GDL – GNU Data Language

a free/libre/open-source implementation of IDL/PV-WAVE*

developed by Marc Schellens and The GDL team

documentation maintained by Sylwester Arabas and Alain Coulais

January 3, 2012

* IDL (Interactive Data Language) and PV-WAVE (Precision Visuals - Workstation Analysis and Visualization Environment) are (were) registered trademarks of EXELIS VIS (ITT VIS; Research Systems, Inc.) and Rogue Wave Software (Visual Numerics; Precision Visuals), respectively

Contents

About GDL

GNU Data Language (GDL) is a free/libre/open source incremental compiler compatible with IDL and to some extent with PV-WAVE. Together with its library routines it serves as a tool for data analysis and visualization in such disciplines as astronomy, geosciences and medical imaging.

GDL as a language is dynamically-typed, vectorized and has object-oriented programming capabilities. GDL library routines handle numerical calculations, data visualisation, signal/image processing, interaction with host OS and data input/output. GDL supports several data formats such as netCDF, HDF4, HDF5, GRIB, PNG, TIFF, DICOM, etc. Graphical output is handled by X11, PostScript, SVG or z-buffer terminals, the last one allowing output graphics (plots) to be saved in a variety of raster graphics formats. GDL features integrated debugging facilities. GDL has also a Python bridge (Python code can be called from GDL; GDL can be compiled as a Python module).

Packaged versions of GDL are available for several Linux and BSD flavours as well as Mac OS X. The source code compiles as well on other UNIX systems, including Solaris. GDL source code is available for download from Sourceforge.net at: http://sourceforge.net/projects/gnudatalanguage/.

Other open-source numerical data analysis tools similar to GDL include:

— GNU Octave: http://www.gnu.org/software/octave/
— NCL – NCAR Command Language: http://www.ncl.ucar.edu/
— PDL – Perl Data Language: http://pdl.perl.org/
— R: http://www.r-project.org/
— Scilab: http://www.scilab.org/
— SciPy: http://www.scipy.org/
— Yorick: http://yorick.sourceforge.net/

License

GDL is a free, libre and open-source software released under the GNU General Public License version 2 Fundation [1]. It basicaly means that any GDL user has the freedom to run, copy, distribute, study, change and improve GDL.

Credits

GDL have been developed by a team of volunteers led by **Marc Schellens** – the project's founder and maintainer. As of 2011 the core team consists additionally of (in alphabetical order) Sylwester Arabas, Alain Coulais and Jeol Gales.

Among many good folks who provided patches and valuable feedback (in alphabetical order) there are: Médéric Bocquien, Justin Bronn, Pierre Chanial, Pedro Corona Romero, Gilles Duvert, Christoph Fuchs, Nicolas Galmiche, Greg Huey, Gaurav Khanna, Christopher Lee, Maxime Lenoir, Peter Messmer, Gregory Marchal, Thibaut Mermet, Lea Noreskal, Orion Poplawski, Rene Preusker, Mateusz Turcza, Joanna Woo, H Xu, . . .

GDL contains snippets of code borrowed from other free and open-source projects credited to: Deepak Bandyopadhyay, Sergio Gelato, Lutz Kettner, Craig B. Markwardt, Paul Ricchiazzi, Danny Smith, J.D. Smith, Richard Schwartz, Paul Wessel, Bob Withers, . . .

Pre-compiled or pre-configured packages of GDL are available for numerous systems thanks to: Juan A. Añel, Axel Beckert, Markus Dittrich, Takeshi Enomoto, Sébastien Fabbro, Orlando Garcia Feal, Gaurav Khanna, Justin Lecher, Sebastien Maret, Lea Noreskal, Orion Poplawski, Marius Schamschula, Gürkan Sengün, Thierry Thomas, . . .

GDL is written in C++ using the Terence Parr's ANTLR language-recognition framework. Most of the library routines are implemented as interfaces to open-source packages such as GNU Scientific Library, PLPlot, FFTW, ImageMagick, and many many more.

Last but not least, we would like to acknowledge the designers of IDL and PV-WAVE.

Please do report any missing name on the lists above in the same way as any other bug in GDL (see section below).

Providing fedback

Your comments are welcome! Let us know what you use GDL for. Or if you don't, why not. Which functionality are you missing/would appreciate most for comming versions. Please send your bug reports, complaints, suggestions, comments and patches using the trackers or forums available at GDL's project website at SourceForge: http://sourceforge.net/projects/gnudatalanguage/.

Organization of this document

This document is divided into two parts:

— User's guide: intended for users developing programs written in GDL,
— Developer's guide: intended for those interested in developing or packaging GDL.

Most of GDL functionalities are exemplified with short GDL scripts. For each such script there are two listings provided: a source code listing with line numbers to the left and a log of output below, e.g.:

```
print, 'Hello world!'
```

```
Hello world!
```

All scripts are run by invoking `gdl script.pro` what is equivalent to loading the script with the @ operator or typing every line of script at the GDL's interactive mode command prompt.

Often the scripts contain lines beginning with a dollar sign "$" which is the GDL syntax for executing shell commands, e.g.

```
$ echo "Hello world!"
```

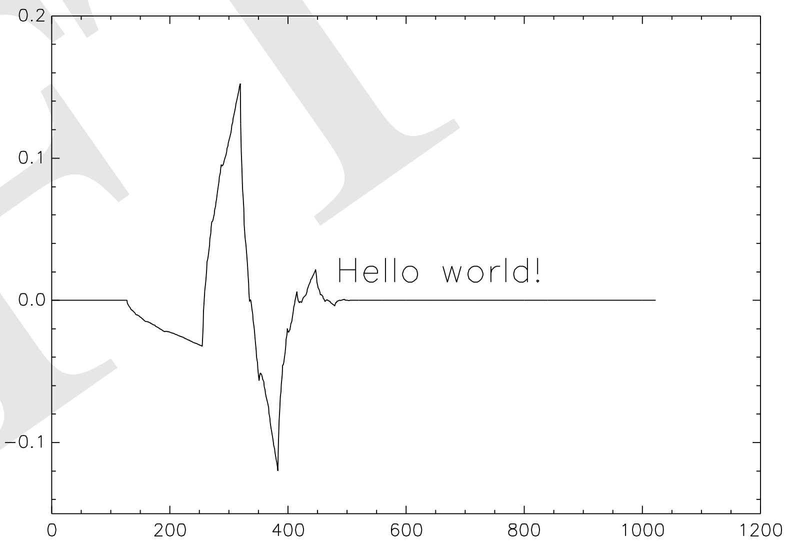
```
Hello world!
```

If a script involves creation of a plot, the resultant postscript file is displayed below the output listing, e.g.:

```
plot, wtn([fltarr(9), 1, fltarr(1014)], 4, /inverse)
xyouts, 480, .02, 'Hello world!', charsize=2
```

```

```



While GDL itself reached a beta status of development, the hereby documentation is far from reaching an alpha status – **help is very welcome!**

Part I

User's guide

Chapter 1

Obtaining, installing, and invoking GDL

Requirements and supported environments

Availability of pre-compiled packages

Compiling GDL from source

Compiler requirements

GNU g++ clang Intel C++

Autotools

Cmake

Installation layout

Command-line options

Influential environmental variables

Chapter 2

Language reference

Syntax basics

IDL_VALIDNAME() TEMPORARY()

Datatypes

ASSOC()

BYTE() COMPLEX(), DCOMPLEX() (CONJ(), ATAN(), IMAGINARY(), REAL_PART())
DOUBLE() FIX() FLOAT() LONG() LONG64() UINT() ULONG() ULONG64()

SIZE()

Operators

LOGICAL_AND() LOGICAL_OR() LOGICAL_TRUE()

SQRT()

Flow control structures

Conditional execution

IF

```
a = 10
if a gt 5 then print, 'a is greater than 5'
```

```
a is greater than 5
```

```
a = 10
if a gt 5 then print, 'a > 5' else print, 'a <= 5'
```

```
a > 5
```

contrary to... cannot be used in interactive mode nor in batch scripts, but only within ...

```
$ cat replace_with_nans.pro
x = [1.1, 2.1, -3.3, 4.1, -999, 6]
replace_with_nans, x, -999
print, x
```

```
pro replace_with_nans, x, val
  whr = where(x eq val, cnt)
  if cnt gt 0 then begin
    x[whr] = !VALUES.F_NAN
    message, 'nan count: ' + strtrim(cnt, 2), /conti
  endif
end
% Compiled module: REPLACE_WITH_NANS.
% REPLACE_WITH_NANS: nan count: 1
      1.10000      2.10000     -3.30000      4.10000           nan
```

data type	size	constants	min	max	casting	array allocation	index array alloc.	freeing
natural numbers incl. zero (unsigned)	8b	1b	0	255	BYTE()	BYTARR()	BINDGEN()	TEMPORARY()
	16b	1u	0	65535	UINT()	UINTARR()	UINDGEN()	
	32b	1ul	0	$4{\cdot}10^9$	ULONG()	ULONARR()	ULINDGEN()	
	64b	1ull	0	$1{,}8{\cdot}10^{19}$	ULONG64()	ULON64ARR()	UL64INDGEN()	
integer numbers (signed)	16b	1	-32768	32767	FIX()	INTARR()	INDGEN()	TEMPORARY()
	32b	1l	$-2{\cdot}10^9$	$2{\cdot}10^9$	LONG()	LONARR()	LINDGEN()	
	64b	1ll	$-9{\cdot}10^{18}$	$9{\cdot}10^{18}$	LONG64()	LONG64ARR()	L64INDGEN()	
real numbers	32b	1.	-10^{38}	10^{38}	FLOAT()	FLTARR()	FINDGEN()	TEMPORARY()
	64b	1d	-10^{308}	10^{308}	DOUBLE()	DBLARR()	DINDGEN()	
complex numbers	64b	complex(1,0)	2x float	2x float	COMPLEX()	COMPLEXARR()	CINDGEN()	TEMPORARY()
	128b	dcomplex(1,0)	2x double	2x double	DCOMPLEX()	DCOMPLEXARR()	DCINDGEN()	
character (byte) strings	variable	'one'	–	–	STRING()	STRARR()	–	TEMPORARY()
structures	variable	{a:1, b:1}	–	–	–	REPLICATE()	–	TEMPORARY()
pointers	n/a	ptr_new(1)	–	–	–	PTRARR()	–	PTR_FREE()
objects	n/a	obj_new('One')	–	–	–	OBJARR()	–	OBJ_DESTROY()

CASE

SWITCH

Loops

FOR

FOREACH

FOREACH statement allows to simplify loop constructs when the array index is not used within the loop:

```
tocompare = ['apples', 'orrages']
foreach a, tocompare do help, a
```

```
A               STRING    = 'apples'
A               STRING    = 'orrages'
```

As with index variables in FOR loops, the lifetime of the "loop variables" in FOREACH statements extends beyond the loop execution (see example below). Both BREAK and CONTINUE statements work in FOREACH in the same way as in other loop constructs:

```
$ cat example.pro
example
```

```
pro example
  letters = ['a', 'b', 'c', 'd', 'e']
  foreach l, letters do begin
    if l eq 'c' then continue
    if l eq 'd' then break
    print, 'trying to replace '+ l + ' with ''x'''
    l = 'x'
  endforeach
  print, letters
  print, l
end
% Warning: Assignment to loop variable detected.
% Compiled module: EXAMPLE.
trying to replace a with 'x'
trying to replace b with 'x'
a b c d e
d
```

Loop variables in FOREACH statements contain copies of the array elements thus assigning them a value within the loop does not change contents of the array and as a potentially bug-prone situation causes a compiler warning (see example above).

REPEAT

WHILE

Jumps

GOTO

Highly deprecated as it usually make the code difficult to read and prone to errors. Anyhow, the syntax is as follows

```
$ cat example.pro
example
```

```
pro example
  x = 0
  goto, a
  x++
  a: print, 'x = ', x
end
% Compiled module: EXAMPLE.
x =        0
```

As most of the flow control operator described in this section GOTO is usable only within a GDL routine – not within a batch script which is equivalent to a series of statements in the interactive mode.

Other

EXECUTE()

Variable scoping rules

```
```

```
```

Functions and procedures

There may exist a function and a procedure of the same name (e.g. PYTHON() and PYTHON, CALL_METHOD() and CALL_METHOD)

EXPAND_PATH(), FILEPATH()

CALL_FUNCTION() CALL_PROCEDURE()

Argument passing

N_PARAMS() KEYWORD_SET() ARG_PRESENT() N_ELEMENTS() SIZE()

_EXTRA _STRICT_EXTRA _REF_EXTRA

when by reference, when by value...

Keyword name abbreviations are allowed if unambiguous, e.g.:

```
help, strpos('kayak', 'a', /reverse_search)
help, strpos('kayak', 'a', /reverse_s)
help, strpos('kayak', 'a', /rev)
help, strpos('kayak', 'a', 2, /reverse_search, /reverse_offset)
```

```
<Expression>    LONG      =           3
<Expression>    LONG      =           3
% STRPOS: Ambiguous keyword abbreviation: REV
% Execution halted at: $MAIN$
<Expression>    LONG      =           1
```

Arrays

PRINT (TV) PM

N_ELEMENTS() SIZE()

REFORM() REBIN() REVERSE() ROTATE() TRANSPOSE()

SORT() UNIQ()

WHERE() ARRAY_INDICES()

ARRAY_EQUAL()

MAKE_ARRAY() REPLICATE() REPLICATE_INPLACE

BYTARR() COMPLEXARR() DBLARR() DCOMPLEXARR() FLTARR() INTARR() LON64ARR() LONARR() OBJARR() PTRARR() STRARR() UINTARR() ULON64ARR() ULONARR()

BINDGEN() CINDGEN() DCINDGEN() DINDGEN() FINDGEN() INDGEN() L64INDEGEN() LINDEGEN() SINDGEN() UINDGEN() UL64INDGEN() ULINDGEN()

IDENTITY()

Structures

CREATE_STRUCT() N_TAGS() STRUCT_ASSIGN TAG_NAMES()

System variables (global)

DEFSYSV (checking if running GDL)

Heap variables (pointers)

HEAP_GC PTRARR PTR_FREE PTR_NEW() PTR_VALID()

The HELP procedure

HELP

Object-oriented programming

CALL_METHON CALL_METHON() OBJARR()

OBJ_CLASS() OBJ_DESTROY OBJ_ISA() OBJ_NEW() OBJ_VALID()

Handling Overflows, Floating Point Special Values

CHECK_MATH() FINITE() MACHAR()

Error handling

MESSAGE CATCH ON_ERROR ON_IOERROR EXECUTE

Compile options

```
$ cat example.pro
help, 1
example
```

```
pro example
  compile_opt idl2
  help, 1
end
<Expression>    INT       =        1
% Compiled module: EXAMPLE.
<Expression>    LONG      =              1
```

```
$cat example.pro
example
```

```
pro example_helper
  compile_opt hidden
  print, 'example procedure helper'
end
pro example
```

```
    example_helper
end
% Compiled module: EXAMPLE.
example procedure helper
```

Chapter 3

Interpreter commands and built-in debugging facilities

MESSAGE RETALL STOP .COMPILE .STEP .CONTINUE

CHECK_MATH

JOURNAL RECALL_COMMANDS

MEMORY (TEMPORARY())

RESOLVE_ROUTINE ROUTINE_INFO() ROUTINE_NAMES() SCOPE_VARFETCH()

Chapter 4

Maths

Basic Scalar, vector and array operations

TOTAL() SQRT() REVERSE() SHIFT() MAX() MIN() MEAN() NORM() CONVOL() PRODUCT() CROSSP() DERIV() INVERT() MATRIX_MULTIPLY() TRACE() TRANSPOSE() (ROTATE())

UNIQ()?

Basic and special function library

GDL has a built-in collection of mathematical functions that are listed below. A great majority of these routines accept both scalar and vector arguments of any numerical type and return the result as scalars or vectors, respectively, preserving the type of the argument, e.g.:

```
help, abs(-1l), abs([-!PI,0,!PI])
```

```
<Expression>    LONG      =           1
<Expression>    FLOAT     = Array[3]
```

Some of the routines support a /DOUBLE keyword (flag) which enables one to force GDL to perform the calculations in (if applicable) and return the value[s] as double precision floating point numbers regardless of the type of the argument[s] passed, e.g:

```
help, gamma(36b), gamma(36b, /double)
```

```
<Expression>    FLOAT     =          inf
<Expression>    DOUBLE    =      1.0333148e+40
```

Similarily, if a functions returns integer numbers, the /L64 keyword (flag) can be used to force usage of 64-bit integers, e.g.:

```
help, round(1d10), round(1d10, /l64)
```

```
<Expression>    LONG      =  -2147483648
<Expression>    LONG64    =          10000000000
```

If GDL was compiled with OpenMP support (which is the default if the compiler supports it, and most of them do nowadays), and if GDL is run on a multi-cpu (or multi-core) system, and if the array[s] passed as the argument[s] are big enough (see chapter ... TODO) the computations are performed by multiple threads. Consult the individual documentation entries of each of the routines for details.

ABS() returns the absolute value[s] of the real number[s] passed as the argument (integer or floating point) or the magnitude[s] in case of complex number[s]
CEIL() returns the smallest integer number[s] greater than or equal to the argument
FLOOR() returns the greatest integer number[s] less than or equal to the argument (aka the Gauss' symbol)
ROUND() returns an integer value[s] closest to the argument

ERF()
IMSL_ERF()
ERFC()
ERRORF()
EXPINT()
ALOG()
ALOG10()
EXP() (GSL_EXP())

... the following trigonometric functions:

SIN() returns the sine of the argument
ASIN() returns the cosine of the argument
COS()
ACOS()
TAN()
ATAN() ... complex! ...

the following hyperbolic functions:

SINH()
COSH()
TANH()

as well as the following related functions:

LL_ARC_DISTANCE()

BESELI()
BESELJ()
BESELK()
BESELY()

SPHER_HARM()
LAGUERRE()
LEGENDRE()

GAUSSINT() GAUSS_CVF() GAUSS_PDF()

T_PDF()

FACTORIAL() GAMMA() BETA() IGAMMA() LNGAMMA()

PRIMES()

VOIGT()

Linear algebra

LA_TRIRED LUDC SVDC

IDENTITY() REPLICATE() REPLICATE_INPLACE

Statistics

CORRELATE()

HISTOGRAM() HIST_2D() (implemented using HIST_ND())

IMSL_BINOMIALCOEF()

GAUSSINT() GAUSS_CVF() GAUSS_PDF()

T_PDF()

KURTOSIS() SKEWNESS() MEAN() MIN() MAX() MEDIAN() MEANABS-DEV() MOMENT() STDDEV() VARIANCE()

Interpolation

INTERPOL() (implemented using FINDEX()) INTERPOLATE()

REBIN()

DERIV()

SPL_INIT() SPL_INTERP()

VALUE_LOCATE()

Polynomials

IMSL_ZEROPOLY() POLY()

Geometric calculations

POLY_AREA() TRIGRID()

Bitwise operations

ISHFT() BYTEORDER SWAP_ENDIAN() SWAP_ENDIAN_INPLACE

Function fitting

Markwardt [3]

Fourier analysis

FFT() DIST()

Multidimensional root-finding

BROYDEN() IMSL_ZEROPOLY() NEWTON()

Random numbers

RANDOMN() RANDOMU()

Ordinary differential equations

RK4()

Wavelet analysis

WTN()

Mathematical and physical constants

!PI !DPI IDL_CONSTANT()

Chapter 5

Input/output, supported data formats

Basics – accessing files and io streams

PRINT PM GET_KBRD READ

BYTEORDER CLOSE EOF

READ WRITE

READF READS READU

GET_LUN FREE_LUN POINT_LUN SKIP_LUN

OPENR OPENU OPENW

ASCII

PRINTF READF READ_ASCII

CSV

Binary data (raw access)

READ_BINARY()

BYTEORDER SWAP_ENDIAN() SWAP_ENDIAN_INPLACE

FITS

Astron

netCDF

NCDF_ATTCOPY() NCDF_ATTDEL NCDF_ATTGET NCDF_ATTINQ() NCDF_ATTNAME(
NCDF_ATTPUT NCDF_ATTRENAME NCDF_CLOSE NCDF_CONTROL NCDF_CREATE()
NCDF_DIMDEF() NCDF_DIMID() NCDF_DIMINQ NCDF_DIMRENAME NCDF_EXISTS()
NCDF_INQUIRE() NCDF_OPEN() NCDF_VARDEF() NCDF_VARGET NCDF_VARGET1
NCDF_VARID() NCDF_VARINQ() NCDF_VARPUT NCDF_VARRENAME

HDF4

HDF_CLOSE HDF_OPEN()

HDF_SD_ADDDATA HDF_SD_ATTRFIND() HDF_SD_ATTRINFO HDF_SD_CREATE()
HDF_SD_DIMGET HDF_SD_DIMGETID() HDF_SD_END HDF_SD_ENDACCESS
HDF_SD_FILEINFO HDF_SD_GETDATA HDF_SD_GETINFO HDF_SD_NAMETOINDEX()
HDF_SD_SELECT() HDF_SD_START()

HDF_VD_ATTACH() HDF_VD_DETACH HDF_VD_FIND() HDF_VD_GET
HDF_VD_READ()

HDF_VG_ATTACH() HDF_VG_DETACH HDF_VG_GETID() HDF_VG_GETINFO
HDF_VG_GETTRS

HDF5

H5A_CLOSE H5A_GET_NAME() H5A_GET_NUM_ATTRS() H5A_GET_SPACE()
H5A_GET_TYPE() H5A_OPEN_IDX() H5A_OPEN_NAME() H5A_READ()
H5D_CLOSE H5D_GET_SPACE() H5D_GET_TYPE() H5D_OPEN() H5D_READ()
H5F_CLOSE H5F_IS_HDF5() H5F_OPEN() H5G_CLOSE H5G_OPEN() H5S_CLOSE
H5S_GET_SIMPLE_EXTENT_DIMS() H5T_CLOSE H5T_GET_SIZE() H5_GET_LIBVERSIO

raster images (TIFF, PNG, JPEG, . . .)

see chapter in Image Processing

DICOM

GRIB

GRIBAPI_CLONE() GRIBAPI_CLOSE_FILE GRIBAPI_COUNT_IN_FILE() GRIBAPI_GET GRIBAPI_GET_DATA GRIBAPI_GET_SIZE() GRIBAPI_NEW_FROM_FILE() GRIBAPI_OPEN_FILE() GRIBAPI_RELEASE

IDL save files

RESTORE SAVE

Chapter 6

Plotting and mapping

2D plots

AXIS CONTOUR OPLOT PLOT PLOTERR PLOTS POLYFILL XYOUTS

3D plots

SURFACE PLOTS

Plotting raster data

BYTSCL() TV() TVLCT() TVRD() TVSCL()

Managing multiple windows

WDELETE WINDOW WSHOW WSET

Map projections

MAP_CONTINENTS MAP_PROJ_FORWARD MAP_PROJ_INVERSE
LL_ARC_DISTANCE()
MAP_CLIP_SET

Output terminals

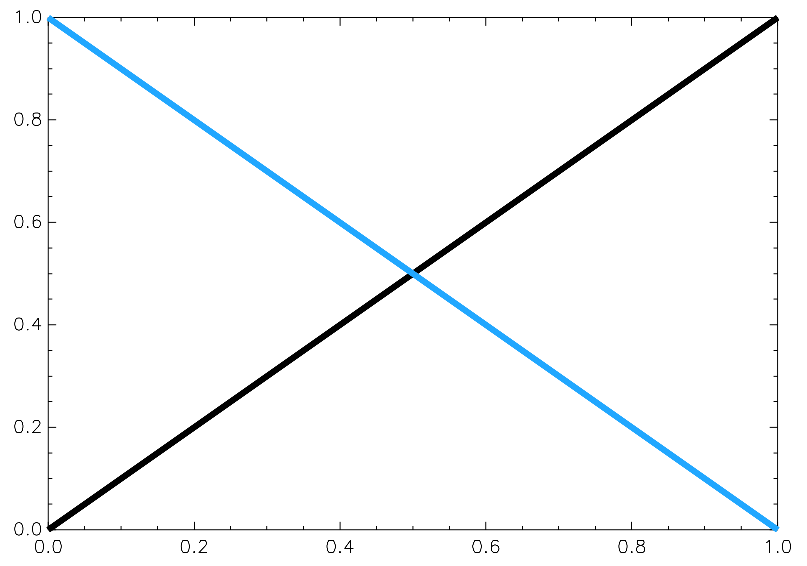
SET_PLOT DEVICE CURSOR ERASE FLUSH

Working with colours

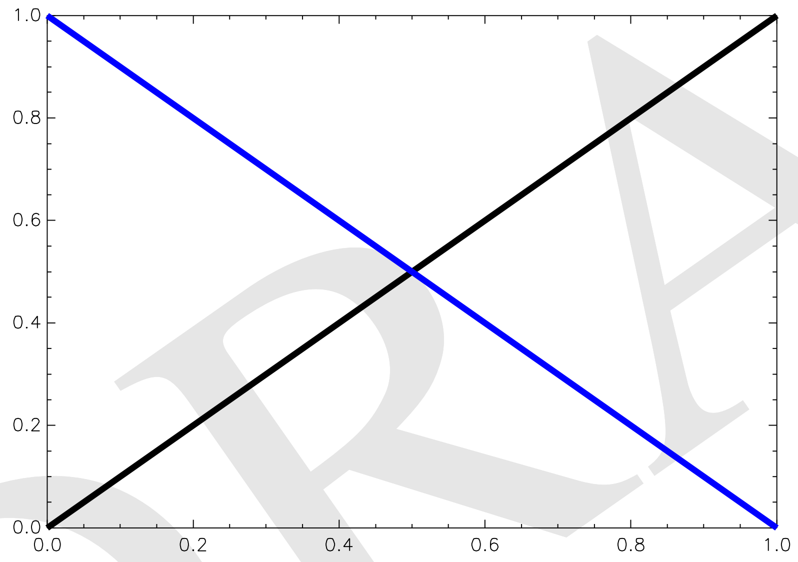
LOADCT

```
device, /color, decomposed=0
loadct, 1
plot, [0,1], thick=20
oplot, [1,0], color=200, thick=20
```

```
% Compiled module: LOADCT.
% LOADCT: Loading table BLUE/WHITE
```



```
device, /color, decomposed=1
plot, [0,1], thick=20
oplot, [1,0], color='ff0000'x, thick=20
```



Fonts, symbols and text formatting

SHOWFONT Harshey fonts [8]

Misc

CONVERT_COORD() GET_SCREEN_SIZE()

Chapter 7

Interaction with host OS

CD POPD PUSHD PRINTD EXIT WAIT

Executing external commands (via shell or not)

SPAWN (while EXECUTE() ...)

Filesystem operations

CD FILE_BASENAME() FILE_COPY FILE_DELETE FILE_DIRNAME() FILE_EXPAND_PATH()
(EXPAND_PATH()) FILE_INFO() FILE_LINES() FILE_MKDIR FILE_SAME()
FILE_SEARCH() FILE_TEST() FILE_WHICH() FINDFILE() FSTAT() PATH_SEP()

Network operations

SOCKET PARSE_URL()

Command-line options and environmental variables

COMMAND_LINE_ARGS() SETENV GETENV() LOCALE_GET()

Chapter 8

Manipulating strings

STRCMP() STRCOMPRESS() STREGEX() STRJOIN() STRLEN()

STRLOWERCASE() STRUPCASE()

STRMID() STRPOS() RSTRPOS() STRPUT() STRSPLIT() STRTOK() STR-TRIM() STR_SEP()

READS()

STRARR() STRING() SINDGEN()

IDL_BASE64() IDL_VALIDANEM() SORT() UNIQ() PARSE_URL()

Chapter 9

Representing date & time

CALDAT CALENDAR SYSTIME()

Chapter 10

Image processing

QUERY_BMP() QUERY_DICOM() QUERY_GIF() QUERY_IMAGE() QUERY_JPEG() QUERY_PICT() QUERY_PNG() QUERY_PPM() QUERY_TIFF()

READ_BMP() READ_DICOM() READ_JPEG READ_PICT READ_PNG() READ_TIFF() READ_XWD()

WRITE_BMP WRITE_JPEG WRITE_PICT WRITE_PNG

BYTSCL() CONVOL() MEDIAN() POLY_2D() PREWITT() RADON() ROBERTS() ROTATE() REBIN() SMOOTH() SOBEL()

Chapter 11

Parallel processing

Built-in features (OpenMP)

CPU

Semaphores and shared memory (library routines)

SEM_CREATE() SEM_DELETE SEM_LOCK() SEM_RELEASE

ImageMagick's features

MPI and GDL

Chapter 12

GUI programming (widgets)

DIALOG_MESSAGE() DIALOG_PICKFILE()

WIDGET_BASE() WIDGET_BUTTON() WIDGET_CONTROL WIDGET_DROPLIST()
WIDGET_EVENT() WIDGET_INFO() WIDGET_LABEL() WIDGET_TEXT()

Chapter 13

Dynamic loading

CALL_EXTERNAL() LINKIMAGE()

Chapter 14

The Python bridge

van Rossum and Fred L. Drake [6]

calling Python code from GDL

PYTHON() PYTHON

calling GDL code from Python

Chapter 15

Alphabetical list of library routines

ABS() function

positional arguments: 1
keywords: none

Returns absolute value of a number passed as the first argument or an array of absolute values if argument is an array. For complex arguments the length of the argument in the complex plane is returned (the phase of a complex number may be obtained using ATAN()).

```
print , abs(-2.2)
print , abs([-1,1,0])
print , abs(.5 * sqrt(2) * complex(1, 1))
```

```
      2.20000
       1         1         0
      1.00000
```

multi-threading: this routine uses GDL thread pool if working on large array, see the...

ACOS() function

positional arguments: 1
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

ALOG() function

positional arguments: 1
keywords: none

ALOG10() function

positional arguments: 1
keywords: none

APPLEMAN procedure

positional arguments: 2
keywords: HELP, NODISPLAY, RESULT, TEST, XSIZE, YSIZE

Computes and optionally renders the Mandelbrot set. The two positional arguments are optional and allow specification of the range over which the set is computed (default values: [-1.0,2.3] and [-1.3,1.3]).

RESULT keyword

Allows passing a variable into which the computed data will be sotred. If set, no rendering is done.

XSIZE keyword

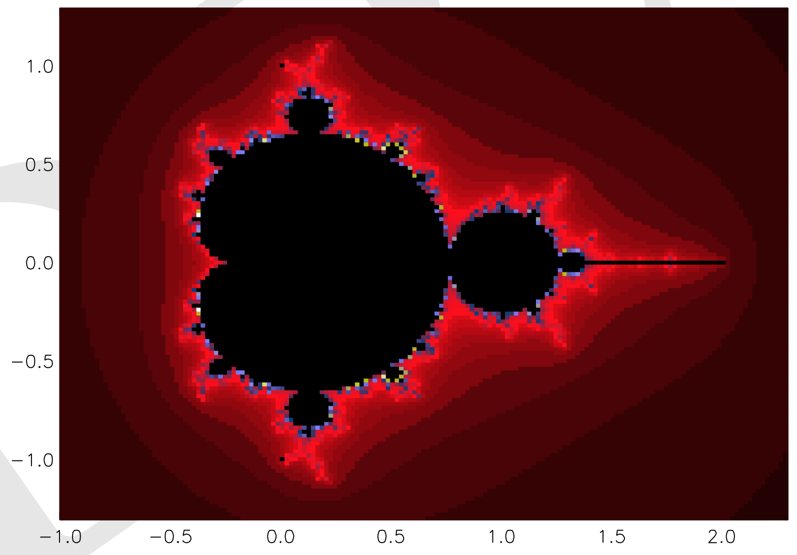
Allows specification of the width of the domain over which the set is computed.

YSIZE keyword

Allows specification of the height of the domain over which the set is computed.

```
rng_x = [-1, 2.3]
rng_y = [-1.3, 1.3]
appleman, rng_x, rng_y, result=fractal, xsize=165, ysize=130
device, /color
plot, [0], /nodata, xrange=rng_x, yrange=rng_y
loadct, 15
tvscl, fractal, rng_x[0], rng_y[0], $
  xsize=rng_x[1]-rng_x[0], ysize=rng_y[1]-rng_y[0]
```

```
% Compiled module: APPLEMAN.
% Compiled module: LOADCT.
% LOADCT: Loading table BOW SPECIAL
% Compiled module: TVSCL.
```



ARG_PRESENT() function

positional arguments: 1

keywords: none

ARRAY_EQUAL() function

positional arguments: 2
keywords: NO_TYPECONV

ARRAY_INDICES() function

positional arguments: 2
keywords: none

see also: WHERE()

ASIN() function

positional arguments: 1
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

ASSOC() function

positional arguments: 3
keywords: PACKED

ATAN() function

positional arguments: 2
keywords: PHASE

multi-threading: this routine uses GDL thread pool if working on large array, see the...

AXIS procedure

positional arguments: 3
keywords: CHARSIZE, CHARTHICK, COLOR, DATA, DEVICE, FONT, NODATA, NOERASE, NORMAL, SAVE, SUBTITLE, T3D, THICK, TICKLEN, XAXIS, XCHARSIZE, XGRIDSTYLE, XLOG, XMARGIN, XMINOR, XRANGE, XSTYLE, XTHICK, XTICKFORMAT, XTICKINTERVAL, XTICKLEN, XTICKNAME, XTICKS, XTITLE, XTYPE, YAXIS, YCHARSIZE, YGRIDSTYLE, YLOG, YMARGIN, YMINOR, YNOZERO, YRANGE, YSTYLE, YTHICK, YTICKFORMAT, YTICKINTERVAL, YTICKLEN, YTICKNAME, YTICKS, YTITLE, YTYPE, ZCHARSIZE, ZGRIDSTYLE, ZMARGIN, ZMINOR, ZRANGE, ZSTYLE, ZTHICK, ZTICKFORMAT, ZTICKLEN, ZTICKNAME, ZTICKS, ZTITLE, ZVALUE

BESELI() function

positional arguments: 2
keywords: DOUBLE, HELP, ITER

BESELJ() function

positional arguments: 2
keywords: DOUBLE, HELP, ITER

BESELK() function

positional arguments: 2
keywords: DOUBLE, HELP, ITER

BESELY() function

positional arguments: 2
keywords: DOUBLE, HELP, ITER

BETA() function

positional arguments: 2
keywords: DOUBLE

BILINEAR() function

positional arguments: 3
keywords: MISSING

BINDGEN() function

positional arguments: 8
keywords: none

BROYDEN() function

positional arguments: 2
keywords: DOUBLE, ITMAX, TOLF, TOLX

BYTARR() function

positional arguments: 8
keywords: NOZERO

BYTE() function

positional arguments: 10
keywords: none

BYTEORDER procedure

positional arguments: any number
keywords: DTOXDR, FTOXDR, HTONL, HTONS, L64SWAP, LSWAP, NTOHL, NTOHS, SSWAP, SWAP_IF_BIG_ENDIAN, SWAP_IF_LITTLE_ENDIAN, XDRTOD, XDRTOF

BYTSCL() function

positional arguments: 3
keywords: MAX, MIN, NAN, TOP

CALDAT procedure

positional arguments: 7
keywords: none

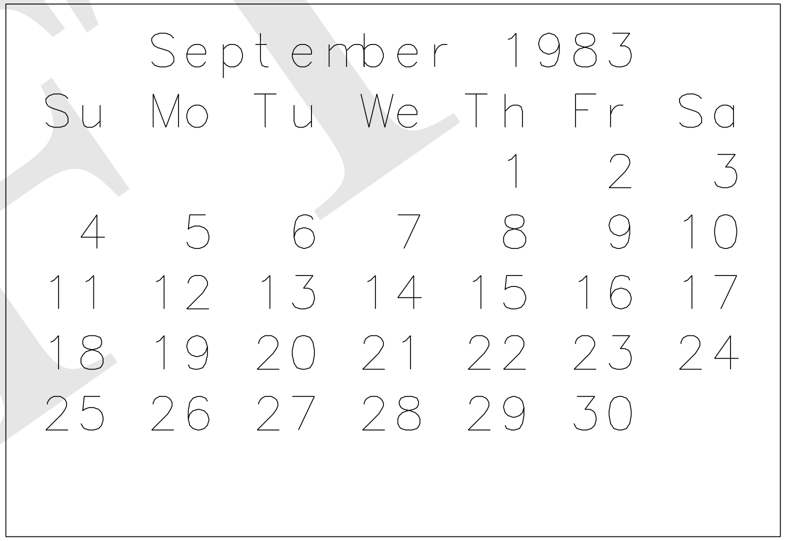
CALENDAR procedure

positional arguments: 2
keywords: none

An interface to the UNIX *cal* command. Displays a calendar using the current graphics device (i.e. X, PS, . . .). The two optional arguments allow to specify a month, or a month and a year.

```
calendar, 9, 1983
```

```
% Compiled module: CALENDAR.
```



```
        September  1983
Su  Mo  Tu  We  Th  Fr  Sa
                 1   2   3
 4   5   6   7   8   9  10
11  12  13  14  15  16  17
18  19  20  21  22  23  24
25  26  27  28  29  30
```

CALL_EXTERNAL() function

positional arguments: any number
keywords: ALL_GDL, ALL_VALUE, B_VALUE, D_VALUE, F_VALUE, I_VALUE, L64_VALUE, L_VALUE, RETURN_TYPE, STRUCT_ALIGN_BYTES, S_VALUE, UI_VALUE, UL64_VALUE, UL_VALUE, UNLOAD, VALUE

Calls a routine from a sharable object library. The first argument should be a string containing the filename of the sharable object to load (standard library paths are searched). The second argument should be a string with the name of the routine in the image to ba called. All subsequent arguments are passed to the routine.

Here is a, hopefully concise, example covering all the steps one could take to write, build and call a C routine from GDL:

```
$ echo '$ cat libexample.c'
$ cat libexample.c
$ echo '$ cat CMakeLists.txt'
$ cat CMakeLists.txt
$ echo '$ cmake .'
$ cmake .|awk '{print (length($0)>50?substr($0,0,50) "...":$0)}'
$ echo
$ echo '$ make'
$ make
$ echo

img = 'libexample.'+(!VERSION.OS_NAME eq 'darwin'?"dylib":"so")
message, '1d308 vs. a next representable double:', /continue
print, format='(E)', 1d308, $
  call_external(img, 'c_nextafter', 1d308, 2d308, /d_value)

$ make clean
```

```
$ cat libexample.c
#include <math.h>
double c_nextafter(int argc, void* argv[]) {
  return nextafter(*(double*)argv[0], *(double*)argv[1]);
}

$ cat CMakeLists.txt
project(libexaple C)
cmake_minimum_required(VERSION 2.0)
add_library(example SHARED libexample.c)
set_directory_properties(PROPERTIES ADDITIONAL_MAKE_CLEAN_FILES
  "Makefile;CMakeCache.txt;cmake_install.cmake;CMakeFiles")

$ cmake .
-- The C compiler identification is GNU
-- Checking whether C compiler has -isysroot
-- Checking whether C compiler has -isysroot - yes
-- Checking whether C compiler supports OSX deploy...
-- Checking whether C compiler supports OSX deploy...
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- w...
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/slayoo...

$ make
Scanning dependencies of target example
[100%] Building C object CMakeFiles/example.dir/libexample.o
Linking C shared library libexample.dylib
[100%] Built target example

% $MAIN$: 1d308 vs. a next representable double:
   1.0000000000000000E+308
   1.0000000000000002E+308
```

RETURN_TYPE keyword

Indicates the type of the return value of the called routine, this value will be returned by CALL_EXTERNAL to GDL. The value of the keyword is interpreted in the same way as the type field of the SIZE() function. Possible values for it are those for numeric types except COMPLEX and DCOMPLEX. The default value is 3 (GDL type LONG, which corresponds to C type int). Alternatively one of the following keywords may be used:

B_VALUE keyword

equivalent to RETURN_TYPE=1 (BYTE)

I_VALUE keyword

equivalent to RETURN_TYPE=2 (INTEGER)

L_VALUE keyword

equivalent to RETURN_TYPE=3 (LONG)
This correspodns to the default behaviour.

F_VALUE keyword

equivalent to RETURN_TYPE=4 (FLOAT)

D_VALUE keyword

equivalent to RETURN_TYPE=5 (DOUBLE)

UI_VALUE keyword

equivalent to RETURN_TYPE=12 (UINT)

UL_VALUE keyword

equivalent to RETURN_TYPE=13 (ULONG)

L64_VALUE keyword

equivalent to RETURN_TYPE=14 (LONG64)

UL64_VALUE keyword

equivalent to RETURN_TYPE=15 (ULONG64)

S_VALUE keyword

equivalent to RETURN_TYPE=6 (STRING, the called function should return char*)

ALL_VALUE keyword

The default is to pass all parameters by reference. If this keyword is set, all parameters are passed by value.

UNLOAD keyword

If set (/UNLOAD or UNLOAD=1) the shared object will be unloaded after calling the routine.

STRUCT_ALIGN_BYTES keyword

If set to an integer n, CALL_EXTERNAL assumes that structures in the shared object are aligned at boundaries of n bytes, where n should be a power of 2. If n=0 or if this keyword is not given, the default machine dependent alignment is assumed (normally 4/8 bytes on 32/64 bit systems). It should only be necessary to use this keyword if the called shared object has been compiled with a different alignment, e.g. with #pragma pack(n)

implementation details: This routine uses the dlopen/dlsym/dlclose calls, and thus is available only on systems that support them. It has been tested on Linux, Apple OS X and Solaris.

see also: LINKIMAGE

disclaimer: CALL_EXTERNAL was implemented in GDL by Christoph Fuchs, who also wrote the documentation for it which was the base for this entry. Copyright: (C) 2010 by Christoph Fuchs. The original file was licensed under GNU GPL v>=2.

CALL_FUNCTION() function

positional arguments: any number
keywords: _REF_EXTRA

CALL_METHOD procedure

positional arguments: any number
keywords: _REF_EXTRA

CALL_METHOD() function

positional arguments: any number
keywords: _REF_EXTRA

CALL_PROCEDURE procedure

positional arguments: any number
keywords: _REF_EXTRA

CATCH procedure

positional arguments: 1
keywords: CANCEL

CD procedure

positional arguments: 1
keywords: CURRENT

CDF_EPOCH procedure

positional arguments: 8
keywords: BREAKDOWN_EPOCH, COMPUTE_EPOCH

CEIL() function

positional arguments: 1
keywords: L64

multi-threading: this routine uses GDL thread pool if working on large array, see the...

CHECK_MATH() function

positional arguments: 2
keywords: MASK, NOCLEAR, PRINT

CINDGEN() function

positional arguments: 8
keywords: none

CLOSE procedure

positional arguments: any number
keywords: ALL, EXIT_STATUS, FILE, FORCE

COMMAND_LINE_ARGS() function

positional arguments: none
keywords: COUNT

COMPLEX() function

positional arguments: 10
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

COMPLEXARR() function

positional arguments: 8
keywords: NOZERO

CONGRID() function

positional arguments: 4
keywords: CENTER, CUBIC, HELP, INTERP, MINUS_ONE, MISSING, TEST

CONJ() function

positional arguments: 1
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

CONTOUR procedure

positional arguments: 3
keywords: BACKGROUND, CHARSIZE, CHARTHICK, CLIP, COLOR, C_CHARSIZE, C_COLORS, C_LINESTYLE, DATA, DEVICE, FILL, FOLLOW, FONT, ISOTROPIC, LEVELS, MAX_VALUE, MIN_VALUE, NLEVELS, NOCLIP, NODATA, NOERASE, NORMAL, OVERPLOT, POSITION, SUBTITLE, T3D, THICK, TICKLEN, TITLE, XCHARSIZE, XGRIDSTYLE, XLOG, XMARGIN, XMINOR, XRANGE, XSTYLE, XTHICK, XTICKFORMAT, XTICKLEN, XTICKNAME, XTICKS, XTICKV, XTICK_GET, XTITLE, XTYPE, YCHARSIZE, YGRIDSTYLE, YLOG, YMARGIN, YMINOR, YRANGE, YSTYLE, YTHICK, YTICKFORMAT, YTICKLEN, YTICKNAME, YTICKS, YTICKV, YTICK_GET, YTITLE, YTYPE, ZCHARSIZE, ZGRIDSTYLE, ZLOG, ZMARGIN, ZMINOR, ZRANGE, ZSTYLE, ZTHICK, ZTICKFORMAT, ZTICKLEN, ZTICKNAME, ZTICKS, ZTICKV, ZTICK_GET, ZTITLE, ZTYPE, ZVALUE

CONVERT_COORD() function

positional arguments: 3

keywords: DATA, DEVICE, DOUBLE, NORMAL, T3D, TO_DATA, TO_DEVICE, TO_NORMAL

CONVOL() function

positional arguments: 3
keywords: CENTER, EDGE_TRUNCATE, EDGE_WRAP

CORRELATE() function

positional arguments: 2
keywords: COVARIANCE, DOUBLE

When called with two vector arguments x and y it returns the correlation coefficient r defined as:

$$r = \frac{\mathrm{cov}(x,y)}{\mathrm{stdev}(x)\cdot\mathrm{stdev}(y)} \tag{15.1}$$

where

$$\mathrm{cov}(x,y) = \frac{1}{N-1}\sum_{i=0}^{N-1}(x[i]-\overline{x})\cdot(y[i]-\overline{y}) \tag{15.2}$$

$$\mathrm{stdev}(x) = \sqrt{\frac{1}{N-1}\sum_{i=0}^{N-1}[x[i]-\overline{x}]^2} \tag{15.3}$$

and

$$\overline{x} = \sum_{i=0}^{N-1}\frac{x[i]}{N} \tag{15.4}$$

(N is the length of the longer vector).

```
print, correlate([-1,0,1], [1,0,-1])
```

```
% Compiled module: CORRELATE.
% Compiled module: MEAN.
     -1.00000
```

DOUBLE keyword

Forces double-precision calculations and output value type.

```
x = [1,   2,   3,   4,   5]
y = [1.1, 1.9, 3.1, 3.9, 5, 6, 7, 8, 9]
help, correlate(x, y)
help, correlate(x, y, /double)
```

```
% Compiled module: CORRELATE.
% Compiled module: MEAN.
<Expression>    FLOAT     =       0.99813
<Expression>    DOUBLE    =        0.9981310
```

COVARIANCE keyword

If called with the COVARIANCE keyword, the covariance $\mathrm{cov}(x,y)$ of the two vectors is returned instead.

```
x = [-1, 0, 1.]
y = [-2, 0, 2.]
print, correlate(x, y, /covariance)
```

```
% Compiled module: CORRELATE.
% Compiled module: MEAN.
      2.00000
```

COS() function

positional arguments: 1
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

COSH() function

positional arguments: 1

keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

CPU procedure

positional arguments: none
keywords: RESET, RESTORE, TPOOL_MAX_ELTS, TPOOL_MIN_ELTS, TPOOL_NTHREADS, VECTOR_ENABLE

CREATE_STRUCT() function

positional arguments: any number
keywords: NAME

CROSSP() function

positional arguments: 2
keywords: none

CURSOR procedure

positional arguments: 3
keywords: CHANGE, DATA, DEVICE, DOWN, NORMAL, NOWAIT, UP, WAIT

DBLARR() function

positional arguments: 8
keywords: NOZERO

DCINDGEN() function

positional arguments: 8
keywords: none

DCOMPLEX() function

positional arguments: 10
keywords: none

DCOMPLEXARR() function

positional arguments: 8
keywords: NOZERO

DEFSYSV procedure

positional arguments: 3
keywords: EXISTS

DERIV() function

positional arguments: 2
keywords: HELP, NO_CHECK, TEST

DETERM() function

positional arguments: 1
keywords: DOUBLE

DEVICE procedure

positional arguments: none
keywords: CLOSE_FILE, COLOR, DECOMPOSED, ENCAPSULATED, FILE-NAME, GET_DECOMPOSED, GET_SCREEN_SIZE, GET_VISUAL_DEPTH, INCHES, LANDSCAPE, PORTRAIT, SCALE_FACTOR, SET_CHARACTER_SIZE, SET_RESOLUTION, WINDOW_STATE, XOFFSET, XSIZE, YOFFSET, YSIZE, Z_BUFFERING

DIALOG_MESSAGE() function

positional arguments: 1
keywords: CANCEL, CENTER, DEFAULT_CANCEL, DEFAULT_NO, DIALOG_PARENT, DISPLAY_NAME, ERROR, HELP, INFORMATION, QUESTION, RESOURCE_NAME, TITLE, ZENITY_NAME, ZENITY_PATH

DIALOG_PICKFILE() function

positional arguments: none
keywords: DEBUG, DEFAULT_EXTENSION, DIALOG_PARENT, DIRECTORY, DISPLAY_NAME, FILE, FILTER, FIX_FILTER, GET_PATH, GROUP, HELP, MULTIPLE_FILES, MUST_EXIST, OVERWRITE_PROMPT, PATH, READ, RESOURCE_NAME, TEST, TITLE, VERBOSE, WRITE, ZENITY_NAME, ZENITY_PATH, ZENITY_SEP

DINDGEN() function

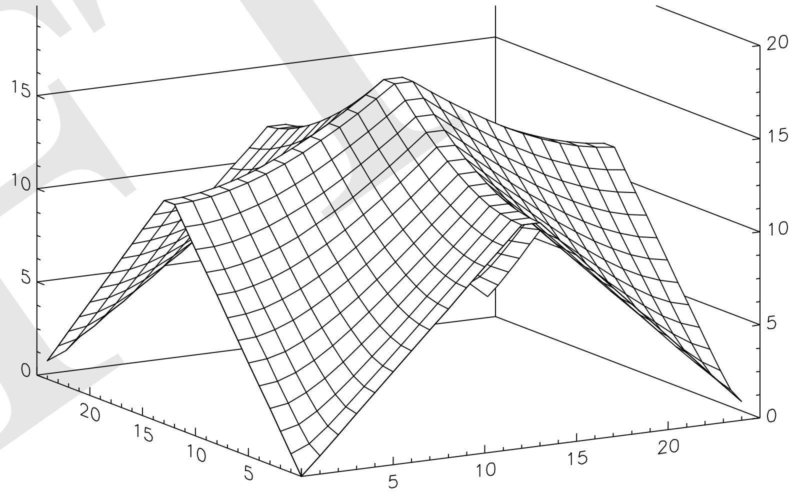
positional arguments: 8
keywords: none

DIST() function

positional arguments: 2
keywords: none

```
surface, dist(25)
```

```
% Compiled module: DIST.
```



DOUBLE() function

positional arguments: 10
keywords: none

EOF() function

positional arguments: 1
keywords: none

ERASE procedure

positional arguments: 1
keywords: none

ERF() function

positional arguments: 1
keywords: DOUBLE

ERFC() function

positional arguments: 1
keywords: DOUBLE

ERRORF() function

positional arguments: 1
keywords: DOUBLE

ESCAPE_SPECIAL_CHAR() function

positional arguments: 1
keywords: HELP, LIST_OF_SPECIAL_CHAR, SHOW_LIST, TEST, VERBOSE

EXECUTE() function

positional arguments: 2
keywords: none

Executes the statement passed in the first arguement, returns 1 if no error occured or 0 if the execution failed, e.g.

```
status = execute('print, "Hello world!"')
help, status
status = execute('print, Hello world!)')
help, status
```

```
Hello world!
STATUS          INT       =        1
% Parser syntax error: unexpected token: HELLO
STATUS          INT       =        0
```

EXIT procedure

positional arguments: none
keywords: NO_CONFIRM, STATUS

STATUS keyword

```
spawn, '../../../../src/gdl -quiet -e "exit, status=44" 1>/dev/null', $
  exit_status=s
print, 'spawned GDL process exited with code ', strtrim(s, 2)
```

```
spawned GDL process exited with code 44
```

EXP() function

positional arguments: 1
keywords: none

```
print, exp([0, 1, -!VALUES.F_INFINITY])
print, alog(exp([!PI]))
```

```
      1.00000       2.71828       0.00000
      3.14159
```

multi-threading: this routine uses GDL thread pool if working on large array, see the...

EXPAND_PATH() function

positional arguments: 1
keywords: ALL_DIRS, ARRAY, COUNT

EXPINT() function

positional arguments: 2
keywords: DOUBLE

FACTORIAL() function

positional arguments: 1
keywords: STIRLING, UL64

FFT() function

positional arguments: 2
keywords: DIMENSION, DOUBLE, INVERSE, OVERWRITE

$$F[m] = \frac{1}{N}\sum_k f[k] \cdot e^{-\frac{2\pi i}{N}mk} \tag{15.5}$$

```
$ tail stddev*.pro
x = [1.31, 2.44, 2.51, 3.01, 2.96, 2.50, 0.05, 3.24, 0.13]
print, stddevsum(x), stddevfft(x)
```

```
==> stddevfft.pro <==
function stddevfft, x
  return, sqrt(total((abs(fft(x))^2)[1:-1]))
end

==> stddevsum.pro <==
function stddevsum, x
  return, sqrt(mean(x^2) - mean(x)^2)
end
% Compiled module: STDDEVSUM.
% Compiled module: MEAN.
% Compiled module: STDDEVFFT.
      1.15258       1.15258
```

multi-threading: this routine uses GDL thread pool if working on large array, see the...

implementation details: FFTW vs. GSL - TODO

FILEPATH() function

positional arguments: 1
keywords: ROOT_DIR, SUBDIRECTORY, TERMINAL, TMP

FILE_BASENAME() function

positional arguments: 2
keywords: FOLD_CASE, HELP

```
print, file_basename('/etc/passwd')
```

```
% Compiled module: FILE_BASENAME.
% Compiled module: ESCAPE_SPECIAL_CHAR.
% Compiled module: STRSPLIT.
% Compiled module: UNIQ.
passwd
```

```
print, file_basename('/etc/resolv.conf', '.conf')
```

```
% Compiled module: FILE_BASENAME.
% Compiled module: ESCAPE_SPECIAL_CHAR.
% Compiled module: STRSPLIT.
% Compiled module: UNIQ.
resolv
```

```
print, file_basename(file_search('../../../../src/gdl*.g'))
```

```
% Compiled module: FILE_BASENAME.
% Compiled module: ESCAPE_SPECIAL_CHAR.
% Compiled module: STRSPLIT.
% Compiled module: UNIQ.
gdlc.g gdlc.i.g gdlc.tree.g
```

see also: FILE_DIRNAME(), PATH_SEP()

FILE_COPY procedure

positional arguments: 2
keywords: ALLOW_SAME, HELP, NOEXPAND_PATH, OVERWRITE, QUIET, RECURSIVE, REQUIRE_DIRECTORY, TEST, VERBOSE

FILE_DELETE procedure

positional arguments: 30
keywords: ALLOW_NONEXISTENT, HELP, NOEXPAND_PATH, QUIET, RECURSIVE, TEST, VERBOSE

FILE_DIRNAME() function

positional arguments: 1
keywords: HELP, MARK_DIRECTORY

FILE_EXPAND_PATH() function

positional arguments: 1
keywords: none

FILE_INFO() function

positional arguments: 2
keywords: NOEXPAND_PATH

FILE_LINES() function

positional arguments: 1
keywords: COMPRESS, NOEXPAND_PATH

```
print, file_lines('../../../../ChangeLog')
```

```
% Compiled module: FILE_LINES.
        6335
```

FILE_MKDIR procedure

positional arguments: any number
keywords: NOEXPAND_PATH

implementation details: Current implementation uses the system() call and executes the mkdir using using a shell subprocess

FILE_SAME() function

positional arguments: 2
keywords: NOEXPAND_PATH

FILE_SEARCH() function

positional arguments: 2
keywords: COUNT, EXPAND_ENVIRONMENT, EXPAND_TILDE, FOLD_CASE, FULLY_QUALIFY_PATH, ISSUE_ACCESS_ERROR, MARK_DIRECTORY, MATCH_ALL_INITIAL_DOT, MATCH_INITIAL_DOT, NOSORT, QUOTE

FILE_TEST() function

positional arguments: 1
keywords: BLOCK_SPECIAL, CHARACTER_SPECIAL, DIRECTORY, EXECUTABLE, GET_MODE, NAMED_PIPE, NOEXPAND_PATH, READ, REGULAR, SOCKET, SYMLINK, WRITE, ZERO_LENGTH

FILE_WHICH() function

positional arguments: 2
keywords: DEBUG, HELP, INCLUDE_CURRENT_DIR, TEST

FINDEX() function

positional arguments: 2
keywords: none

FINDFILE() function

positional arguments: 1
keywords: COUNT, HELP, QUIET, SH_LOCATION, SPAWN_OPTIONS, TEST, VERBOSE

FINDGEN() function

positional arguments: 8
keywords: none

FINITE() function

positional arguments: 1
keywords: INFINITY, NAN

FIX() function

positional arguments: 10
keywords: PRINT, TYPE

FLOAT() function

positional arguments: 10
keywords: none

FLOOR() function

positional arguments: 1
keywords: L64

multi-threading: this routine uses GDL thread pool if working on large array, see the...

FLTARR() function

positional arguments: 8
keywords: NOZERO

FLUSH procedure

positional arguments: any number
keywords: none

FREE_LUN procedure

positional arguments: any number
keywords: EXIT_STATUS, FORCE

FSTAT() function

positional arguments: 1
keywords: none

GAMMA() function

positional arguments: 1
keywords: DOUBLE

GAUSSINT() function

positional arguments: 1
keywords: DOUBLE

GAUSS_CVF() function

positional arguments: 1
keywords: none

GAUSS_PDF() function

positional arguments: 1
keywords: none

GDL_ERFINV() function

positional arguments: 1
keywords: DOUBLE

GETENV() function

positional arguments: 1
keywords: ENVIRONMENT

GET_DRIVE_LIST() function

positional arguments: none
keywords: COUNT

GET_KBRD() function

positional arguments: 1
keywords: none

GET_LOGIN_INFO() function

positional arguments: none
keywords: none

Returns a structure with current username and hostname:

```
help, get_login_info(), /structure
```

```
** Structure <Anonymous>, 2 tags, data length=16:
   MACHINE_NAME      STRING      'eyrie.prac.igf'
   USER_NAME         STRING      'slayoo'
```

GET_LUN procedure

positional arguments: 1
keywords: none

GET_SCREEN_SIZE() function

positional arguments: 1
keywords: RESOLUTION

GRIBAPI_CLONE() function

positional arguments: 1
keywords: none

GRIBAPI_CLOSE_FILE procedure

positional arguments: 1
keywords: none

GRIBAPI_COUNT_IN_FILE() function

positional arguments: 1
keywords: none

GRIBAPI_GET procedure

positional arguments: 3
keywords: none

GRIBAPI_GET_DATA procedure

positional arguments: 4
keywords: none

GRIBAPI_GET_SIZE() function

positional arguments: 2
keywords: none

GRIBAPI_NEW_FROM_FILE() function

positional arguments: 1
keywords: none

GRIBAPI_OPEN_FILE() function

positional arguments: 1
keywords: none

GRIBAPI_RELEASE procedure

positional arguments: 1
keywords: none

GSL_EXP() function

positional arguments: 1
keywords: none

H5A_CLOSE procedure

positional arguments: 1
keywords: none

H5A_GET_NAME() function

positional arguments: 1
keywords: none

H5A_GET_NUM_ATTRS() function

positional arguments: 1
keywords: none

H5A_GET_SPACE() function

positional arguments: 1
keywords: none

H5A_GET_TYPE() function

positional arguments: 1
keywords: none

H5A_OPEN_IDX() function

positional arguments: 2
keywords: none

H5A_OPEN_NAME() function

positional arguments: 2
keywords: none

H5A_READ() function

positional arguments: 1
keywords: none

H5D_CLOSE procedure

positional arguments: 1
keywords: none

H5D_GET_SPACE() function

positional arguments: 1
keywords: none

H5D_GET_TYPE() function

positional arguments: 1
keywords: none

H5D_OPEN() function

positional arguments: 2
keywords: none

H5D_READ() function

positional arguments: 1
keywords: none

H5F_CLOSE procedure

positional arguments: 1
keywords: none

H5F_IS_HDF5() function

positional arguments: 1
keywords: none

H5F_OPEN() function

positional arguments: 1
keywords: none

H5G_CLOSE procedure

positional arguments: 1
keywords: none

H5G_OPEN() function

positional arguments: 2
keywords: none

H5S_CLOSE procedure

positional arguments: 1
keywords: none

H5S_GET_SIMPLE_EXTENT_DIMS() function

positional arguments: 1
keywords: none

H5T_CLOSE procedure

positional arguments: 1
keywords: none

H5T_GET_SIZE() function

positional arguments: 1
keywords: none

H5_GET_LIBVERSION() function

positional arguments: none
keywords: none

Returns a string containing the version number of the HDF5 library.

```
help, h5_get_libversion()
```

```
<Expression>    STRING    = '1.8.8'
```

HDF_CLOSE procedure

positional arguments: 1
keywords: none

HDF_OPEN() function

positional arguments: 2
keywords: ALL, CREATE, NUM_DD, RDWR, READ, WRITE

HDF_SD_ADDDATA procedure

positional arguments: 2
keywords: COUNT, START, STRIDE

HDF_SD_ATTRFIND() function

positional arguments: 2
keywords: none

HDF_SD_ATTRINFO procedure

positional arguments: 2
keywords: COUNT, DATA, HDF_TYPE, NAME, TYPE

HDF_SD_CREATE() function

positional arguments: 3
keywords: BYTE, DFNT_CHAR, DFNT_FLOAT32, DFNT_FLOAT64, DFNT_INT16, DFNT_INT32, DFNT_INT8, DFNT_UINT16, DFNT_UINT32, DFNT_UINT8, DOUBLE, FLOAT, HDF_TYPE, INT, LONG, SHORT, STRING

HDF_SD_DIMGET procedure

positional arguments: 1
keywords: COUNT, NAME, NATTR, SCALE

HDF_SD_DIMGETID() function

positional arguments: 2
keywords: none

HDF_SD_END procedure

positional arguments: 1
keywords: none

HDF_SD_ENDACCESS procedure

positional arguments: 1
keywords: none

HDF_SD_FILEINFO procedure

positional arguments: 3
keywords: none

HDF_SD_GETDATA procedure

positional arguments: 2
keywords: COUNT, START, STRIDE

HDF_SD_GETINFO procedure

positional arguments: 1
keywords: COORDSYS, DIMS, FORMAT, HDF_TYPE, LABEL, NAME, NATTS, NDIMS, TYPE, UNIT

HDF_SD_NAMETOINDEX() function

positional arguments: 2
keywords: none

HDF_SD_SELECT() function

positional arguments: 2
keywords: none

HDF_SD_START() function

positional arguments: 2
keywords: CREATE, RDWR, READ

HDF_VD_ATTACH() function

positional arguments: 2
keywords: READ, WRITE

HDF_VD_DETACH procedure

positional arguments: 1
keywords: none

HDF_VD_FIND() function

positional arguments: 2
keywords: none

HDF_VD_GET procedure

positional arguments: 1
keywords: CLASS, COUNT, NAME, REF, TAG

HDF_VD_READ() function

positional arguments: 2
keywords: FIELDS, FULL_INTERLACE, NO_INTERLACE, NRECORDS

HDF_VG_ATTACH() function

positional arguments: 2
keywords: READ, WRITE

HDF_VG_DETACH procedure

positional arguments: 1
keywords: none

HDF_VG_GETID() function

positional arguments: 2
keywords: none

HDF_VG_GETINFO procedure

positional arguments: 1
keywords: CLASS, NAME, NENTRIES, REF, TAG

HDF_VG_GETTRS procedure

positional arguments: 3
keywords: none

HEAP_GC procedure

positional arguments: none
keywords: OBJ, PTR, VERBOSE

HELP procedure

positional arguments: any number
keywords: BRIEF, CALLS, FUNCTIONS, INFO, LIB, MEMORY, OUTPUT, PROCEDURES, RECALL_COMMANDS, ROUTINES, STRUCTURES

HELPFORM() function

positional arguments: 2
keywords: FULL_STRUCT, SHORTFORM, SINGLE, SIZE, STRUCTURE_NAME, TAGFORM, WIDTH

HISTOGRAM() function

positional arguments: 1
keywords: BINSIZE, INPUT, LOCATIONS, MAX, MIN, NAN, NBINS, OMAX, OMIN, REVERSE_INDICES

HIST_2D() function

positional arguments: 2
keywords: BIN1, BIN2, MAX1, MAX2, MIN1, MIN2

implementation details: this routine is implemented as a wrapper to the HIST_ND() function

HIST_ND() function

positional arguments: 2
keywords: MAX, MIN, NBINS, REVERSE_INDICES

Performs an N-dimensional histogram, also known as the joint density function of N variables.

The first argument is an N×P array representing P data points in N dimensions. The second argument is optional, and it may be used to specify the size of the bin to use. Either an N point vector specifying a separate size for each dimension, or a scalar, which will be used for all dimensions. If BINSIZE is not passed, the NBINS keyword must be set (see below).

The function returns the N-Dimensional histogram, an array of size N1×N2×N3×...×ND where the Ni's are the number of bins implied by the data, and/or the optional inputs (see below).

MIN keyword

The minimum value for the histogram. Either a P point vector specifying a separate minimum for each dimension, or a scalar, which will be used for all dimensions. If omitted, the natural minimum within the dataset will be used.

MAX keyword

The maximum value for the histogram. Either a P point vector specifying a separate maximum for each dimension, or a scalar, which will be used for all dimensions. If omitted, the natural maximum within the dataset will be used.

NBINS keyword

Rather than specifying the binsize, you can pass NBINS, the number of bins in each dimension, which can be a P point vector, or a scalar. If BINSIZE it also passed, NBINS will be ignored, otherwise BINSIZE will then be calculated as binsize=(max-min)/nbins.

REVERSE_INDICES keyword

Set to a named variable to receive the reverse indices, for mapping which points occurred in a given bin. Note that this is a 1-dimensional reverse index vector (see HISTOGRAM()). E.g., to find the indices of points which fell in a histogram bin [i,j,k], look up:

```
ind=[i+nx*(j+ny*k)]
ri[ri[ind]:ri[ind+1]-1]
```

See also ARRAY_INDICES() for converting in the other direction.

see also: HISTOGRAM(), HIST_2D()

disclaimer: Entry based on J.D. Smith's documentation for his implementation of HIST_ND which was included in GDL unchanged. Copyright (C) 2001-2007, J.D Smith. This software is provided as is without any warranty whatsoever. Permission to use, copy, modify, and distribute modified or unmodified copies is granted, provided this copyright and disclaimer are included unchanged.

IDENTITY() function

positional arguments: 1
keywords: DOUBLE

IDL_BASE64() function

positional arguments: 1
keywords: none

disclaimer: the name of this GDL routine includes the **IDL_** prefix for compatibility with IDL, it has no ...

IDL_VALIDNAME() function

positional arguments: 1
keywords: CONVERT_ALL, CONVERT_SPACES, HELP, TEST

IGAMMA() function

positional arguments: 2
keywords: DOUBLE

IMAGE_STATISTICS procedure

positional arguments: 1
keywords: COUNT, DATA_SUM, HELP, LUT, MASK, MAXIMUM, MEAN, MINIMUM, STDDEV, SUM_OF_SQUARES, TEST, VARIANCE, VECTOR, VERBOSE, WEIGHTED, WEIGHT_SUM

IMAGINARY() function

positional arguments: 1
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

IMSL_BINOMIALCOEF() function

positional arguments: 2
keywords: DOUBLE

Returns the binomial coefficient defined as:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \text{ for } 0 \le k \le n \tag{15.6}$$

where n and k are the first and second arguments, respectively.

The routine can be used for example to construct the Pascal's trangle:

```
$ cat pascal.pro
pascal, 8
```

```
pro pascal, n
  tri = replicate('   ', 2 * n - 1, n)
  for i=0, n-1 do for j=0, i do tri[2*j + (n-i)-1, i] = $
    string(imsl_binomialcoef(i, j), f='(I3)')
  print, tri
end
% Compiled module: PASCAL.
                               1
                           1       1
                       1       2       1
                   1       3       3       1
               1       4       6       4       1
           1       5      10      10       5       1
       1       6      15      20      15       6       1
   1       7      21      35      35      21       7       1
```

DOUBLE keyword

Forces double precision:

```
help, imsl_binomialcoef(1000, 20)
help, imsl_binomialcoef(1000, 20, /double)
```

```
<Expression>    FLOAT     =          inf
<Expression>    DOUBLE    =    3.3948281e+41
```

implementation details: this routine is a wrapper to the GSL's gsl_sf_choose() function [2]

disclaimer: the name of this GDL routine includes the **IMSL_** prefix for compatibility with IDL, it has no ...

IMSL_CONSTANT() function

positional arguments: 2
keywords: DOUBLE

```
print, 'Unified atomic mass, amu. [kg]:                ', $
  imsl_constant('amu')
print, 'Pressure of 1 standard atmosphere [Pa]:        ', $
  imsl_constant('atm')
print, '  -||-                                         ', $
  imsl_constant('StandardPressure')
print, 'Astronomical unit [m]:                         ', $
  imsl_constant('AU')
print, "Avogadro's number [1/mole]:                    ", $
  imsl_constant('Avogadro')
print, 'Boltzmann constant [J/K]:                      ', $
  imsl_constant('Boltzman')
print, 'Speed of light in vacuum [m/s]:                ', $
  imsl_constant('C')
print, '  -||-                                         ', $
  imsl_constant('Speedlight')
print, 'Base of the natural logarithm [1]:             ', $
  imsl_constant('E')
print, 'Charge of the electron [C]:                    ', $
  imsl_constant('ElectronCharge')
print, 'Mass of the electron [kg]:                     ', $
  imsl_constant('ElectronMass')
print, 'The energy of 1 electron volt, eV [J]:         ', $
  imsl_constant('ElectronVolt')
print, 'Euler-Mascheroni (gamma) constant [1]:         ', $
  imsl_constant('Euler')
print, '  -||-                                         ', $
  imsl_constant('Gamma')
print, 'Molar charge of 1 Faraday [C/mole]:            ', $
  imsl_constant('Faraday')
print, 'Electromagnetic fine structure constant [1]:', $
  imsl_constant('FineStructure')
print, 'The molar gas constant [J/mole/K]:             ', $
  imsl_constant('Gas')
print, 'The gravitational constant [N*m2/kg2]:         ', $
  imsl_constant('Gravity')
print, "Planck's constant divided by 2 pi [J*s]:       ", $
  imsl_constant('Hbar')
print, 'The standard gas volume [m3 / mole]:           ', $
  imsl_constant('PerfectGasVolume')
print, 'Pi [1]:                                        ', $
  imsl_constant('Pi')
print, "Planck's constant [J*s]:                       ", $
  imsl_constant('Planck')
print, 'Mass of the proton [kg]:                       ', $
  imsl_constant('ProtonMass')
print, "Rydberg's constant [1/m]:                      ", $
  imsl_constant('Rydberg')
print, 'Standard gravitational acc. on Earth [m/s2]:', $
  imsl_constant('StandardGravity')
print, 'Stefan-Boltzmann radiation const. [W/K4/m2]:', $
  imsl_constant('StefanBoltzman')
print, 'Triple point temperature for water [K]:        ', $
  imsl_constant('WaterTriple')
```

```
Unified atomic mass, amu. [kg]:                1.66054e-27
Pressure of 1 standard atmosphere [Pa]:            101325.
  -||-                                             101325.
Astronomical unit [m]:                         1.49598e+11
Avogadro's number [1/mole]:                    6.02214e+23
```

```
Boltzmann constant [J/K]:                              1.38065e−23
Speed of light in vacuum [m/s]:                        2.99792e+08
  −||−                                                 2.99792e+08
Base of the natural logarithm [1]:                        2.71828
Charge of the electron [C]:                            1.60218e−19
Mass of the electron [kg]:                             9.10938e−31
The energy of 1 electron volt, eV [J]:                 1.60218e−19
Euler−Mascheroni (gamma) constant [1]:                    0.57722
  −||−                                                    0.57722
Molar charge of 1 Faraday [C/mole]:                       96485.3
Electromagnetic fine structure constant [1]:              0.00730
The molar gas constant [J/mole/K]:                        8.31447
The gravitational constant [N∗m2/kg2]:                 6.67300e−11
Planck's constant divided by 2 pi [J∗s]:               1.05457e−34
The standard gas volume [m3 / mole]:                      0.02271
Pi [1]:                                                   3.14159
Planck's constant [J∗s]:                               6.62607e−34
Mass of the proton [kg]:                               1.67262e−27
Rydberg's constant [1/m]:                              1.09737e+07
Standard gravitational acc. on Earth [m/s2]:              9.80665
Stefan−Boltzmann radiation const. [W/K4/m2]:           5.67040e−08
Triple point temperature for water [K]:                   273.160
```

implementation details: this routine uses the GSL's constants catalogue [2], the unit conversion is implemented using the UDUNITS-2 library

disclaimer: the name of this GDL routine includes the **IMSL_** prefix for compatibility with IDL, it has no ...

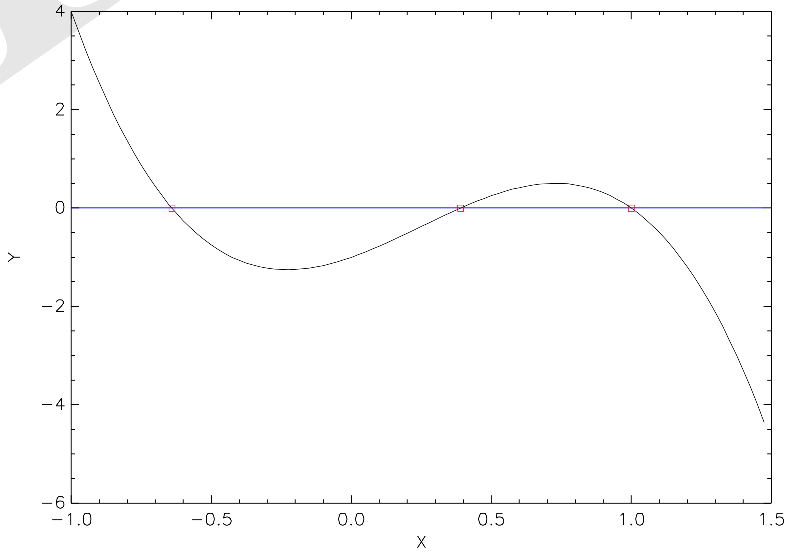
IMSL_ERF() function

positional arguments: 1
keywords: DOUBLE, INVERSE

IMSL_ZEROPOLY() function

positional arguments: 1
keywords: COMPANION, DOUBLE, JENKINS_TRAUB

```
c = [−1,2,3,−4]
x = −1 + findgen(100) / 40
device, /color, /decomposed
plot, x, c[0] + c[1] ∗ x + c[2] ∗ x^2 + c[3] ∗ x^3, $
  xtitle='X', ytitle='Y', thick=3
oplot, x, replicate(0,n_elements(x)), color='ff0000'x
foreach z, imsl_zeropoly(c) do $
  plots, z, 0., psym=6, thick=3, color='0000ff'x
```



implementation details: this routine is a wrapper to the GSL's gsl_poly_complex_solve() function [2]

disclaimer: the name of this GDL routine includes the **IMSL_** prefix for compatibility with IDL, it has no ...

IMSL_ZEROSYS() function

positional arguments: 2

keywords: DOUBLE, ERR_REL, FNORM, ITMAX, JACOBIAN, XGUESS

INDGEN() function

positional arguments: 8
keywords: BYTE, COMPLEX, DCOMPLEX, DOUBLE, FLOAT, L64, LONG, STRING, TYPE, UINT, UL64, ULONG

INTARR() function

positional arguments: 8
keywords: NOZERO

INTERPOL() function

positional arguments: 3
keywords: LSQUADRATIC, QUADRATIC, SPLINE

INTERPOLATE() function

positional arguments: 4
keywords: CUBIC, GRID, MISSING

INVERT() function

positional arguments: 2
keywords: DOUBLE

ISHFT() function

positional arguments: 2
keywords: _EXTRA

JOURNAL procedure

positional arguments: 1
keywords: none

KEYWORD_SET() function

positional arguments: 1
keywords: none

KURTOSIS() function

positional arguments: 1
keywords: DOUBLE, NAN

L64INDGEN() function

positional arguments: 8
keywords: none

LAGUERRE() function

positional arguments: 3
keywords: COEFFICIENTS, DOUBLE

LAST_ITEM() function

positional arguments: 1
keywords: none

LA_TRIRED procedure

positional arguments: 3
keywords: DOUBLE, UPPER

LEGENDRE() function

positional arguments: 3
keywords: DOUBLE

LINDGEN() function

positional arguments: 8
keywords: none

LINKIMAGE procedure

positional arguments: 4
keywords: none

see also: CALL_EXTERNAL()

LL_ARC_DISTANCE() function

positional arguments: 3
keywords: DEGREES

Snyder [eqs. 5-5 and 5-6 in 5]

LMGR() function

positional arguments: none
keywords: CLIENTSERVER, DEMO, EMBEDDED, EXPIRE_DATE, FORCE_DEMO, INSTALL_NUM, LMHOSTID, RUNTIME, SITE_NOTICE, STUDENT, TRIAL, VM

LNGAMMA() function

positional arguments: 1
keywords: DOUBLE

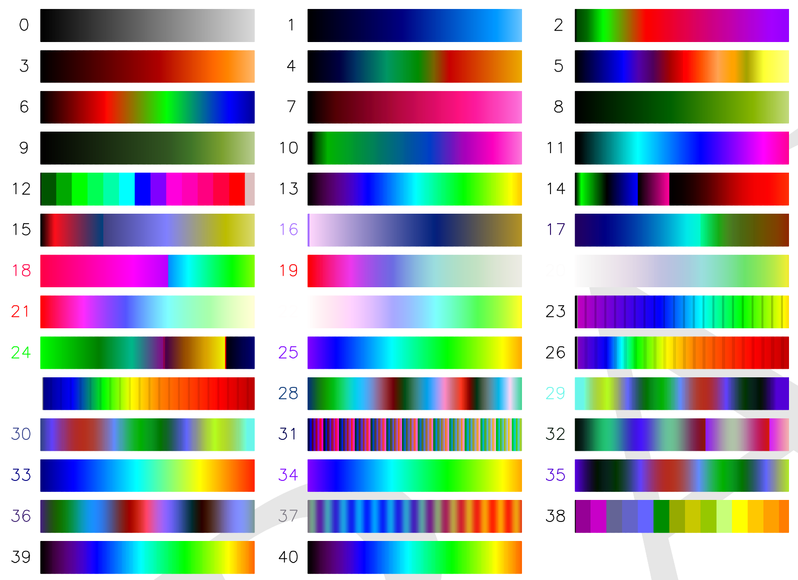
LOADCT procedure

positional arguments: 1
keywords: BOTTOM, FILE, GET_NAMES, NCOLORS, SILENT

Loads a colour table that defines the RGB values corresponding to given colour indices (used when a plotting terminal is not set to the decomposed mode). The first argument may be used to chose from one of the 41 predefined colour tables, see example below for a graphical list of the colour predefined tables.

```
$ cat listct.pro
listct
```

```
pro listct
  !X.STYLE=5
  !Y.STYLE=5
  !P.MULTI=[0,3,14]
  !X.MARGIN=[10,0]
  !Y.MARGIN=[1,0]
  device, /color
  for i=0, 40 do begin
    loadct, i, /silent
    contour, [[indgen(255)],[indgen(255)]], nlevels=256, /fill
    xyouts, -77, .5, strmid(i, 2)
  endfor
end
% Compiled module: LISTCT.
% Compiled module: LOADCT.
```



GET_NAMES keyword

When set to a variable, a list of colour table names (string array) is assigned to that variable.

```
loadct , get_names=names
for i=0, n_elements(names)-1 do $
  print , i , names[i], format='(%"%d: %s")'
```

```
% Compiled module: LOADCT.
0: B-W LINEAR
1: BLUE/WHITE
2: GRN-RED-BLU-WHT
3: RED TEMPERATURE
4: BLUE/GREEN/RED/YELLOW
5: STD GAMMA-II
6: PRISM
7: RED-PURPLE
8: GREEN/WHITE LINEAR
9: GRN/WHT EXPONENTIAL
10: GREEN-PINK
11: BLUE-RED
12: 16 LEVEL
13: RAINBOW
14: STEPS
15: BOW SPECIAL
16: Haze
17: Blue - Pastel - Red
18: Pastels
19: Hue Sat Lightness 1
20: Hue Sat Lightness 2
21: Hue Sat Value 1
22: Hue Sat Value 2
23: Purple-Red + Stripes
24: Beach
25: Mac Style
26: Eos A
27: Eos B
28: Hardcandy
29: Nature
30: Ocean
31: Peppermint
32: Plasma
33: Blue-Red
34: Rainbow
35: Blue Waves
36: Volcano
37: Waves
38: Rainbow18
39: Rainbow + white
40: Rainbow + black
```

LOADCT_INTERNALGDL procedure

positional arguments: 1
keywords: GET_NAMES

LOCALE_GET() function

positional arguments: none

keywords: none

LOGICAL_AND() function

positional arguments: 2
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

LOGICAL_OR() function

positional arguments: 2
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

LOGICAL_TRUE() function

positional arguments: 1
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

LON64ARR() function

positional arguments: 8
keywords: NOZERO

LONARR() function

positional arguments: 8
keywords: NOZERO

LONG() function

positional arguments: 10
keywords: none

LONG64() function

positional arguments: 10
keywords: none

LUDC procedure

positional arguments: 2
keywords: COLUMN, DOUBLE, INTERCHANGES

LUSOL() function

positional arguments: 3
keywords: COLUMN, DOUBLE

MACHAR() function

positional arguments: none
keywords: DOUBLE

MAGICK_ADDNOISE procedure

positional arguments: 1
keywords: GAUSSIANNOISE, IMPULSENOISE, LAPLACIANNOISE, MULTIPLICATIVE-GAUSSIANNOISE, NOISE, POISSONNOISE, UNIFORMNOISE

MAGICK_CLOSE procedure

positional arguments: 1
keywords: none

MAGICK_COLORMAPSIZE() function

positional arguments: 2
keywords: none

MAGICK_COLUMNS() function

positional arguments: 1
keywords: none

MAGICK_CREATE() function

positional arguments: 3
keywords: none

MAGICK_DISPLAY procedure

positional arguments: 1
keywords: none

MAGICK_EXISTS() function

positional arguments: none
keywords: none

MAGICK_FLIP procedure

positional arguments: 1
keywords: none

MAGICK_INDEXEDCOLOR() function

positional arguments: 1
keywords: none

MAGICK_INTERLACE procedure

positional arguments: 1
keywords: LINEINTERLACE, NOINTERLACE, PLANEINTERLACE

MAGICK_MAGICK() function

positional arguments: 2
keywords: none

MAGICK_MATTE procedure

positional arguments: 1
keywords: none

MAGICK_OPEN() function

positional arguments: 1
keywords: none

MAGICK_PING() function

positional arguments: 2
keywords: CHANNELS, DIMENSIONS, GAUSSIANNOISE, HAS_PALETTE, IMAGE_INDEX, IMPULSENOISE, INFO, LAPLACIANNOISE, MULTIPLICATIVEGAUSSIANNOISE, NOISE, NUM_IMAGES, PIXEL_TYPE, POISSONNOISE, TYPE, UNIFORMNOISE

MAGICK_QUALITY procedure

positional arguments: 2
keywords: none

MAGICK_QUANTIZE procedure

positional arguments: 2
keywords: DITHER, GRAYSCALE, TRUECOLOR, YUV

MAGICK_READ() function

positional arguments: 1
keywords: MAP, RGB, SUB_RECT

MAGICK_READCOLORMAPRGB procedure

positional arguments: 4
keywords: none

MAGICK_READINDEXES() function

positional arguments: 1
keywords: none

MAGICK_ROWS() function

positional arguments: 1
keywords: none

MAGICK_WRITE procedure

positional arguments: 2
keywords: RGB

MAGICK_WRITECOLORTABLE procedure

positional arguments: 4
keywords: none

MAGICK_WRITEFILE procedure

positional arguments: 3
keywords: none

MAGICK_WRITEINDEXES procedure

positional arguments: 2
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

MAKE_ARRAY() function

positional arguments: 8
keywords: BYTE, COMPLEX, DCOMPLEX, DIMENSION, DOUBLE, FLOAT, INDEX, INTEGER, L64, LONG, NOZERO, OBJ, PTR, SIZE, STRING, TYPE, UINT, UL64, ULONG, VALUE

MAP_CLIP_SET procedure

positional arguments: none
keywords: CLIP_PLANE, CLIP_UV, RESET, SPLIT, TRANSFORM

MAP_CONTINENTS procedure

positional arguments: none
keywords: COLOR, COUNTRIES, FILL_CONTINENTS, HIRES, RIVERS

Wessel and Smith [7]

MATRIX_MULTIPLY() function

positional arguments: 2
keywords: ATRANSPOSE, BTRANSPOSE

MAX() function

positional arguments: 2
keywords: DIMENSION, MIN, NAN, SUBSCRIPT_MIN

MEAN() function

positional arguments: 1
keywords: DOUBLE, NAN

MEANABSDEV() function

positional arguments: 1
keywords: DOUBLE, NAN

MEDIAN() function

positional arguments: 2
keywords: DIMENSION, DOUBLE, EVEN

MEMORY() function

positional arguments: 1
keywords: CURRENT, HIGHWATER, L64, NUM_ALLOC, NUM_FREE, STRUCTURE

MESSAGE procedure

positional arguments: 1
keywords: CONTINUE, INFORMATIONAL, IOERROR, NONAME, NOPREFIX, NO-PRINT, RESET, TRACEBACK

MIN() function

positional arguments: 2
keywords: DIMENSION, MAX, NAN, SUBSCRIPT_MAX

MOMENT() function

positional arguments: 1
keywords: DOUBLE, MAXMOMENT, MDEV, NAN, SDEV

NCDF_ATTCOPY() function

positional arguments: 5
keywords: IN_GLOBAL, OUT_GLOBAL

NCDF_ATTDEL procedure

positional arguments: 3
keywords: GLOBAL

NCDF_ATTGET procedure

positional arguments: 4
keywords: GLOBAL

NCDF_ATTINQ() function

positional arguments: 3
keywords: GLOBAL

NCDF_ATTNAME() function

positional arguments: 3
keywords: GLOBAL

NCDF_ATTPUT procedure

positional arguments: 4
keywords: BYTE, CHAR, DOUBLE, FLOAT, GLOBAL, LENGTH, LONG, SHORT

NCDF_ATTRENAME procedure

positional arguments: 4
keywords: GLOBAL

NCDF_CLOSE procedure

positional arguments: 1
keywords: none

NCDF_CONTROL procedure

positional arguments: 1
keywords: ABORT, ENDEF, FILL, NOFILL, NOVERBOSE, OLDFILL, REDEF, SYNC, VERBOSE

NCDF_CREATE() function

positional arguments: 1
keywords: CLOBBER, NOCLOBBER

NCDF_DIMDEF() function

positional arguments: 3
keywords: UNLIMITED

NCDF_DIMID() function

positional arguments: 2
keywords: none

NCDF_DIMINQ procedure

positional arguments: 4
keywords: none

NCDF_DIMRENAME procedure

positional arguments: 3
keywords: none

NCDF_EXISTS() function

positional arguments: none
keywords: none

```
print, 'GDL compiled with netCDF support: ' $
  + (ncdf_exists() ? 'yes' : 'no')
```

```
GDL compiled with netCDF support: yes
```

NCDF_INQUIRE() function

positional arguments: 1
keywords: none

NCDF_OPEN() function

positional arguments: 1
keywords: NOWRITE, WRITE

NCDF_VARDEF() function

positional arguments: 3
keywords: BYTE, CHAR, DOUBLE, FLOAT, LONG, SHORT

NCDF_VARGET procedure

positional arguments: 3
keywords: COUNT, OFFSET, STRIDE

NCDF_VARGET1 procedure

positional arguments: 3
keywords: OFFSET

NCDF_VARID() function

positional arguments: 2
keywords: none

NCDF_VARINQ() function

positional arguments: 2
keywords: none

NCDF_VARPUT procedure

positional arguments: 3
keywords: COUNT, OFFSET, STRIDE

NCDF_CONTROL with SYNC to force...

NCDF_VARRENAME procedure

positional arguments: 3
keywords: none

NEWTON() function

positional arguments: 2
keywords: DOUBLE, HYBRID, ITMAX, TOLF, TOLX

Galassi et al. [2]

NORM() function

positional arguments: 1
keywords: DOUBLE

N_ELEMENTS() function

positional arguments: 1
keywords: none

N_PARAMS() function

positional arguments: 1
keywords: none

N_TAGS() function

positional arguments: 1
keywords: DATA_LENGTH, LENGTH

OBJARR() function

positional arguments: 8
keywords: NOZERO

OBJ_CLASS() function

positional arguments: 1
keywords: COUNT, SUPERCLASS

Returns the name of the class of an object passed as the first argument.

SUPERCLASS keyword

Returns instead an array of all direct superclasses of the object passed as the first argument. In this case the first argument may be a string defining the object name.

COUNT keyword

Allows to pass a reference to a variable into which the number of direct superclasses will be stored.

```
$ tail *__define.pro
bottle = obj_new('beer')
print, 'bottle is a[n] ', obj_class(bottle)
spr = obj_class('beer', /superclass, count=cnt)
print, 'beer has ', strtrim(cnt,2) , ' direct superclass[es]: ', strjoin(spr, ',')
```

```
==> alcoholic_drink__define.pro <==
pro alcoholic_drink__define
  struct = {alcoholic_drink, proof : 0, inherits drink}
end

==> beer__define.pro <==
pro beer__define
  struct = {beer, inherits alcoholic_drink}
end

==> drink__define.pro <==
pro drink__define
  struct = {drink, color : 0}
end
% Compiled module: BEER__DEFINE.
% Compiled module: ALCOHOLIC_DRINK__DEFINE.
% Compiled module: DRINK__DEFINE.
bottle is a[n] BEER
beer has 1 direct superclass[es]: ALCOHOLIC_DRINK
```

A list of all known classes is returned if called without any argument:

```
classes = obj_class()
help, classes
print, classes
```

```
CLASSES         STRING    = Array[24]
!PLT !GNUDATALANGUAGE !AXIS !VERSION !MOUSE !ERROR_STATE !VALUES !MAP !CPU !WARN !USERSYM IDL_SIZE FSTAT64 FSTAT FILE_INFO IDL_MEMORY
IDL_MEMORY64 MACHAR DMACHAR WIDGET_BUTTON WIDGET_DROPLIST WIDGET_TEXT WIDGET_VERSION !DEVICE
```

OBJ_DESTROY procedure

positional arguments: any number
keywords: _REF_EXTRA

OBJ_ISA() function

positional arguments: 2
keywords: none

OBJ_NEW() function

positional arguments: any number
keywords: _REF_EXTRA

Beware that values of object fields may only be initialised in the constructor, and not while defining the object structure, i.e.:

```
$ cat test__define.pro
a = obj_new('test')
a->printXY
```

```
pro test::printXY
  print, self.x, self.y
end
function test::init
  self.x = 10
  return, 1
end
pro test__define
  struct = {test, x : 5, y : 5}
end
% Compiled module: TEST__DEFINE.
      10       0
```

OBJ_VALID() function

positional arguments: 1
keywords: CAST, COUNT

ON_ERROR procedure

positional arguments: 1
keywords: none

OPENR procedure

positional arguments: 3
keywords: APPEND, BINARY, BLOCK, BUFSIZE, COMPRESS, DELETE, ERROR, F77_UNFORMATTED, GET_LUN, MORE, NOAUTOMODE, STDIO, STREAM, SWAP_ENDIAN, SWAP_IF_BIG_ENDIAN, SWAP_IF_LITTLE_ENDIAN, VAX_FLOAT, WIDTH, XDR

COMPRESS keyword

```
$ echo "GDL rocks!" > file.txt
$ gzip -f file.txt
openr, u, 'file.txt.gz', /get_lun, /compress
s = ''
readu, u, s
free_lun, u
print, s
$ rm file.txt.gz
```

```
GDL rocks!
```

OPENU procedure

positional arguments: 3
keywords: APPEND, BINARY, BLOCK, BUFSIZE, COMPRESS, DELETE, ERROR, F77_UNFORMATTED, GET_LUN, MORE, NOAUTOMODE, STDIO, STREAM, SWAP_ENDIAN, SWAP_IF_BIG_ENDIAN, SWAP_IF_LITTLE_ENDIAN, VAX_FLOAT, WIDTH, XDR

OPENW procedure

positional arguments: 3
keywords: APPEND, BINARY, BLOCK, BUFSIZE, COMPRESS, DELETE, ERROR, F77_UNFORMATTED, GET_LUN, MORE, NOAUTOMODE, STDIO, STREAM, SWAP_ENDIAN, SWAP_IF_BIG_ENDIAN, SWAP_IF_LITTLE_ENDIAN, VAX_FLOAT, WIDTH, XDR

OPLOT procedure

positional arguments: 2
keywords: CLIP, COLOR, LINESTYLE, MAX_VALUE, MIN_VALUE, NOCLIP, NSUM, POLAR, PSYM, SYMSIZE, T3D, THICK

PARSE_URL() function

positional arguments: 1
keywords: none

Returns a structure describing components of the URL passed as an argument, e.g.:

```
help, parse_url('http://root:qwerty@kgb.ru:666/?hack'), /stru
```

```
** Structure <Anonymous>, 7 tags, data length=56:
   SCHEME          STRING     'http'
   USERNAME        STRING     'root'
   PASSWORD        STRING     'qwerty'
   HOST            STRING     'kgb.ru'
   PORT            STRING     '666'
   PATH            STRING     '/'
   QUERY           STRING     'hack'
```

PATH_SEP() function

positional arguments: none
keywords: PARENT_DIRECTORY, SEARCH_PATH, TEST

PLOT procedure

positional arguments: 2
keywords: BACKGROUND, CHARSIZE, CHARTHICK, CLIP, COLOR, DATA, DEVICE, LINESTYLE, MAX_VALUE, MIN_VALUE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, PSYM, SUBTITLE, SYMSIZE, THICK, TICKLEN, TITLE, XCHARSIZE, XLOG, XMARGIN, XMINOR, XRANGE, XSTYLE, XTHICK, XTICKFORMAT, XTICKLEN, XTICKS, XTITLE, XTYPE, YCHARSIZE, YLOG, YMARGIN, YMINOR, YNOZERO, YRANGE, YSTYLE, YTHICK, YTICKFORMAT, YTICKLEN, YTICKS, YTITLE, YTYPE, ZCHARSIZE, ZGRIDSTYLE, ZMARGIN, ZMINOR, ZRANGE, ZSTYLE, ZTHICK, ZTICKFORMAT, ZTICKLEN, ZTICKS, ZTITLE, ZVALUE

PLOTERR procedure

positional arguments: 4
keywords: BAR_COLOR, HAT, HELP, LENGTH_OF_HAT, PSYM, TEST, TYPE, XLOG, XRANGE, YLOG, YRANGE, _EXTRA

PLOTS procedure

positional arguments: 3
keywords: CLIP, COLOR, CONTINUE, DATA, DEVICE, LINESTYLE, NOCLIP, NORMAL, PSYM, SYMSIZE, T3D, THICK

PM procedure

positional arguments: any number
keywords: FORMAT, TITLE

```
arr = indgen(4,4)
fmt = '(4I3)'
print, 'PM'
pm, arr, format=fmt
print, 'PRINT:'
print, arr, format=fmt
```

```
PM
  0  4  8 12
  1  5  9 13
  2  6 10 14
  3  7 11 15
PRINT:
  0  1  2  3
  4  5  6  7
  8  9 10 11
 12 13 14 15
```

see also: ORDER keyword in TV, TVRD(), ... (TODO: section on # and ## ops.)

POINT_LUN procedure

positional arguments: 2
keywords: none

POLY() function

positional arguments: 2
keywords: none

POLYFILL procedure

positional arguments: 3
keywords: CLIP, COLOR, DATA, DEVICE, LINESTYLE, LINE_FILL, NOCLIP, NORMAL, ORIENTATION, SPACING, THICK

POLY_2D() function

positional arguments: 6
keywords: CUBIC, MISSING

POLY_AREA() function

positional arguments: 2
keywords: DOUBLE, SIGNED

POPD procedure

positional arguments: none
keywords: none

PREWITT() function

positional arguments: 1
keywords: HELP

PRIMES() function

positional arguments: 1
keywords: none

PRINT procedure

positional arguments: any number
keywords: AM_PM, DAYS_OF_WEEK, FORMAT, MONTH, STDIO_NON_FINITE

PRINTD procedure

positional arguments: none
keywords: none

PRINTF procedure

positional arguments: any number
keywords: AM_PM, DAYS_OF_WEEK, FORMAT, MONTH, STDIO_NON_FINITE

PRODUCT() function

positional arguments: 2
keywords: CUMULATIVE, INTEGER, NAN, PRESERVE_TYPE

multi-threading: this routine uses GDL thread pool if working on large array, see the...

PTRARR() function

positional arguments: 8
keywords: ALLOCATE_HEAP, NOZERO

multi-threading: this routine uses GDL thread pool if working on large array, see the...

PTR_FREE procedure

positional arguments: any number
keywords: none

PTR_FREE can also be used to deallocate a variable:

```
a = 1
ptr_free , ptr_new(a, /no_copy)
help , a
```

```
A               UNDEFINED = <Undefined>
```

see also: PTR_NEW(), PTR_VALID()

PTR_NEW() function

positional arguments: 1
keywords: ALLOCATE_HEAP, NO_COPY

PTR_VALID() function

positional arguments: 1
keywords: CAST, COUNT

PUSHD procedure

positional arguments: 1
keywords: none

PYTHON procedure

positional arguments: any number
keywords: ARGV

PYTHON() function

positional arguments: any number
keywords: ARGV, DEFAULTRETURNVALUE

Executes a python function whose name is specified using the second argument, the first argument defines the package (e.g. numpy). All other argument are passed as positional arguments to the function.

```
print, python('numpy', 'arange', 4.)
```

```
        0.0000000       1.0000000       2.0000000       3.0000000
```

PY_PLOT procedure

positional arguments: 2
keywords: GRID, TITLE, XLABEL, YLABEL

PY_PRINT procedure

positional arguments: 1
keywords: none

QUERY_BMP() function

positional arguments: 2
keywords: none

```
$ wget --quiet http://wikipedia.org/favicon.ico
$ convert favicon.ico favicon.bmp
ok = query_bmp('favicon.bmp', info)
if ok then help, info, /structure else print, 'query failed!'
$ rm favicon.*
```

```
% Compiled module: QUERY_BMP.
** Structure <Anonymous>, 7 tags, data length=56:
   CHANNELS        LONG                        4
   DIMENSIONS      LONG          Array[2]
   HAS_PALETTE     INT                    0
   IMAGE_INDEX     LONG                        0
   NUM_IMAGES      LONG                        1
   PIXEL_TYPE      INT                    1
   TYPE            STRING        'BMP'
```

QUERY_DICOM() function

positional arguments: 2
keywords: none

QUERY_GIF() function

positional arguments: 2
keywords: none

QUERY_IMAGE() function

positional arguments: 2
keywords: _REF_EXTRA

QUERY_JPEG() function

positional arguments: 2
keywords: none

QUERY_PICT() function

positional arguments: 2
keywords: none

QUERY_PNG() function

positional arguments: 2
keywords: none

QUERY_PPM() function

positional arguments: 2
keywords: none

QUERY_TIFF() function

positional arguments: 2
keywords: IMAGE_INDEX

RADON() function

positional arguments: 1
keywords: BACKPROJECT, DOUBLE, DRHO, DX, DY, GRAY, LINEAR, NRHO, NTHETA, NX, NY, RHO, RMIN, THETA, XMIN, YMIN

RANDOMN() function

positional arguments: 8
keywords: BINOMIAL, DOUBLE, GAMMA, LONG, NORMAL, POISSON, UNIFORM

RANDOMU() function

positional arguments: 8
keywords: BINOMIAL, DOUBLE, GAMMA, LONG, NORMAL, POISSON, UNIFORM

READ procedure

positional arguments: any number
keywords: AM_PM, DAYS_OF_WEEK, FORMAT, MONTH, PROMPT

READF procedure

positional arguments: any number
keywords: AM_PM, DAYS_OF_WEEK, FORMAT, MONTH, PROMPT

READS procedure

positional arguments: any number
keywords: AM_PM, DAYS_OF_WEEK, FORMAT, MONTH

READU procedure

positional arguments: any number
keywords: TRANSFER_COUNT

READ_ASCII() function

positional arguments: 1
keywords: COMMENT_SYMBOL, COUNT, DATA_START, DELIMITER, HEADER, HELP, MISSING_VALUE, NUM_RECORDS, RECORD_START, TEMPLATE, TEST, VERBOSE

READ_BINARY() function

positional arguments: 1
keywords: DATA_DIMS, DATA_START, DATA_TYPE, ENDIAN, TEMPLATE

READ_BMP() function

positional arguments: 4
keywords: RGB

READ_DICOM() function

positional arguments: 4
keywords: IMAGE_INDEX

READ_GIF procedure

positional arguments: 5
keywords: DEBUG, HELP, TEST

READ_JPEG procedure

positional arguments: 3
keywords: BUFFER, COLORS, DEBUG, DITHER, GRAYSCALE, HELP, ORDER, TEST, TRUE, TWO_PASS_QUANTIZE, UNIT

READ_PICT procedure

positional arguments: 5
keywords: none

READ_PNG() function

positional arguments: 4
keywords: HELP, ORDER, TEST, TRANSPARENT, VERBOSE

READ_TIFF() function

positional arguments: 4
keywords: CHANNELS, GEOTIFF, IMAGE_INDEX, INTERLEAVE, ORIENTATION, PLANARCONFIG, SUB_RECT, VERBOSE

READ_XWD() function

positional arguments: 4
keywords: none

REAL_PART() function

positional arguments: 1
keywords: none

REBIN() function

positional arguments: 9
keywords: SAMPLE

RECALL_COMMANDS() function

positional arguments: none
keywords: none

REFORM() function

positional arguments: 8
keywords: OVERWRITE

REPLICATE() function

positional arguments: 9
keywords: none

REPLICATE_INPLACE procedure

positional arguments: 6
keywords: none

RESOLVE_ROUTINE procedure

positional arguments: 1
keywords: none

RESTORE procedure

positional arguments: 1
keywords: DESCRIPTION, FILENAME, RELAXED_STRUCTURE_ASSIGNMENT, RE-STORED_OBJECTS, VERBOSE

RETALL procedure

positional arguments: none
keywords: RETALL

REVERSE() function

positional arguments: 2
keywords: OVERWRITE

RK4() function

positional arguments: 5
keywords: DOUBLE, ITER

RK4JMG() function

positional arguments: 5
keywords: DOUBLE

ROBERTS() function

positional arguments: 1
keywords: HELP

ROTATE() function

positional arguments: 2
keywords: none

ROUND() function

positional arguments: 1
keywords: L64

multi-threading: this routine uses GDL thread pool if working on large array, see the...

ROUTINE_INFO() function

positional arguments: 1
keywords: DISABLED, ENABLED, FUNCTIONS, PARAMETERS, SYSTEM

ROUTINE_NAMES() function

positional arguments: any number
keywords: ARG_NAME, FETCH, LEVEL, STORE, S_FUNCTIONS, S_PROCEDURES, VARIABLES

Examines variables and parameters of procedures and the call stack Using ROUTINE_NAMES a subroutine can interrogate, and in some cases change, the values and names of variables and parameters in its calling routine, or at the $MAIN$ level.

ROUTINE_NAMES uses a notion of the current "call level," which is the numerical stack depth of the currently executing routine. At each procedure or function call, the call level becomes one **deeper**, and upon each RETURN, the call level becomes one **shallower**. The call stack always begins at the $MAIN$ level. The current call stack can always be printed by executing HELP.

When specifying the call level to ROUTINE_NAMES, one can use one of two numbering systems, depending on whichever is most convenient. In the **absolute** numbering system, the $MAIN$ level starts at number 1, and becomes deeper with increasing numbers. In the **relative** numbering system, the current (deepest) call level is number 0, and becomes shallower with more negative numbers. Hence, if the deepest level is N, then the correspondence is thus:

VALUE	MEANING
1 or -N+1	$MAIN$ level
2 or -N+2	NEXT deeper level
...	...
N or 0	DEEPEST (currently executing) level

When called without any keyword ROUTINE_NAMES returns a string array containing a list of currently compiled functions and procedures, e.g.:

```
$ cat library.pro
.compile library.pro
print, routine_names()
```

```
pro a_procedure
  print, 'Hello world!'
end
function a_function
  return, 'Hello world!'
end
% Compiled module: A_PROCEDURE.
% Compiled module: A_FUNCTION.
```

```
$MAIN$ A_FUNCTION A_PROCEDURE
```

ROUTINE_NAMES can be invoked in several other ways, which are detailed below together with keyword descriptions.

S_PROCEDURES keyword

The lists of system procedures is returned, as a string array. The list does not cover procedures written in GDL itself which are also part of GDL's routine library (e.g. WRITE_PNG).

```
print, (routine_names(/s_pro))[0:5]
```

```
AXIS BYTEORDER CALDAT CALL_METHOD CALL_PROCEDURE CATCH
```

S_FUNCTIONS keyword

The lists of system functions is returned, as a string array. The list does not cover functions written in GDL itself which are also part of GDL's routine library (e.g. READ_PNG()).

```
help, routine_names(/s_functions)
```

```
<Expression>    STRING    = Array[250]
```

LEVEL keyword

The call level of the calling routine is returned, e.g.:

```
$ cat func.pro
print, routine_names(/level), func()
```

```
function func
  return, routine_names(/level)
end
% Compiled module: FUNC.
           1            2
```

ARG_NAME keyword

The names of variables passed as positional arguments at call level specified with the ARG_NAME keyword are returned, as a string array. Note that the arguments passed are the actual parameters, not strings containing their names. All of the arguments must be parameters that have been passed to the calling procedure. Variables that are unnamed at the specified call level will return the empty string.

```
$ cat procedure.pro
a1 = 1
a2 = '2'
a3 = [3b]
procedure, a1, a2, a3
```

```
pro procedure, arg0, arg1, arg2
  print, routine_names(arg1, arg2, arg_name=0)
  print, routine_names(arg1, arg2, arg_name=-1)
end
% Compiled module: PROCEDURE.
ARG1 ARG2
A2 A3
```

VARIABLES keyword

The names of variables at call level specified with the VARIABLES keyword are returned, as a string array, e.g.:

```
$ cat procedure.pro
str = 'Hello world!'
arr = ['Hello', 'world', '!']
int = 0
procedure
```

```
pro procedure
  print, routine_names(variables=-1)
end
% Compiled module: PROCEDURE.
STR ARR INT
```

FETCH keyword

The value of a variable which name is passed in the first argument (string) at call level specified with the FETCH keyword is returned. If the value is undefined, then the assignment will cause an error. Therefore, the only safe way to retrieve a value is by using a variant of the following:

```
if n_elements(routine_names('a', fetch=0)) gt 0 $
  then value = routine_names('a', fetch=0) $
  else message, 'a is not defined!'
```

```
% $MAIN$: a is not defined!
% Execution halted at: $MAIN$
```

STORE keyword

The value specified with the second argument is stored into the variable which name is passed in the first argument (string) at the call level specified with the STORE keyword. Note that there is no way to cause the named variable to become undefined. The value returned can be ignored.

```
a = 1
dummy = routine_names('a', 2, store=0)
print, a
```

```
       2
```

see also: ROUTINE_INFO(), ARG_PRESENT(), SCOPE_VARFETCH()

disclaimer: Entry based on Craig Markwardt's documentation for ROUTINE_NAMES: Copyright (C) 2000, Craig Markwardt. This software is provided as is without any warranty whatsoever. Permission to use, copy, modify, and distribute modified or unmodified copies is granted, provided this copyright and disclaimer are included unchanged.

RSTRPOS() function

positional arguments: 3
keywords: none

SAVE procedure

positional arguments: 30
keywords: ALL, APPEND, COMPATIBLE, DATA, ERRMSG, FILENAME, MTIMES, NAMES, NOCATCH, PASS_METHOD, QUIET, STATUS, TEST, USEUNIT, VARSTATUS, VERBOSE, XDR

SCOPE_VARFETCH() function

positional arguments: 1
keywords: LEVEL

SEM_CREATE() function

positional arguments: 1
keywords: DESTROY_SEMAPHORE

SEM_DELETE procedure

positional arguments: 1
keywords: none

SEM_LOCK() function

positional arguments: 1
keywords: none

SEM_RELEASE procedure

positional arguments: 1
keywords: none

SETENV procedure

positional arguments: 1
keywords: none

SET_PLOT procedure

positional arguments: 1
keywords: COPY, INTERPOLATE

SHIFT() function

positional arguments: 9
keywords: none

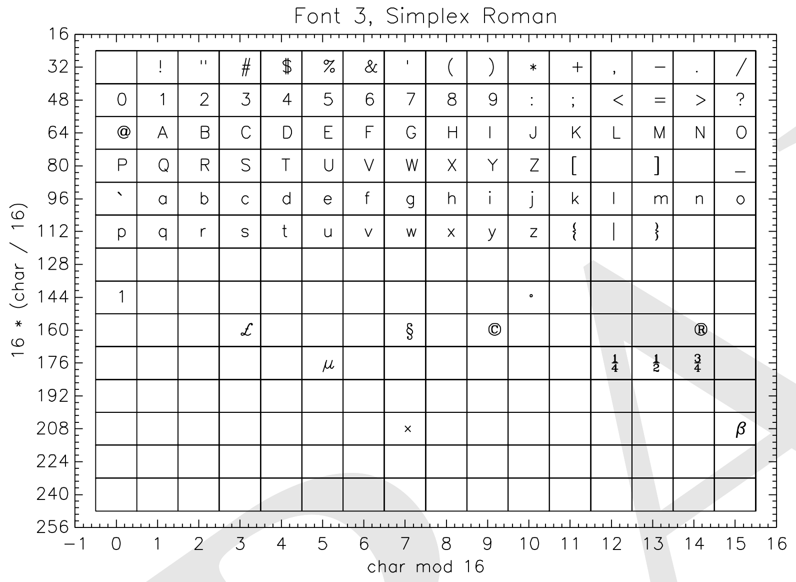
SHOWFONT procedure

positional arguments: 2
keywords: BASE, BEG, ENCAPSULATED, FIN, TT_FONT

Displays a table of fonts for a give font number (first argument) in the current graphics terminal, e.g.:

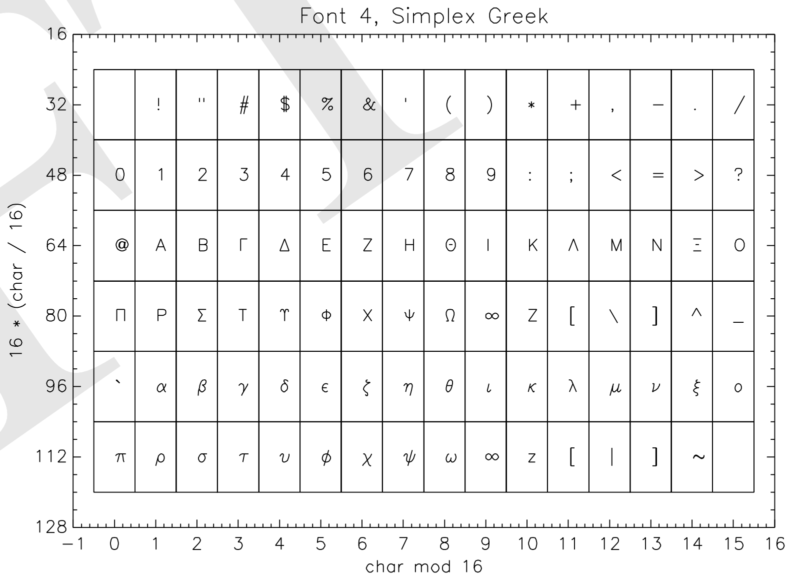
```
showfont, 3, 'Simplex Roman'
```

```
% Compiled module: SHOWFONT.
```



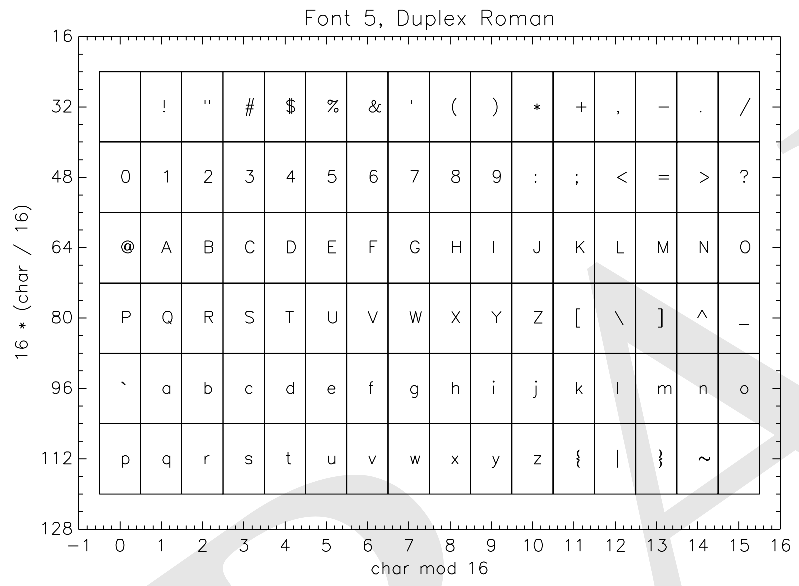
```
showfont, 4, 'Simplex Greek'
```

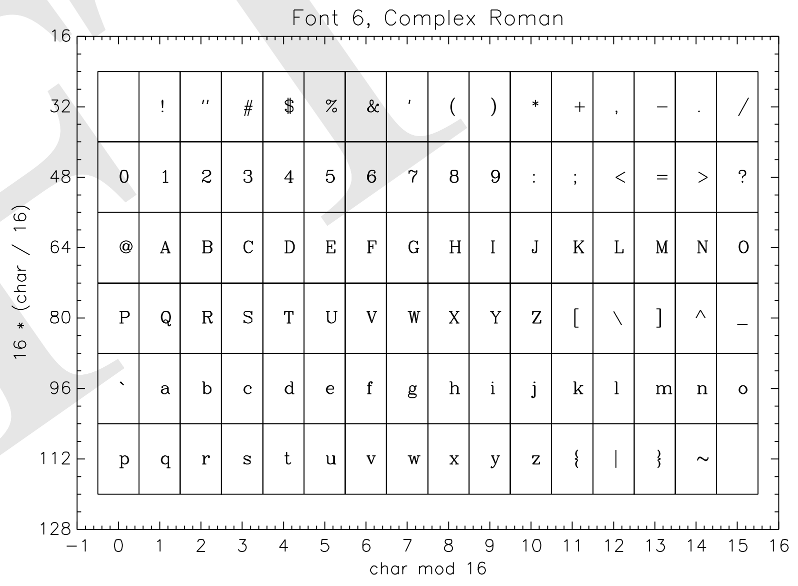
```
% Compiled module: SHOWFONT.
```



```
showfont, 5, 'Duplex Roman'
```

```
% Compiled module: SHOWFONT.
```



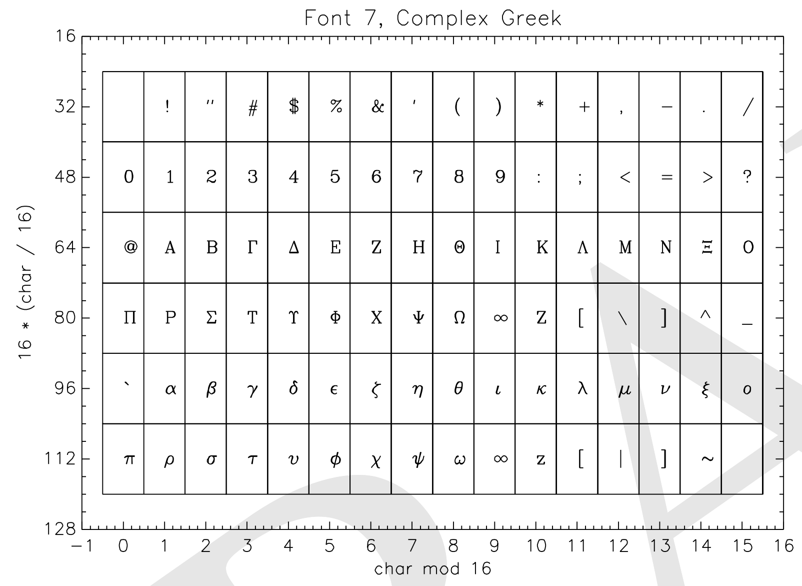


```
showfont, 6, 'Complex Roman'
```

```
showfont, 7, 'Complex Greek'
```

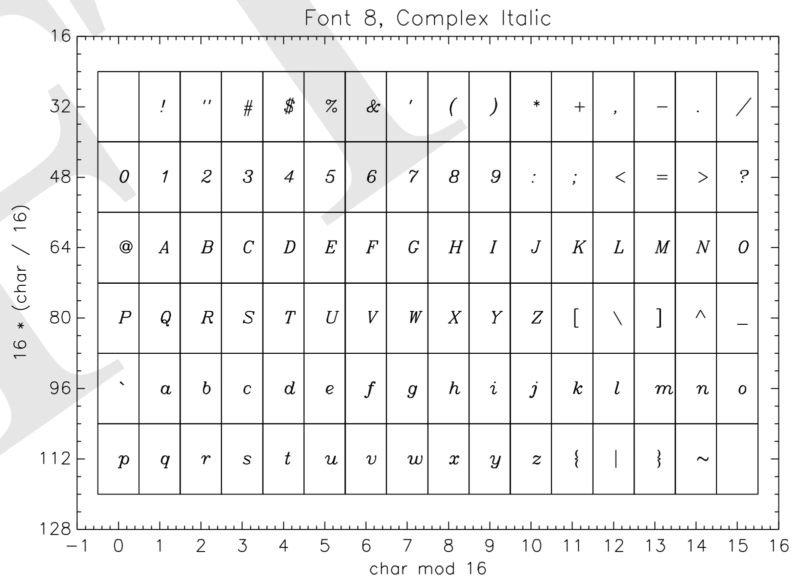
```
% Compiled module: SHOWFONT.
```

```
% Compiled module: SHOWFONT.
```



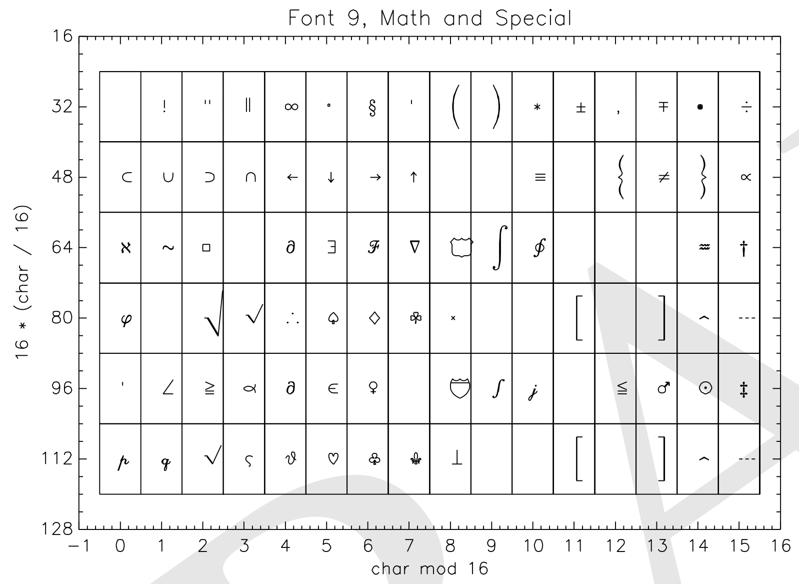
```
showfont, 8, 'Complex Italic'
```

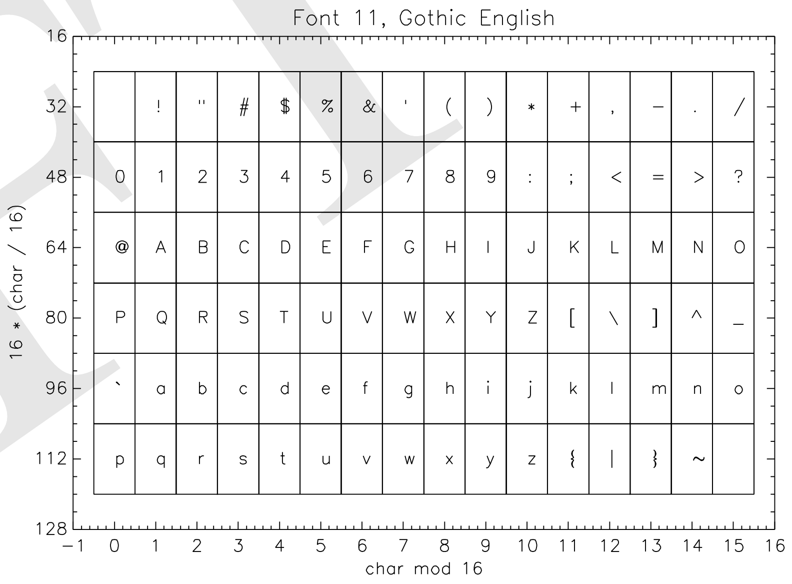
```
% Compiled module: SHOWFONT.
```



```
showfont, 9, 'Math and Special'
```

```
% Compiled module: SHOWFONT.
```



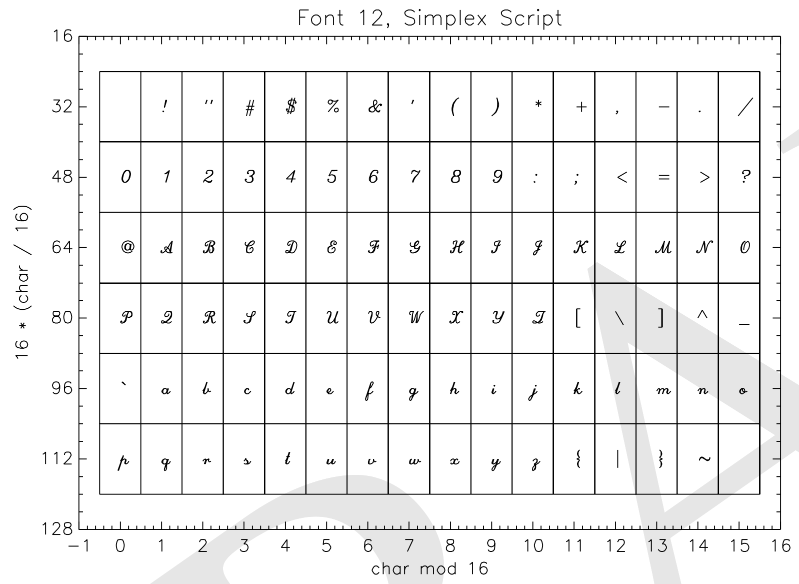


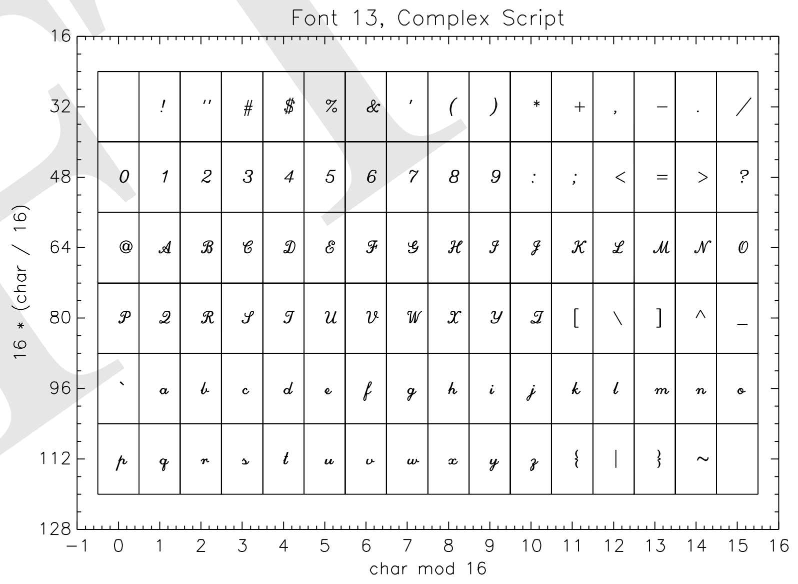
```
showfont, 11, 'Gothic English'
```

```
% Compiled module: SHOWFONT.
```

```
showfont, 12, 'Simplex Script'
```

```
% Compiled module: SHOWFONT.
```



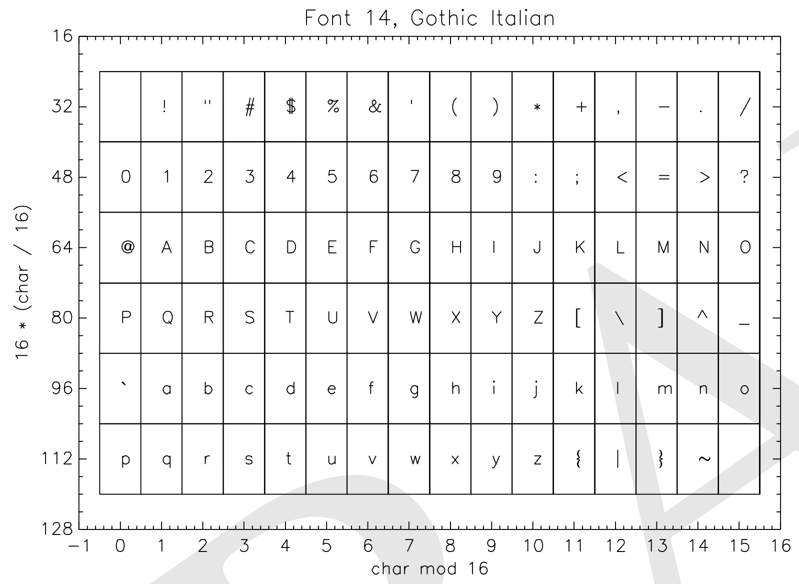


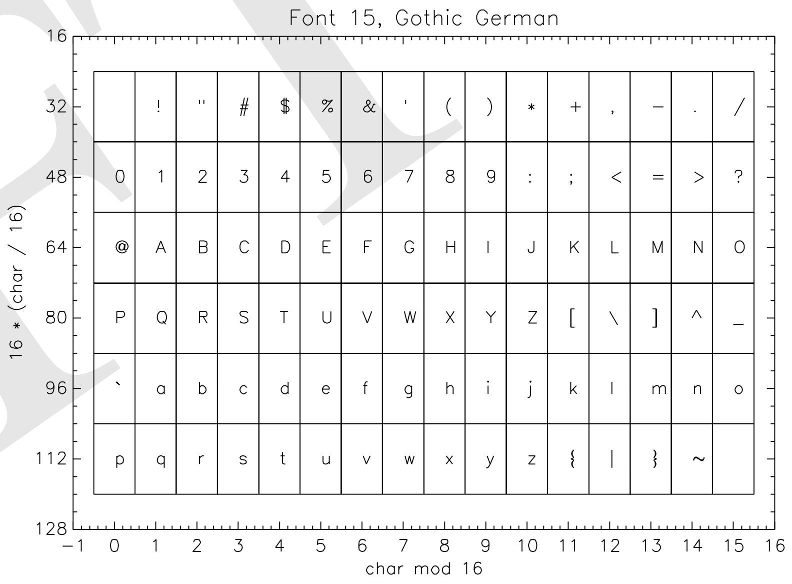
```
showfont, 13, 'Complex Script'
```

```
% Compiled module: SHOWFONT.
```

```
showfont, 14, 'Gothic Italian'
```

```
% Compiled module: SHOWFONT.
```



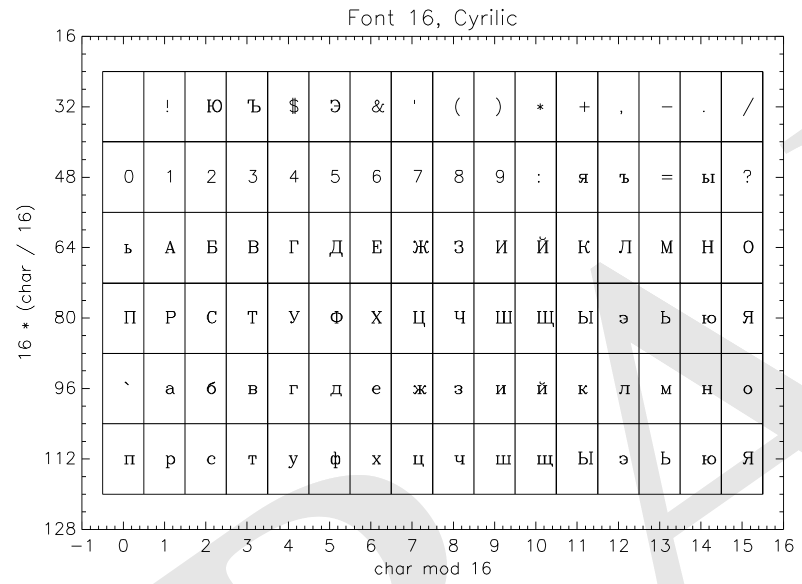


```
showfont, 15, 'Gothic German'
```

```
% Compiled module: SHOWFONT.
```

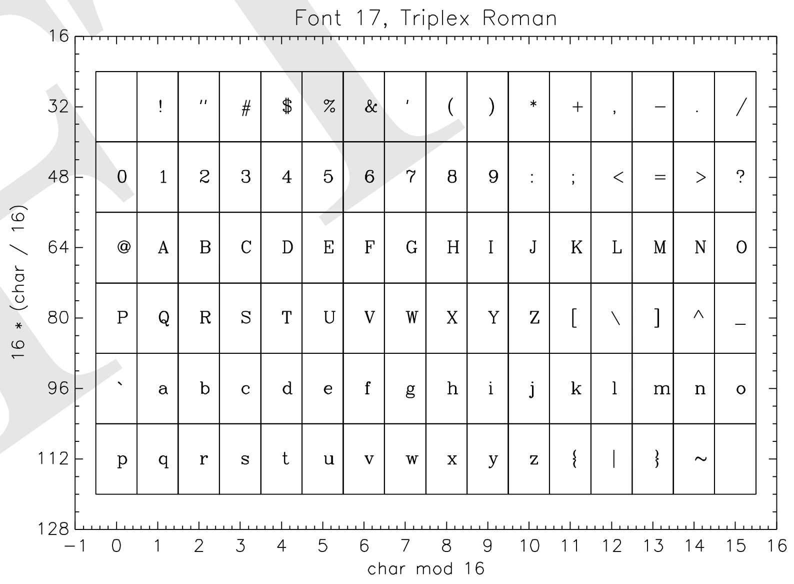
```
showfont, 16, 'Cyrilic'
```

```
% Compiled module: SHOWFONT.
```



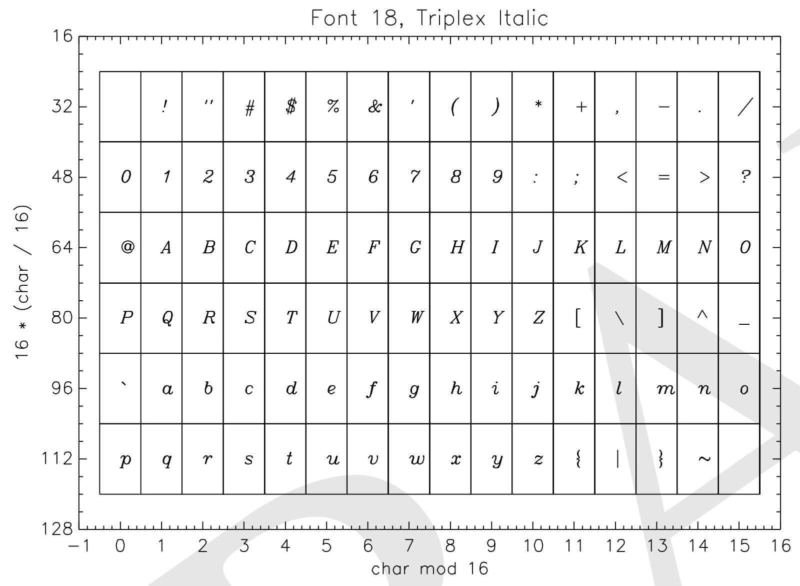
```
showfont, 17, 'Triplex Roman'
```

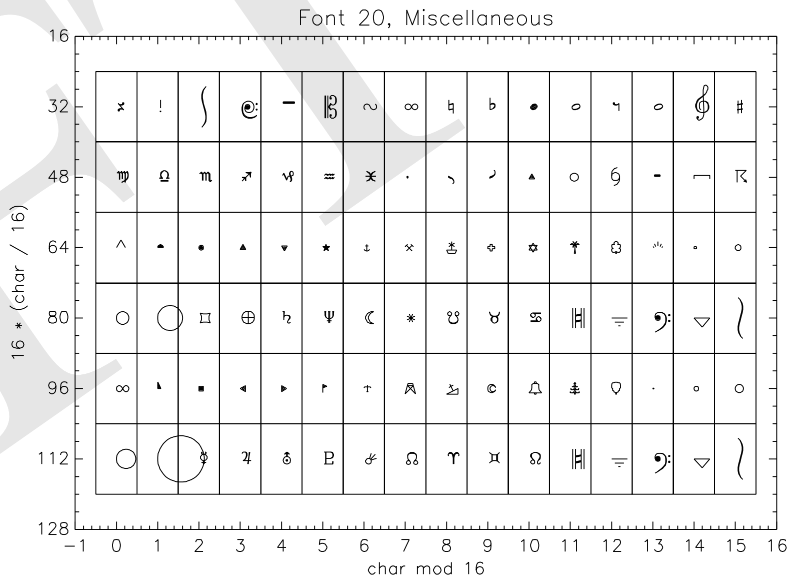
```
% Compiled module: SHOWFONT.
```



```
showfont, 18, 'Triplex Italic'
```

```
% Compiled module: SHOWFONT.
```





```
showfont, 20, 'Miscellaneous'
```

```
% Compiled module: SHOWFONT.
```

SIN() function

positional arguments: 1
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

SINDGEN() function

positional arguments: 8
keywords: none

SINH() function

positional arguments: 1
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

SIZE() function

positional arguments: 1
keywords: DIMENSIONS, FILE_LUN, L64, N_DIMENSIONS, N_ELEMENTS, STRUCTURE, TNAME, TYPE

SKEWNESS() function

positional arguments: 1
keywords: DOUBLE, NAN

SKIP_LUN procedure

positional arguments: 2
keywords: EOF, HELP, LINES, TEST, TRANSFER_COUNT

SMOOTH() function

positional arguments: 2
keywords: EDGE_TRUNCATE, HELP, NAN, TEST, VERBOSE

SOBEL() function

positional arguments: 1
keywords: HELP

SOCKET procedure

positional arguments: 3
keywords: CONNECT_TIMEOUT, ERROR, GET_LUN, READ_TIMEOUT, STDIO, SWAP_ENDIAN, SWAP_IF_BIG_ENDIAN, SWAP_IF_LITTLE_ENDIAN, WIDTH, WRITE_TIMEOUT

SORT() function

positional arguments: 1
keywords: L64

SPAWN procedure

positional arguments: 3
keywords: COUNT, EXIT_STATUS, NOSHELL, PID, SH, UNIT

SPHER_HARM() function

positional arguments: 4
keywords: DOUBLE

SPL_INIT() function

positional arguments: 2
keywords: DOUBLE, HELP, YP0, YPN_1

SPL_INIT_OLD() function

positional arguments: 2
keywords: DEBUG, DOUBLE, YP0, YPN_1

SPL_INTERP() function

positional arguments: 4
keywords: DOUBLE, HELP

SPL_INTERP_OLD() function

positional arguments: 4
keywords: DOUBLE

SQRT() function

positional arguments: 1
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

STDDEV() function

positional arguments: 1
keywords: DOUBLE, NAN

STOP procedure

positional arguments: any number
keywords: AM_PM, DAYS_OF_WEEK, FORMAT, MONTH, STDIO_NON_FINITE

STRARR() function

positional arguments: 8
keywords: NOZERO

STRCMP() function

positional arguments: 3
keywords: FOLD_CASE

STRCOMPRESS() function

positional arguments: 1
keywords: REMOVE_ALL

multi-threading: this routine uses GDL thread pool if working on large array, see the...

STREGEX() function

positional arguments: 2
keywords: BOOLEAN, EXTRACT, FOLD_CASE, LENGTH, SUBEXPR

STRING() function

positional arguments: any number
keywords: AM_PM, DAYS_OF_WEEK, FORMAT, MONTH, PRINT

PRINT keyword

```
help, string(55b)
help, string(55b, /print)
help, string(findgen(2,2))
help, string(findgen(2,2), /print)
help, string(findgen(2), /print)
```

```
<Expression>    STRING    = '7'
<Expression>    STRING    = '  55'
<Expression>    STRING    = Array[2, 2]
<Expression>    STRING    = Array[2]
<Expression>    STRING    = '      0.00000      1.00000'
```

STRJOIN() function

positional arguments: 2
keywords: SINGLE

```
arr = ['a', 'b', 'c']
str = strjoin(arr)
help, arr, str
```

```
ARR             STRING    = Array[3]
STR             STRING    = 'abc'
```

```
arr = [['a', 'b', 'c'], ['d', 'e', 'f']]
str = strjoin(arr, '-')
help, arr, str
print, str[0]
print, str[1]
```

```
ARR             STRING    = Array[3, 2]
STR             STRING    = Array[2]
a-b-c
d-e-f
```

SINGLE keyword

```
arr = [['a', 'b', 'c'], ['d', 'e', 'f']]
str = strjoin(arr, '-', /single)
help, arr, str
```

```
ARR             STRING    = Array[3, 2]
STR             STRING    = 'a-b-c-d-e-f'
```

STRLEN() function

positional arguments: 1
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

STRLOWCASE() function

positional arguments: 1
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

STRMATCH() function

positional arguments: 2
keywords: FOLD_CASE

STRMID() function

positional arguments: 3
keywords: REVERSE_OFFSET

multi-threading: this routine uses GDL thread pool if working on large array, see the...

STRPOS() function

positional arguments: 3
keywords: REVERSE_OFFSET, REVERSE_SEARCH

multi-threading: this routine uses GDL thread pool if working on large array, see the...

STRPUT procedure

positional arguments: 3
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

STRSPLIT() function

positional arguments: 2
keywords: COUNT, ESCAPE, EXTRACT, FOLD_CASE, HELP, LENGTH, PRESERVE_NULL, REGEX, TEST

STRTOK() function

positional arguments: 2
keywords: ESCAPE, EXTRACT, LENGTH, PRESERVE_NULL, REGEX

STRTRIM() function

positional arguments: 2
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

STRUCT_ASSIGN procedure

positional arguments: 2
keywords: NOZERO, VERBOSE

STRUPCASE() function

positional arguments: 1
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

STR_SEP() function

positional arguments: 2
keywords: ESC, HELP, REMOVE_ALL, TEST, TRIM

. . . STR_SEP separates the string on **any** of the characters of the 2nd string. . . .

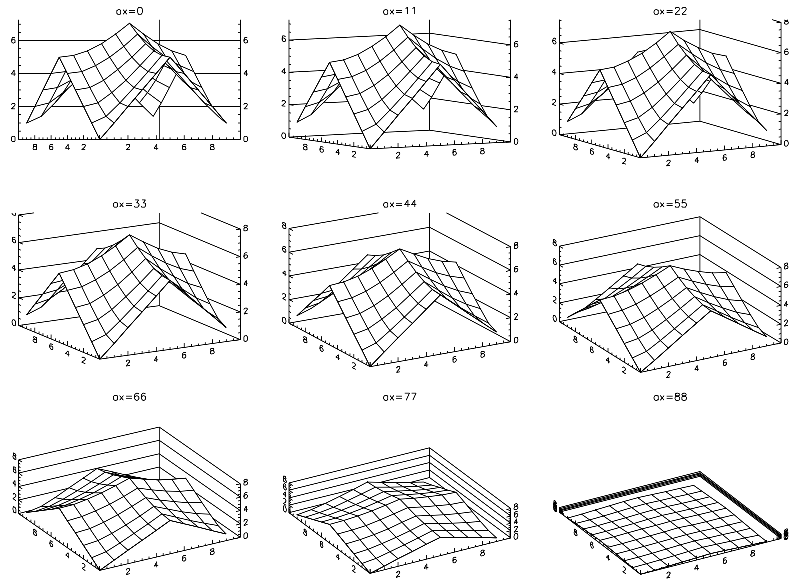
SURFACE procedure

positional arguments: 3
keywords: AX, AZ, BACKGROUND, CHARSIZE, CHARTHICK, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, MAX_VALUE, MIN_VALUE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, SUBTITLE, T3D, THICK, TICKLEN, TITLE, XCHARSIZE, XGRIDSTYLE, XLOG, XMARGIN, XMINOR, XRANGE, XSTYLE, XTHICK, XTICKFORMAT, XTICKINTERVAL, XTICKLAYOUT, XTICKLEN, XTICKNAME, XTICKS, XTICKUNITS, XTICKV, XTICK_GET, XTITLE, XTYPE, YCHARSIZE, YGRIDSTYLE, YLOG, YMARGIN, YMINOR, YRANGE, YSTYLE, YTHICK, YTICKFORMAT, YTICKINTERVAL, YTICKLAYOUT, YTICKLEN, YTICKNAME, YTICKS, YTICKUNITS, YTICKV, YTICK_GET, YTITLE, YTYPE, ZCHARSIZE, ZGRIDSTYLE, ZLOG, ZMARGIN, ZMINOR, ZRANGE, ZSTYLE, ZTHICK, ZTICKFORMAT, ZTICKINTERVAL, ZTICKLAYOUT, ZTICKLEN, ZTICKNAME, ZTICKS, ZTICKUNITS, ZTICKV, ZTICK_GET, ZTITLE, ZTYPE, ZVALUE

AX keyword

```
!P.MULTI = [0,3,3]
d = dist(10)
for ax = 0, 90, 11 do $
  surface, d, ax=ax, title='ax=' + strtrim(ax,2)
```

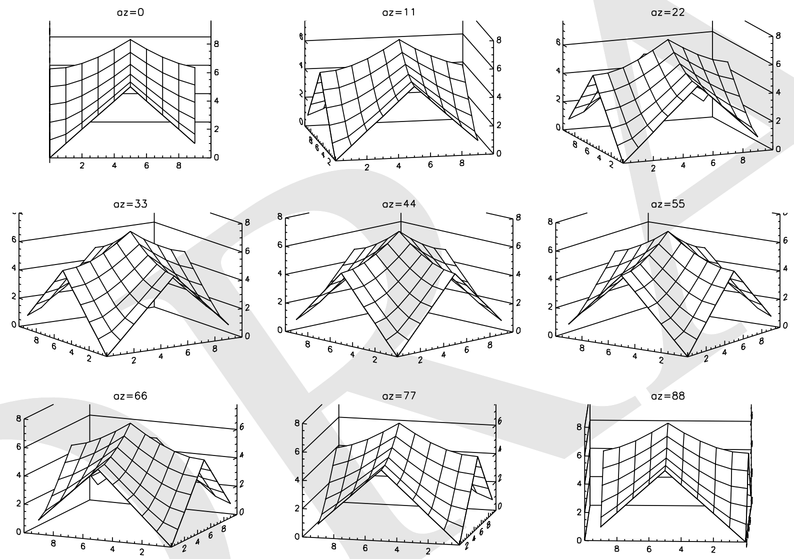
```
% Compiled module: DIST.
```



AZ keyword

```
!P.MULTI = [0,3,3]
d = dist(10)
for az = 0, 90, 11 do $
  surface, d, az=az, title='az=' + strtrim(az,2)
```

```
% Compiled module: DIST.
```



SVDC procedure

positional arguments: 4
keywords: COLUMN, DOUBLE, ITMAX

SWAP_ENDIAN() function

positional arguments: 1
keywords: SWAP_IF_BIG_ENDIAN, SWAP_IF_LITTLE_ENDIAN

SWAP_ENDIAN_INPLACE procedure

positional arguments: 1
keywords: SWAP_IF_BIG_ENDIAN, SWAP_IF_LITTLE_ENDIAN

SYSTIME() function

positional arguments: 2
keywords: JULIAN, SECONDS, UTC

TAG_NAMES() function

positional arguments: 1
keywords: STRUCTURE_NAME

TAN() function

positional arguments: 1
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

TANH() function

positional arguments: 1
keywords: none

multi-threading: this routine uses GDL thread pool if working on large array, see the...

TEMPLATE procedure

positional arguments: none
keywords: none

TEMPLATE_BLANK procedure

positional arguments: none
keywords: none

TEMPORARY() function

positional arguments: 1
keywords: none

TEST procedure

positional arguments: any number
keywords: none

TOTAL() function

positional arguments: 2
keywords: CUMULATIVE, DOUBLE, INTEGER, NAN, PRESERVE_TYPE

multi-threading: this routine uses GDL thread pool if working on large array, see the...

TRACE() function

positional arguments: 1
keywords: DOUBLE

TRANSPOSE() function

positional arguments: 2
keywords: none

TRIGRID() function

positional arguments: 6
keywords: MAP, MAX_VALUE, MISSING, NX, NY

TV procedure

positional arguments: 4
keywords: CHANNEL, DATA, DEVICE, NORMAL, ORDER, TRUE, XSIZE, YSIZE

TVLCT procedure

positional arguments: 4
keywords: GET, HLS, HSV

TVRD() function

positional arguments: 5
keywords: CHANNEL, ORDER, TRUE, WORDS

TVSCL procedure

positional arguments: 3
keywords: NAN, _EXTRA

T_PDF() function

positional arguments: 2
keywords: none

UINDGEN() function

positional arguments: 8
keywords: none

UINT() function

positional arguments: 10
keywords: none

UINTARR() function

positional arguments: 8
keywords: NOZERO

UL64INDGEN() function

positional arguments: 8
keywords: none

ULINDGEN() function

positional arguments: 8
keywords: none

ULON64ARR() function

positional arguments: 8
keywords: NOZERO

ULONARR() function

positional arguments: 8
keywords: NOZERO

ULONG() function

positional arguments: 10
keywords: none

ULONG64() function

positional arguments: 10
keywords: none

UNIQ() function

positional arguments: 2
keywords: none

USERSYM procedure

positional arguments: 2
keywords: COLOR, FILL, THICK

VALUE_LOCATE() function

positional arguments: 2
keywords: L64

VARIANCE() function

positional arguments: 1
keywords: DOUBLE, NAN

VOIGT() function

positional arguments: 2
keywords: DOUBLE, ITER

WAIT procedure

positional arguments: 1
keywords: none

WDELETE procedure

positional arguments: any number
keywords: none

WHERE() function

positional arguments: 2
keywords: COMPLEMENT, NCOMPLEMENT

see also: ARRAY_INDICES()

multi-threading: this routine uses GDL thread pool if working on large array, see the...

WIDGET_BASE() function

positional arguments: 1
keywords: ALIGN_BOTTOM, ALIGN_CENTER, ALIGN_LEFT, ALIGN_RIGHT, ALIGN_TOP, BASE_ALIGN_BOTTOM, BASE_ALIGN_CENTER, BASE_ALIGN_LEFT, BASE_ALIGN_RIGHT, BASE_ALIGN_TOP, COLUMN, CONTEXT_EVENTS, CONTEXT_MENU, DISPLAY_NAME, EVENT_FUNC, EVENT_PRO, EXCLUSIVE, FLOATING, FRAME, FUNC_GET_VALUE, GRID_LAYOUT, GROUP_LEADER, KBRD_FOCUS_EVENTS, KILL_NOTIFY, MAP, MBAR, MODAL, NONEXCLUSIVE, NOTIFY_REALIZE, NO_COPY, PRO_SET_VALUE, RESOURCE_NAME, RNAME_MBAR, ROW, SCROLL, SCR_XSIZE, SCR_YSIZE, SENSITIVE, SPACE, TITLE, TLB_FRAME_ATTR, TLB_ICONIFY_EVENTS, TLB_KILL_REQUEST_EVENTS, TLB_MOVE_EVENTS, TLB_SIZE_EVENTS, TOOLBAR, TRACKING_EVENTS, UNAME, UNITS, UVALUE, XOFFSET, XPAD, XSIZE, X_SCROLL_SIZE, YOFFSET, YPAD, YSIZE, Y_SCROLL_SIZE

WIDGET_BUTTON() function

positional arguments: 1
keywords: ACCELERATOR, ALIGN_CENTER, ALIGN_LEFT, ALIGN_RIGHT, BITMAP, CHECKED_MENU, DYNAMIC_RESIZE, EVENT_FUNC, EVENT_PRO, FONT, FRAME, FUNC_GET_VALUE, GROUP_LEADER, HELP, KILL_NOTIFY, MENU, NOTIFY_REALIZE, NO_COPY, NO_RELEASE, PRO_SET_VALUE, PUSHBUTTON_EVENTS, SCR_XSIZE, SCR_YSIZE, SENSITIVE, SEPARATOR, TAB_MODE, TOOLTIP, TRACKING_EVENTS, UNAME, UNITS, UVALUE, VALUE, XOFFSET, XSIZE, X_BITMAP_EXTRA, YOFFSET, YSIZE

WIDGET_CONTROL procedure

positional arguments: 1
keywords: DESTROY, EVENT_PRO, FUNC_GET_VALUE, GET_UVALUE, GET_VALUE, MANAGED, MAP, NO_COPY, PRO_SET_VALUE, REALIZE, SENSITIVE, SET_BUTTON, SET_DROPLIST_SELECT, SET_UNAME, SET_UVALUE, SET_VALUE, XMANAGER_ACTIVE_COMMAND

WIDGET_DROPLIST() function

positional arguments: 1
keywords: DYNAMIC_RESIZE, EVENT_FUNC, EVENT_PRO, FONT, FRAME, FUNC_GET_VALUE, GROUP_LEADER, KILL_NOTIFY, NOTIFY_REALIZE, NO_COPY, PRO_SET_VALUE, RESOURCE_NAME, SCR_XSIZE, SCR_YSIZE, SENSITIVE, TAB_MODE, TITLE, TRACKING_EVENTS, UNAME, UNITS, UVALUE, VALUE, XOFFSET, XSIZE, YOFFSET, YSIZE

WIDGET_EVENT() function

positional arguments: 1
keywords: DESTROY, XMANAGER_BLOCK

WIDGET_INFO() function

positional arguments: 1
keywords: CHILD, MANAGED, MODAL, VALID, VERSION, XMANAGER_BLOCK

WIDGET_LABEL() function

positional arguments: 1
keywords: ALL_EVENTS, CONTEXT_EVENTS, EDITABLE, EVENT_FUNC, EVENT_PRO, FONT, FRAME, FUNC_GET_VALUE, GROUP_LEADER, IGNORE_ACCELERATORS, KBRD_FOCUS_EVENTS, KILL_NOTIFY, NOTIFY_REALIZE, NO_COPY, NO_NEWLINE, PRO_SET_VALUE, RESOURCE_NAME, SCROLL, SCR_XSIZE, SCR_YSIZE, SENSITIVE, TAB_MODE, TRACKING_EVENTS, UNAME, UNITS, UVALUE, VALUE, WRAP, XOFFSET, XSIZE, YOFFSET, YSIZE

WIDGET_TEXT() function

positional arguments: 1
keywords: ALL_EVENTS, CONTEXT_EVENTS, EDITABLE, EVENT_FUNC, EVENT_PRO, FONT, FRAME, FUNC_GET_VALUE, GROUP_LEADER, IGNORE_ACCELERATORS, KBRD_FOCUS_EVENTS, KILL_NOTIFY, NOTIFY_REALIZE, NO_COPY, NO_NEWLINE, PRO_SET_VALUE, RESOURCE_NAME, SCROLL, SCR_XSIZE, SCR_YSIZE, SENSITIVE, TAB_MODE, TRACKING_EVENTS, UNAME, UNITS, UVALUE, VALUE, WRAP, XOFFSET, XSIZE, YOFFSET, YSIZE

WINDOW procedure

positional arguments: 1
keywords: COLORS, FREE, PIXMAP, RETAIN, TITLE, XPOS, XSIZE, YPOS, YSIZE

WRITEU procedure

positional arguments: any number
keywords: TRANSFER_COUNT

WRITE_BMP procedure

positional arguments: 5
keywords: DEBUG, FOUR_BIT, HEADER_DEFINE, HELP, IHDR, RGB, TEST

WRITE_GIF procedure

positional arguments: 5
keywords: BACKGROUND_COLOR, CLOSE, DEBUG, DELAY_TIME, DISPOSAL_METHOD, HELP, MULTIPLE, REPEAT_COUNT, TEST, TRANSPARENT, USER_INPUT

WRITE_JPEG procedure

positional arguments: 2
keywords: DEBUG, HELP, ORDER, PROGRESSIVE, QUALITY, TEST, TRUE, UNIT

WRITE_PICT procedure

positional arguments: 5
keywords: DEBUG, HELP, TEST

WRITE_PNG procedure

positional arguments: 5
keywords: DEBUG, HELP, ORDER, TEST, TRANSPARENT, VERBOSE

WSET procedure

positional arguments: 1
keywords: none

WSHOW procedure

positional arguments: 2
keywords: none

WTN() function

positional arguments: 2
keywords: COLUMN, DOUBLE, INVERSE, OVERWRITE

XYOUTS procedure

positional arguments: 3
keywords: ALIGNMENT, CHARSIZE, CHARTHICK, CLIP, COLOR, DATA, DEVICE, NOCLIP, NORMAL, ORIENTATION, WIDTH, Z

Part II

Developer's guide

Chapter 16

General remarks and coding guidelines

... such as the CERN C++ Coding Standard Specification [4] or other similar documents.

Chapter 17

The library-routine API

TODO: extract it using Doxygen or some similar tool.

Chapter 18

Extending the documentation

LaTeX

gdldoc.sty

Natbib:

Chapter 19

Extending the testsuite (testsuite/README)

The list of GDL routines to be executed during the make-check run is defined in the testsuite/Makefile.am file. After adding a new item (filename) to the list, please rerun "automake" being in the root folder of the source tree. CMake also uses the list in Makefile.am.

Each test routine is invoked using the GDL "-e" command-line option by the "try" shell script in the testsuite directory (and in an analogous manner for the case of CMake/CTest). "make" decides on the status of a test basing on the exit code of this script:
- "success" for exit code 0
- "ignorable failure" for code 77
- "failure" for any other exit code, e.g. 1

The "try" script should, in principle, exit with the GDL exit code. Therefore, a failure of a GDL test should be indicated by e.g.:

```
  if ( ...true if test failed... ) begin
    message, 'reason for the failure', /continue
    exit, status=1
  endif
```

An ignorable failure can be indicated by e.g.:

```
  if (!XXX_exists()) then begin
    message, 'GDL was built w/o support for XXX - skipping', /conti
    exit, status=77
  endif
```

Any GDL error (e.g. parser error or library-routine-triggered error) causing GDL to return to the $MAIN$ level will cause make to assume _success_! (GDL exits normally in this case). Any GDL error causing GDL to stop execution on an other-than-$MAIN$ level will bring the GDL interpreter prompt.

The name of the file must match the name of the test routine, e.g. testsuite/test_dummy.pro for

```
  pro test_dummy
    ...
  end
```

GDL segfaults, assertion-exits, std::terminate() exits, etc. are handled as failures by make.

The "try" script always uses the gdl binary in the build tree - not the one installed in the system. The "try" script also sets appropriate env. variables so that the GDL-written library routines are taken from the source tree as well (e.g. src/pro/mean.pro).

Regardless if the autotools or the CMake/CTest configuration mechanism, the testsuite run is invoked by "make check" (not the default CMakes's "make test").

Chapter 20

A short overview of how GDL works internally

Programs (*.pro files) or command line input is parsed (GDLLexer.cpp, GDLParser.cpp generated with ANTLR from gdlc.g). These results in an abstract syntax tree (AST) consisting of 'DNode' (dnode.hpp). This systax tree is further manipulated (compiled) with a tree parser (GDLTreeParser.cpp generated with ANTLR from gdlc.tree.g, dcompiler.hpp). Here the AST is splitted into the different functions/procedures and the DNode(s) are annotated with further information and converted to ProgNode(s). Then these compiled (ProgNode) ASTs are interpreted (GDLInterpreter.cpp generated with ANTLR from gdlc.i.g, dinterpreter.cpp).

Chapter 21

How to make use of OpenMP in GDL

Chapter 22

Notes for packagers

Optional features of PLplot and ImageMagick

The HDF4-netCDF conflict

Part III

Indices

Subject Index

Bibliography

[1] Fundation, F. S.: GNU General Public License, version 2, URL http://www.gnu.org/licenses/old-licenses/gpl-2.0.html, 1991.

[2] Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Alken, P., Booth, M., and Rossi, F.: GNU Scientific Library Reference Manual - Third Edition (v1.12), Network Theory Ltd., URL http://www.gnu.org/software/gsl/manual/, 2009. {7}

[3] Markwardt, C.: Non-linear Least-squares Fitting in IDL with MPFIT, in: Astronomical Society of the Pacific Conference Series, edited by Bohlender, D., Durand, D., and Dowler, P., vol. 411 of *Astronomical Society of the Pacific Conference Series*, URL http://cdsads.u-strasbg.fr/abs/2009ASPC..411..251M, 2009. {19}

[4] Paoli, S.: C++ Coding Standard Specification, Tech. rep., CERN European Laboratory for Particle Physics, URL http://pst.web.cern.ch/PST/HandBookWorkBook/Handbook/Programming/CodingStandard/c++standard.pdf, 2000. {95}

[5] Snyder, J.: Map projections–A working manual, Tech. Rep. 1395, U.S. Geological Survey, URL http://pubs.er.usgs.gov/djvu/PP/pp_1395.djvu, 1987. {57}

[6] van Rossum, G. and Fred L. Drake, J.: The Python Language Reference Manual, Network Theory Ltd., URL http://docs.python.org/reference/, 2006. {32}

[7] Wessel, P. and Smith, W. H. F.: A global, self-consistent, hierarchical, high-resolution shoreline database, J. Geophys. Res., 101, 8741–8743, doi:10.1029/96JB00104, 1996. {61}

[8] Wolcott, N. and Hilsenrath, J.: Tables of coordinates for Hershey's repertory of occidental type fonts and graphic symbols. A contribution to computer typesetting techniques., NBS special publication 424, National Bureau of Standards, 1975. {24}