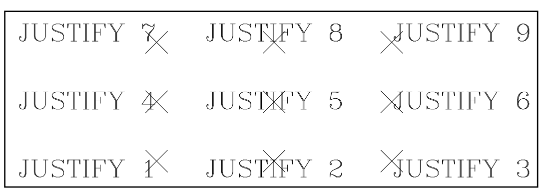


Figure 3.7: Text Justification Specified with `JUSITY` and `PUTLABEL`.

LABEL

Plotting *Saved* *Undo*

LABEL string

Draw text in the parameter string at the current position. 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 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`.

LCOLUMN

Data *Saved*

LCOLUMN column

Read a column of limits data from the file specified by the data command. If the input data is a text (ASCII list) 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 table column names are case-sensitive and may not be abbreviated.

See `DATA` to specify the input file (table) name. See `ULLIM` for use of the limits data in drawing upper or lower limit symbols.

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. 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 3.9 shows the default values for the arguments to `LOCATION`.

Argument	*Default*
left	0.1
right	0.95
bottom	0.1
top	0.95

Table 3.9: Viewport Location Arguments.

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 3.3.)

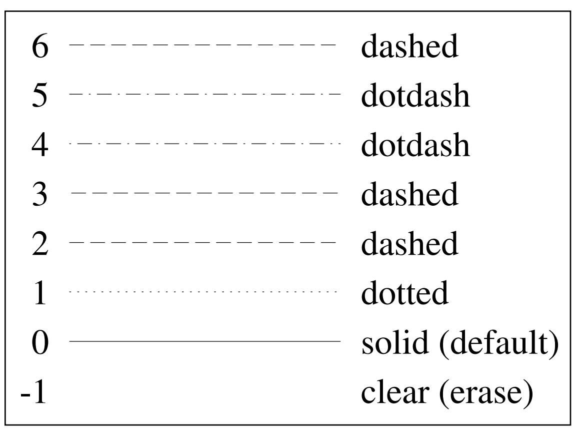


Figure 3.8: Line Types Selected with `LTYPE`.

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

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 `[XYEPLS]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.

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.

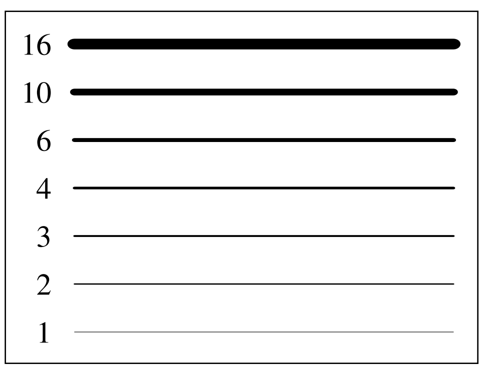


Figure 3.9: Line Weights Specified with `LWEIGHT`.

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.

```
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
```

Figure 3.10: Partial Output of the `MENU` Command.

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
RELOCATE xpos ypos

Pen up move, i.e., change the current position to the world coordinates (WC) specified by the two floating point parameters. This is an alias for `RELOCATE`.

If `:MOVE` is used in cursor mode, `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.

See `DRAW` to draw in WC, `VMOVE` to move in VPC, and `DMOVE` to move in NDC.

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.

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

Table 3.10: Boundaries at Which Scientific Notation is Used in Labels.

Figure 3.11 shows two examples of axes. Graph a was drawn using the default notation. Graph b was drawn with the command:

```
notation 1e-3 1e-4 1e-3 1e-4
```

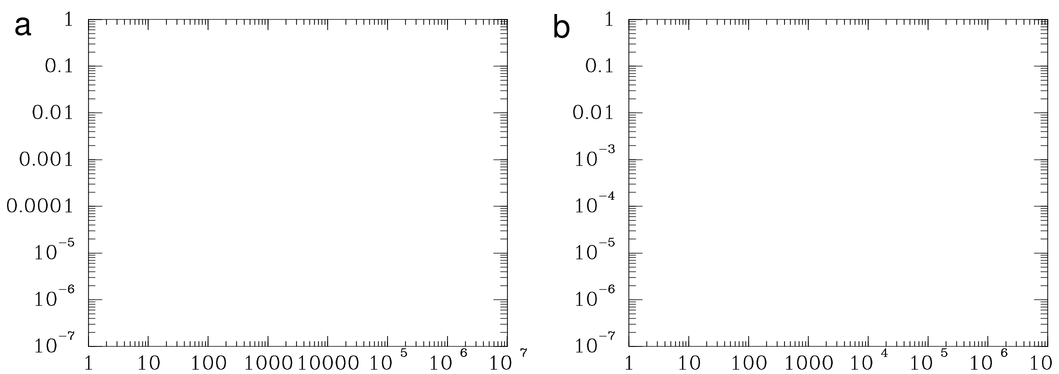


Figure 3.11: Effect of Using `NOTATION`.

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 2.1 on page 25 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.

PCOLUMN

Data *Saved*

PCOLUMN column

Read point specifications from the input data file (table). If the file is a text file, the column number must be an integer. If the file is an STSDAS table, the column name must be a string. Note that table column names are case-sensitive and may not be abbreviated.

See `DATA` to specify the input file (table) name.

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 3.12 on page 84 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.

PEVALUATE

Data *Saved*

PEVALUATE expression

Replace the point marker code data vector by the result of the expression in the string argument. Typical expression syntax may be used to specify operators: +, -, *, /, **, and common functions: sqrt, sin, cos, log, log10, exp, etc. 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`.

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.

Figure 1.5 on page 14 shows an example of using `PHYSICAL` to draw a grid in inches.

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.

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

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.

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 3.12 shows examples of various styles.

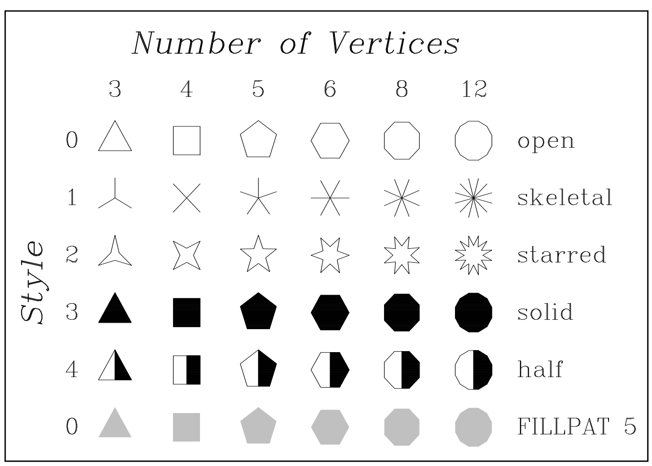


Figure 3.12: Types of Plotted Symbols Specified with PTYPE.

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 3.12.

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 2.12, "Text Escape Codes.," on page 36) for the text escape characters.

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
MOVE xpos ypos

Pen up move, i.e., change the current position to the world (user) coordinates specified by the two floating point parameters. If :RELOCATE is used in cursor mode, 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.

See DVRELOCATE to specify the position in NDC and VPC.

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.

```
# 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)
```

Figure 3.13: Sample SAOimage Colormap File.

Note that the colors displaying correctly depends on the graphics kernel and the output device. Only the PostScript kernel (`psikern`) supports this feature.

SCOLUMN

Data *Saved*

SCOLUMN column

Read a column of scratch data from the file specified by the data command. If the input data is a text (ASCII list) 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 table column names are case-sensitive and may not be abbreviated.

See `DATA` to specify the input file (table) name. See `EVALUATE` for use of the scratch data in manipulating the data buffers.

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.

There is no capability to draw any multi-dimensional plots or gray scale images. 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.

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.

Warning: This command, if used with `IMGWCS`, will modify the contents of the X plot buffer (see above).

SHOW

Attribute

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

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 2.1 on page 25 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.

VDRAW

Plotting | Saved | Undo

VDRAW xpos ypos

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

See `DRAW` to draw in WC and `DDRAW` to draw in NDC.

VMOVE

Plotting *Saved*

VMOVE xpos ypos
VRELOCATE xpos ypos

Pen up move, i.e., change the current position to the virtual page coordinates (VPC) specified by the two floating point parameters. This is an alias for `VRELOCATE`.

If `:VMOVE` is used in cursor mode, `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.

See `VDRAW` to draw in VPC, `MOVE` to move in WC, and `DMOVE` to move in NDC.

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).

VRELOCATE

Plotting *Saved*

VRELOCATE xpos ypos
VMOVE xpos ypos

Pen up move, i.e., change the current position to the virtual page coordinates (VPC) specified by the two floating point parameters. If `:VRELOCATE` is used in cursor mode, `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.

See `VDRAW` to draw in VPC, `RELOCATE` to move in WC, and `DRELOCATE` to move in NDC.

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.

XCOLUMN

Data *Saved*

XCOLUMN column

Read X coordinate data from the specified column in the current input data file. If the file is a text file, the column number must be an integer. If no X data are specified, the Y data are plotted against row number. If the file is an STSDAS table, the column name must be a string. Note that table column names are case-sensitive and may not be abbreviated.

See `DATA` to specify the input file (table) name.

XEVALUATE

Data *Saved*

XEVALUATE expression

Replace the X plot data vector by the result of the expression in the string argument. Typical expression syntax may be used to specify operators: +, -, *, /, **, and common functions: sqrt, sin, cos, log, log10, exp, etc. The current value of any of the four plot vectors is specified by the single 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, `XEVALUATE log10(r**2)` will replace each X value by the logarithm of the square of the element number. The X array may operate on itself.

XFLIP

Scaling *Saved*

Flip the X axis. That is, plot horizontal data and axis labels increasing right to left instead of the default left to right. 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 X limits reversed, but operates on the currently set limits, whether set manually or automatically.

XLABEL

Plotting *Saved* *Undo*

XLABEL text

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

Note that the label will not 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.

XLOGARITHM

Data *Saved*

Take the common logarithm of the X data values. 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 `XEVALUATE` for an alternate way of doing this. You must use `XYLOGARITHM` for `ERROBAR` to work correctly on log data.

XSECTION

Data *Saved*

XSECTION image
XSECTION image project
XSECTION

Read an IRAF (OIF, or STF) image as the *X* column values. The image may be of any dimensionality and data type. The image name 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. There is no capability to draw any multi-dimensional plots or gray scale images.

`XSECTION` overrides the use of the `DATA` and `XCOLUMN` commands in filling the X data array. 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 a string beginning with `a`, `s`, or `v`, for “average”, “sum”, or “vector,” respectively. Only the first character of the string is significant and it may be either upper or lower case.

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 average
```

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

`XSECTION` saves the input file name string with the section specification in the X axis label buffer. This may be used with the `XLABEL` command with no arguments to label the X axis with the section.

YCOLUMN

Data *Saved*

YCOLUMN column

Read Y coordinate data from the specified column in the current input data file. If the file is a text file, the column number must be an integer. If the file is an STSDAS table, the column name must be a string. Note that table column names are case-sensitive and may not be abbreviated. If no X data are specified, the Y data are plotted against row number.

See `DATA` to specify the input file (table) name.

YEVALUATE

Data *Saved*

YEVALUATE expression

Replace the *Y* data vector by the result of the expression in the string argument. Typical expression syntax may be used to specify operators: `+`, `-`, `*`, `/`, `**`, and common functions: sqrt, sin, cos, log, log10, exp, etc. 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, `YEVALUATE log10(r**2)` will replace each Y value by the log of the square of the element number. An array may operate on itself.

YFLIP

Scaling *Saved*

Flip the Y axis. That is, plot vertical data and axis labels increasing top to bottom instead of the default bottom to top. This must be used before the axes are drawn (`BOX`) for the plot to scale correctly. This is equivalent to specifying `LIMITS` with the lower and upper Y limits reversed, but operates on the currently set limits, whether set manually or automatically.

YLABEL

Plotting *Saved* *Undo*

YLABEL text

Draw text centered left of the left Y (vertical) axis. The text escape characters may be used to specify different fonts, superscripts and subscripts (See `LABEL`). 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.

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

YLOGARITHM

Data *Saved*

Take the common logarithm of the Y data values. 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 `YEVALUATE` for an alternate way of doing this. You must use `[XY]LOGARITHM` for `ERROBAR` to work correctly on log data.

YSECTION

Data | Saved

YSECTION image
YSECTION image project

Read an IRAF (OIF, or STF) image as the Y column values. The image may be of any dimensionality and data type. The image name 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. There is no capability to draw any multi-dimensional plots or gray scale images.

`YSECTION` overrides the use of the `DATA` and `YCOLUMN` commands in filling the Y data array. 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 a string beginning with `a`, `s`, or `v`, for "average", "sum", or "vector" respectively. Only the first character of the string is significant and it may be either upper or lower case.

For example, the command:

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

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

```
igi> YSECTION myimage average
```

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

`YSECTION` saves the input file name string with the section specification in the Y axis label buffer. This may be used with the `YLABEL` command with no arguments to label the Y axis with the section.

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.

Warning: This command, if used with `IMGWCS`, will modify the contents of the X plot buffer (see above).

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 to use the 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.

ZSECTION

Data *Saved*

ZSECTION section

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 `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.

APPENDIX A:

Source for Figures

This appendix includes the **igi** source to several of the figures in this document. It may be useful as examples of various features of **igi**. They are more extensive examples than the source listed within the text and demonstrate capabilities other than simple plotting. See the particular section of the document for the plot resulting from these commands.

Axes

Example A.1 demstrates drawing axes using the BOX and AXIS commands. See Figure 2.5 on page 29.

```
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
```

Example A.1: Axis Code.

Curve Styles

This example demonstrates the various styles of plotted curves. See Figure 2.1 on page 25 .

```
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
```

Continued...

Example A.2: Code for Generating Curve Styles Example.

```
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"
```

Example A.2 (Continued): Code for Generating Curve Styles Example.

Marker Types

This example demonstrates the style of centered symbols drawn with the `DOT` and `POINTS` commands. See Figure 3.12 on page 84.

```
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"
```

Continued...

Example A.3: Code for Generating Symbol Types.

```
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
```

Example A.3 (Continued): Code for Generating Symbol Types.

Line Types

This example demonstrates the various line styles available using the `LTYPE` command and different line widths selected using `LWEIGHT`. See Figure 3.3 on page 69.

```
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)
```

Continued...

Example A.4: Code for Generating Line Weights Example.

```
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
```

Example A.4 (Continued): Code for Generating Line Weights Example.

Coordinate Systems

Virtual Page

This example demonstrates virutal page coordinates (VPC) and the `VPAGE` command. See Figure 1.3 on page 13. This includes examples of defining and invoking macros.

```
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
```

Continued...

Example A.5: Code for Generating Virtual Page Example.

```
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"
```

Example A.5 (Continued): Code for Generating Virtual Page Example.

Windowing

This example further demonstrates virtual page coordinates using the `WINDOW` command. See Figure 1.3 on page 13.

```
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
```

Continued...

Example A.6: Code for Generating Window Example.

```
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"
```

Example A.6 (Continued): Code for Generating Windowing Example.

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.*

axis: The edge of a plot having ticks and labels. The X axis typically goes horizontally across an image, the Y axis vertically. Also, 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.

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 text string 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

I

J

L

M

N

P

Q

R

S

T

U

V

W