IDL 5.3
Things you need to know

Manuals

The manuals are available as PDF files. They should be loaded on the computers, if not, we need to ask for them to be installed. They are in PDF format and require Acrobat reader 3.0 or higher. To get to the online manuals:

On Windows, select Start -> Programs -> Research Systems IDL 5.3 -> IDL Online Manuals

On Macintosh, a shortcut can be found in the rsi-directory:RSI:IDL 5.3 folder named IDL Online Manuals.

On UNIX, execute the following at the UNIX prompt:

idlman

This is the best set of online manuals I have seen for any program. They are exact copies of the hard copy manuals and have been carefully developed over 15 years.

Getting Started with IDL – Everyone new to IDL should look though this manual. It quickly covers many important features and points you to the manuals that have more details about the topics. It is about 200 pages long with lots of pictures and examples for you to try. So it should be used when you are at a computer so that you can try everything out. This manual covers the IDL Development Environment, Reading and Writing data, plotting, signal processing, image processing, surface and contour plotting, volume visualization, mapping (which few of you will use), plotting irregularly-gridded data, animation, programming in IDL, manipulating data and using the IDL GUIBuilder. It even has a chapter that is a road map of the rest of the documentation set and manuals.

List of other Manual Titles
Using IDL – explains IDL from an interactive user's point of view.
Building IDL Applications – explains how to use the IDL language to write programs
IDL Reference Guide – contains detailed information about all of IDL's procedures, functions, objects, system variables, and other useful reference materials.
External Development Guide – explains how to use IDL to develop applications that interact with programs written in other programming languages.
Scientific Data Formats – contains detailed information about IDL's routines for dealing with specific data formats such as CDF, HDF, HDF-EOS and NetCDF.

Help from inside the IDL program:
Once you are inside the IDL program you can find hypertext linked help for all topics by using the Help pull down menu in the IDLDE environment or typing a ? at the IDL prompt for those not using the development interface on Unix machines.

Short History and Philosophy
IDL was originally developed by David Stern (now president of Research Systems, Inc) as a higher level programming language that he could use to process images obtained from telescopes and satellites. The language was originally developed under the VMS operating systems in Fortran. This made sense at the time since the Fortran compiler provided with VMS was the most optimized compiler available on any system at the time. The program became very popular with the astronomy community and was often sold with certain image display boards that could be bought for Vax and MicroVax computers. RSI never intended to port IDL to Unix workstations, originally. Instead, Precision Visuals licensed the program from RSI and began that development under the name of PVWave. The laboratory that I was in at the time had used IDL for five years for our magnetic resonance image and spectroscopy display system and was just beginning to look into Unix workstations in 1989. Precision Visuals said they would sell us the program for $10,000. However, a new copy of IDL cost less than $2000 at the time. We told the Silicon Graphics sales representative that we would not buy an SGI computer unless IDL ran on it. SGI quickly investigated this software and decided it was software they definitely wanted on their systems. They gave RSI some loaner computers to port IDL to SGI. Precision Visuals and RSI severed their ties and since then IDL and PVWave has diverged. But many of the basics are the same in both.

IDL was originally designed with image processing in mind. This is not something that was added as a toolbox later. For this reason, it handles multidimensional arrays with ease. There are large numbers of procedures and functions that you can call that deal with image processing. In addition, there are many ways to visualize the images. It has evolved to be a very powerful programming language as well. One thing I like to do with IDL is to test out a programming algorithm in interactive mode where I can view each step as an image or plot. Then I will use these commands in a procedure or function to test my idea out on a wide variety of images or specific situations. If IDL provides answers fast enough, I stop here. If I need it to run faster, I rewrite the computationally heavy routines in C, C++ or Fortran and call them from IDL in one of several ways which we will get into later. In the end, I might only be using IDL for the interface and visualization steps.

Let’s reiterate those steps with some details about IDL:

IDL can be run totally in interpretive mode. You type at the IDL> prompt and upon hitting the return key, the command is executed. Thus, you can work out how commands work, and your ideas at the keyboard outside of a large complicated program that might be introducing other errors. For instance, you can type this line at the prompt:

IDL> plot,sin(findgen(100)*!dtor*3.6)
(!dtor is a system variable that convert degrees to radians, findgen(100) produces a float array of 100 elements starting at 0 and ramping up to 99)

This should plot a single cycle of a sine wave. If it doesn't, you can quickly adjust your numbers so that it does.

IDL can be run as a compiled language. You can write your own procedures and functions and call then from procedures and functions. The main program can be run at the command prompt by typing .run myprogramname.pro. It will be compiled the first time and afterward can simply be invoked by typing its procedure name. For instance, the single line above could be written as a procedure like the one below:

```
pro singlecycle,scalefactor,output
;       singlecycle is the procedure name and scalefactor is the input
;       output is what will be returned.
;       This should be saved as singlecycle.pro

ramp = findgen(100)
arg = ramp*scalefactor*!dtor
output = sin(arg)

end
```

Not a very exciting program since we know we can type it as a single line but it allows me to introduce a few concepts so bare with me.
The first line of the program declares this as a procedure. That means it does not return a value. At the IDL prompt, we would call this procedure by typing the following:

IDL> singlecycle, 3.6, forplot

3.6 will be used as the value for scalefactor and forplot will hold what was calculated and placed in output. Anything that was stored in forplot before the call to singlecycle will be lost. We can use forplot to do other things such as

IDL> plot, forplot

If you haven't guessed, the semicolons signify to the compiler that anything after them is to be ignored and therefore indicates comments. Another thing to note, is that ramp, arg and output are all arrays of 100 elements since findgen created an array of 100 elements. Also, these arrays do not need to be declared ahead of time. They take on the size of ramp and their data type is determined by the calculation. Since findgen creates a floating point array, arg will be a floating point array even if scalefactor is an integer. Also, both findgen and sin are functions. You can also write your own functions. In fact, since there is a single output for the procedure above, it might be better as a function.

```
funct singlecycle,scalefactor
;       singlecycle is the function name and scalefactor is the input
;       This should be saved as singlecycle.pro

ramp = findgen(100)
arg = ramp*scalefactor*!dtor
return, sin(arg)

end
```

This would be called in the following way:

IDL> output = singlecycle(3.6)

If you wanted to plot the output you could call this function in the following way:

IDL> plot, singlecycle(3.6)

One thing to note, if you name the file the same as the procedure or function call with .pro as the extension and place it in a directory on the path IDL looks at, you need not compile the function or procedure before use. IDL will look in its own library and then in any directory along the !path system variable to see if a .pro file exists with the procedure or function name.

IDL Syntax

IDL is not case sensitive. RED = REd = Red = red = rEd = rED = ReD = reD

Function: Result=Function(argument1, argument2, optargument, keyword=value, /keyword)

Procedure: Procedure, argument1, argument2, optargument, keyword=value, /keyword)

Statements

assignment variable = expression assigns a value to a variable

defines a **block of statement** (same as { } in C)

```
BEGIN
    Statement1
    Statement2
    Statement2
END
```

CASE … ENDCASE = selects one statement for execution depending on the value of the expression

```
case expression of
    expression: statement
    expression: statement
    expression: statement
    else:statement          (optional)
endcase
```

common = common block

```
common block_name, variable1, variable2, variable3 … variablen
```

for statements

```
for variable=initial_value, limit, increment do
or
for variable=initial_value, limit, increment do begin
    statement1
    …
endfor
```

goto - transfers program control to point specified by label

```
goto, label
```

if … then … else

```
if expression then statement
if expression then begin
        statements
endif
if expression then statement else statement
if expression then begin
        statements
endif else statement
if expression then begin
        statements
endif else begin
        statements
endelse
```

repeat … until - statements always executed at least once

```
repeat statement until expression
repeat begin
        statements
endrep until expression
```

while … do - statements are never executed if condition is initially false

```
while expression do statement
while expression do begin
        statements
endwhile
```

Executive Commands

Executive commands must be entered at the IDL command prompt. They cannot be used in programs.

.compile compiles programs without running
.continue continues execution of a stopped program
.reset_session resets much of the state of IDL session without requiring user to exit and restart IDL
.full_reset_session does everything as .reset_session but also unloads sharable libraries
.go executes previously compiled main program
.out continues execution until curent routine returns
.return continues execution until return statement
.rnew erases main program variables and then does .run
.run compiles and executes IDL commands from files or keyboard
.skip skips over next n statements and then single steps
.step execues one or n statements from the current position
.stepover executes a single statement if the statement doesn't call a routing
.trace similar to .continue, but displays each line of code before execution

Special Characters

Ampersand (&) – separates multiple commands on a single line
Apostrophe (') – delimits strings or indicates octal or hex
Asterisk (*) – designates an ending subscript range equal to the size of the dimension. Also the multiplication operator and the pointer dereference operator
At sign (@) – include character. Used at beginning of a line to cause the IDL compiler to substitute the contents of the file whose name appears after the @ symbol for the line. In interactive mode, the @ symbol is used to execute a batch file.
Colon (:) – ends label identifiers. Also separates start and end subscript ranges
Dollar Sign ($) – continuation character (at the end of line) or spawn operating system command (at start of line)
Exclamation Point (!) – First character of system variable names and font-positioning commands
Period (.) – first character of executive commands. Also indicates floating-point numbers
Question Mark (?) – invokes the online help facility
Quotation Mark (") – string delimiter or indicates octal number
Semicolon (;) – first character of comment field

OPERATORS

Mathematical Operators

+	Addition, String Concatenation
-	Subtraction and Negation
*	Multiplication, Pointer dereference
/	Division
^	Exponentiation
MOD	modulo
<	The minimum operator
>	The maximum operator
# and ##	Matrix multiplication

Boolean Operators

AND	Boolean AND
NOT	Boolean complement
OR	Boolean OR
XOR	Boolean exclusive OR

Relation Operators

EQ	equal to
GE	greater than or equal to
GT	greater than
LE	less than or equal to
LT	Less than
NE	Not equal to

Other Operators

[ ]	array concatenation, enclosed array subscripts
( )	group expressions to control order of evaluation
=	assignment
?:	conditional expression

Operator Precedence

Highest	( ) expression groups
Second	* (pointer dereference), ^ (exponentiation)
Third	* (multiplication), # and ## (matrix multiplication), / (division), MOD (modulus)
Fourth	+, -, <, >, NOT (Boolean negation)
Fifth	EQ, NE, LE, LT, GE, GT
Sizth	AND OR XOR

Seventh ?: (conditional expression)

Where to look for Example Code

Many of the procedures or functions that you see in the reference manual where also written in IDL. They are stored in the lib directory under the idl53 directory that is most likely under the RSI directory. However, the name of the top level directory is chosen by the person who installed the program. You can look at the .pro files in the lib directory but if you want to change them, please make a copy and put in your own directory. Do not change the ones in the lib directory. This will change how this function or procedure works for everyone who uses IDL on that computer. This is a great place to see good efficient IDL code. These were written by people who work for RSI. Also, looking at this directory gives you a quick overview of some of the available functionality at your fingertips. Be careful though, once RSI determined that some of these features were highly valuable, they rewrote them to be more efficient. For a few of these procedures and functions, there are new internal (i.e. written in C) procedures or functions that work faster. The .pro files are retained for backward compatibility.

You will also see some files with .sav extensions. These are IDL save sets. You can compile a several functions and/or procedures, assign some variables and then save the whole thing in a save set using the IDL save procedure. Then the next time you come into IDL, you can restore this session using the restore procedure and return to the IDL state you were in when you saved the session. There is also a journaling function. I don't use either much but feel free to explore.

Wrap up

You will be hard pressed to find something IDL cannot do. You might think you have to write your own procedure or function to do something but you should thoroughly investigate what is available before you bother. I am often amazed at what I find. Sometimes a simple keyword addition will allow a function to do exactly what I want. Other times, I find a new functionality that was not there in an earlier version and is there now. Sometimes just asking someone else helps me find a more efficient way of doing things than how I would have proceeded. This is a very rich programming environment that allows you to write low level code and/or use already provided features.