

ERDAS Spatial Modeler Language Reference Manual

ERDAS IMAGINE™ V8.5

***ERDAS®, Inc.
Atlanta, Georgia***

Copyright © 2000 by ERDAS, ® Inc. All Rights Reserved.

Printed in the United States of America.

ERDAS Proprietary - Delivered under license agreement.
Copying and disclosure prohibited without express written permission from ERDAS, Inc.

ERDAS, Inc.
2801 Buford Highway, NE, Suite 300
Atlanta, Georgia 30329-2137 USA

Toll Free: 877/GO-ERDAS (463-7327)
Phone: 404/248-9000
Fax: 404/248-9400
User Support: 404/248-9777

www.erdas.com

Acknowledgments

Warning

All information in this document, as well as the software to which it pertains, is proprietary material of ERDAS, Inc., and is subject to an ERDAS license and non-disclosure agreement. Neither the software nor the documentation may be reproduced in any manner without the prior written permission of ERDAS, Inc.

Specifications are subject to change without notice.

Trademarks

ERDAS and ERDAS IMAGINE are registered trademarks; CellArray, ERDAS Stereo Analyst, IMAGINE Developers' Toolkit, IMAGINE OrthoBASE, IMAGINE OrthoMAX, IMAGINE Vector, and IMAGINE VirtualGIS are trademarks. Other brands and product names are the properties of their respective owners. IMAGINE OrthoBASE/ IMAGINE OrthoBASE Pro Version 8.5 1999-2001. .

Table of Contents

Section I

Introduction to SML	1
Introduction	1
Conventions	1
Words enclosed in < >	1
Strings	1
Numbers	1
Statements	2
Object Types	2
Data Types	2
Variables	3
Windows	3
Working Window	3
Setting Windows on Layers	3
Bin Functions	4
Purpose of Bin Functions	4
Example of a Bin Function	4
Types of Bin Functions	4
Default Bin Function for Output Data File Types	6
Modeler Language Statements	7
General Syntax	7
Statements	7
Comments	7
Case	8
Declaration Statements	9
Declaration Statement Format	9
SCALAR Declarations	10
TABLE Declarations	10
Associating Table Variables With Descriptors and Color Tables	11
Examples of Table Declarations	12
Associating Table Variables With Vector Attributes	13
MATRIX Declarations	14
RASTER Declarations	15
Using Files	15
File Parameters	17
Existence Parameters	17
Access Parameters	18
Data Type Parameters	18
Default Data Types	20
Layer Type Parameters	20
Interpolation Parameters	21
Window Specification	21
Area of Interest Specification	21
Statistics Parameters	22
Bin Function Specification	22

Table of Contents

Edge Extension Specification	23
Examples of Raster Declarations	23
Data Type Examples	25
VECTOR Declarations	25
Parameters	26
Window Specification	26
Area of Interest Specification	26
Cellsize Specification	27
Feature Type Specification	27
Rendering Method Specification	28
Expressions	29
Constants	29
Binary Constants	29
Integer Constants	29
Float Constant	29
Complex Constants	30
Color Constants	30
String Constants	31
Variable References	31
Using Operators and Functions	31
Table Subexpressions	32
Matrix Subexpressions	32
Raster Layer Stacks	33
Assignment Statements	35
Example Assignments	35
Data Type Assignment Compatibility	35
Object Type Assignment Compatibility	35
ASCII Input-Output Statements	37
SHOW Statement	37
READ Statement	37
WRITE Statement	37
VIEW Statement	38
Setting Windows	39
SET WINDOW	39
SET CELLSIZE	40
Other SET Statements	40
SET AOI	40
SET DEFAULT <datatype>	41
SET DEFAULT ORIGIN	41
SET DEFAULT INTERPOLATION	41
SET DEFAULT STATISTICS	42
SET TILESIZE	42
SET RANDOM SEED	42
QUIT Statement	43
Statement Blocks	43
Flow Control	45
Conditional Branching	45
Looping	45

Table of Contents

Macro Definitions	46
Running the Spatial Modeler	47
Running from IMAGINE	47
Model Maker	47
Running from the Command Line	47
Model Arguments	48
Command Line Options	49
Statistics Computation and Descriptor Column Output	50
Errors	51
Syntax Errors	51
Processing Errors	51
Common Causes of Errors	51
Standard Rules for Combining Different Types	53
Data Types	53
COLOR Data Type	54
Object Types	54
SectionII	
SML Function Syntax	57
Function Types	57
Point Functions	57
Neighborhood Functions	57
Global Functions	57
Zonal Functions	58
Layer Functions	58
Combination Functions	58
Modeler Function Categories	59
Analysis	61
CLUMP (Clump - Contiguity Analysis)	62
CONVOLVE (Convolution)	64
CORRELATION (Correlation Matrix From Covariance Matrix)	65
CORRELATION (Correlation Matrix From Raster)	66
COVARIANCE (Covariance Matrix)	68
DELROWS (Delete Rows from Sieved Descriptor Column)	70
DIRECT LOOKUP (Map Integer Values Through Lookup Table)	72
EIGENMATRIX (Compute Matrix of Eigenvectors)	73
EIGENVALUES (Compute Table of Eigenvalues)	74
HISTMATCH (Histogram Matching)	75
HISTOEQ (Histogram Equalization)	76
HISTOGRAM (Histogram)	78
LINEARCOMB (Linear Combination)	80
LOOKUP (Map Input Values Through Lookup Table Using Bin Function)	81
PRINCIPAL COMPONENTS (Principal Components)	82
RASTERMATCH (Raster Matching)	84
SIEVETABLE (Get Sieve Lookup Table)	86
STRETCH (Stretch)	87
Arithmetic	89
+ (Addition)	90
- (Subtraction)	91

Table of Contents

- (Negation)	92
* (Multiplication)	93
/ (Division)	94
MOD (Modulus)	95
! (Factorial)	96
Bitwise	97
& (Bitwise And)	98
(Bitwise Or)	99
^ (Bitwise Exclusive Or)	100
~ (Bitwise Not)	101
Boolean	103
AND (Logical And)	104
OR (Logical Or)	105
NOT (Logical NOT)	106
Color	107
COLOR (Create Color Scalar)	108
HUE (Get Hue from RGB)	109
IHSTOBLU (Get Blue from Intensity, Hue, and Saturation)	111
IHSTOGRN (Get Green from Intensity, Hue, and Saturation)	112
IHSTORED (Get Red from Intensity, Hue and Saturation)	113
IHSTORGB (Get Red, Green and Blue from Intensity, Hue and Saturation)	114
INTENS (Get Intensity from RGB)	115
RGBTOIHS (Get Intensity, Hue and Saturation from Red, Green and Blue)	116
SATUR (Get Saturation from RGB)	117
STACK (Convert FLOAT TABLE to COLOR SCALAR)	118
UNSTACK (Convert COLOR SCALAR to FLOAT TABLE)	119
Conditional	121
CONDITIONAL (Conditional)	122
EITHER...IF...OR....OTHERWISE (Select on Binary Test)	123
INDEX (Index - Find Matching Item on List)	124
PICK (Pick - Get nth Item on List)	125
Data Generation	127
MAPX (Create Raster Containing X Map Coordinates)	128
MAPY (Create Raster Containing Y Map Coordinates)	129
MATRIX (Create Matrix from List of Scalars)	130
MATRIX (Read Matrix from Kernel Library)	131
MATRIX SERIES (Create Matrix Containing 2-D Series)	132
PIXELX (Create Raster Containing Column Number)	133
PIXELY (Create Raster Containing Row Number)	134
RANDOM (Generate Random Values)	135
STACKLAYERS (Stack Raster Layers)	136
TABLE (Create Table from List of Scalars)	137
TABLE SERIES (Create Table Containing Series)	138
Descriptor	139
. (Map Raster Through Descriptor Column)	140
:: (Read Descriptor Column or Color Table)	141
Distance	143
CIRC (Test if Inside Unit Circle)	144
DIST (Distance)	145

Table of Contents

RECT (Rectangle)	146
SEARCH (Search - Proximity Analysis)	147
TRI (Triangle)	148
Exponential	149
EXP (Exponential)	150
LOG (Natural Logarithm)	151
LOG10 (Common Logarithm)	152
POWER (Raise to Power)	153
SQRT (Square Root)	154
Focal (Scan)	155
BOUNDARY (Boundary)	156
FOCAL DENSITY (Focal Density)	158
FOCAL DIVERSITY (Focal Diversity)	160
FOCAL MAJORITY (Focal Majority)	162
FOCAL MAX (Focal Maximum)	164
FOCAL MEAN (Focal Mean)	166
FOCAL MEDIAN (Focal Median)	168
FOCAL MIN (Focal Minimum)	170
FOCAL MINORITY (Focal Minority)	172
FOCAL RANK (Focal Rank)	174
FOCAL SD (Focal Standard Deviation)	176
FOCAL STANDARD DEVIATION (Focal Standard Deviation)	178
FOCAL SUM (Focal Sum)	180
Global	183
GLOBAL DIVERSITY (Global Diversity)	184
GLOBAL MAJORITY (Global Majority)	186
GLOBAL MAX (Global Maximum)	188
GLOBAL MEAN (Global Mean)	190
GLOBAL MEDIAN (Global Median)	192
GLOBAL MIN (Global Minimum)	194
GLOBAL MINORITY (Global Minority)	196
GLOBAL SD (Global Standard Deviation)	198
GLOBAL STANDARD DEVIATION (Global Standard Deviation)	200
GLOBAL SUM (Global Sum)	202
Matrix	205
MATDIV (Matrix Division)	206
MATINV (Matrix Inverse)	207
MATMUL (Matrix Multiplication)	208
MATRIXTOTABLE (Convert Matrix to Table)	209
MATTRANS (Matrix Transpose)	210
TABLETOMATRIX (Convert Table to Matrix)	211
Other	213
ABS (Absolute Value)	214
ANGLE (Angle)	215
BINARY (Convert to Binary)	216
CEIL (Ceiling)	217
COMPLEX (Convert to Complex)	218
CONJ (Complex Conjugate)	219
DELTA (Delta)	220
EVEN (Test if Even)	221

Table of Contents

FLOAT (Convert to Float)	222
FLOOR (Floor)	223
GAMMA (Gamma)	224
IMAG (Imaginary Part)	225
INTEGER (Convert to Integer)	226
INV (Multiplicative Inverse)	227
ODD (Test if Odd)	228
REAL (Real Part)	229
ROUND (Round)	230
SIGN (Sign)	231
SINC (Sinc)	232
STEP (Step)	233
TRUNC (Truncate)	234
WHOLE (Test if Whole Number)	235
Relational	237
EQ (Equality)	238
GE (Greater Than or Equal)	239
GT (Greater Than)	240
LE (Less Than or Equal)	241
LT (Less Than)	242
NE (Inequality)	243
=~ (Case Insensitive String Equality)	244
!~ (Case Insensitive String Inequality)	245
ISALLTRUE (Test for All Non-zero)	246
ISNONZERO (Test for Non-zero)	247
Size	249
CELLAREA (Area of Grid Cells)	250
CELLUNITS (Cell Size Units)	251
CELLX (X Cell Size)	252
CELLY (Y Cell Size)	253
LAYERHEIGHT (Height of Raster Layer)	254
LAYERWIDTH (Width of Raster Layer)	255
NUMCOLS (Number of Columns)	256
NUMLAYERS (Number of Layers)	257
NUMROWS (Number of Rows)	258
Stack	259
STACK DIVERSITY (Stack Diversity)	260
STACK MAJORITY (Stack Majority)	261
STACK MAX (Stack Maximum)	262
STACK MEAN (Stack Mean)	263
STACK MEDIAN (Stack Median)	264
STACK MIN (Stack Minimum)	265
STACK MINORITY (Stack Minority)	266
STACK SD (Stack Standard Deviation)	267
STACK STANDARD DEVIATION	268
STACK SUM (Stack Sum)	269
Statistical (Local)	271
DENSITY (Local Density)	272
DIVERSITY (Local Diversity)	273
MAJORITY (Local Majority)	274

Table of Contents

MAX (Local Maximum)	275
MEAN (Local Mean)	276
MEDIAN (Local Median)	277
MIN (Local Minimum)	278
MINORITY (Local Minority)	279
RANK (Local Rank)	280
SD (Local Standard Deviation)	281
STANDARD DEVIATION (Local Standard Deviation)	282
SUM (Local Sum)	283
String	285
CAT (Concatenate Strings)	286
LENGTH (Length of String)	287
LOWERCASE (Lowercase Conversion)	288
MATCHES (String Wildcard Match)	289
UPPERCASE (Uppercase Conversion)	290
Surface	291
ASPECT (Aspect)	292
DEGREE SLOPE (Degree Slope)	293
PERCENT SLOPE (Percent Slope)	294
RELIEF (Shaded Relief)	295
Trigonometric	297
ACOS (Arccosine)	298
ACOSH (Hyperbolic Arccosine)	299
ASIN (Arcsine)	300
ASINH (Hyperbolic Arcsine)	301
ATAN (Arctangent)	302
COS (Cosine)	303
COSH (Hyperbolic Cosine)	304
SIN (Sine)	305
SINH (Hyperbolic Sine)	306
TAN (Tangent)	307
TANH (Hyperbolic Tangent)	308
Zonal	309
SUMMARY (Cross Tabulation)	310
ZONAL DIVERSITY (Zonal Diversity)	311
ZONAL MAJORITY (Zonal Majority)	312
ZONAL MAJORITY COUNT (Zonal Majority Count)	313
ZONAL MAJORITY FRACTION (Zonal Majority Fraction)	314
ZONAL MAX (Zonal Maximum from Summary)	315
ZONAL MAX (Zonal Maximum from Two Rasters)	316
ZONAL MEAN (Zonal Mean from Summary)	317
ZONAL MEAN (Zonal Mean from Two Rasters)	318
ZONAL MEDIAN (Zonal Median)	319
ZONAL MIN (Zonal Minimum from Summary)	320
ZONAL MIN (Zonal Minimum from Two Rasters)	321
ZONAL RANGE (Zonal Range from Summary)	322
ZONAL RANGE (Zonal Range from Two Rasters)	323
ZONAL SD (Zonal Standard Deviation from Summary)	324
ZONAL SD (Zonal Standard Deviation from Two Rasters)	326
ZONAL STANDARD DEVIATION (from Summary)	328

Table of Contents

ZONAL STANDARD DEVIATION (from Two Rasters) 330

Section III

Indexes 333

 Index of Symbols 333

 Index of Keywords 334

Section I

Introduction to SML

Introduction

The **Spatial Modeler Language** is a script language that is designed for GIS modeling and image processing applications. The Spatial Modeler language allows you to define simple or complex processing operations outside of [Model Maker](#), the graphical user interface in the Spatial Modeler component. Models created with Model Maker can be edited with the Spatial Modeler Language. However, the models which are created or edited using the Spatial Modeler Language cannot be accessed from Model Maker.

Operations that you can perform with the Spatial Modeler include:

- mathematical operations on raster layers (adding, subtracting, multiplying, ratioing, or other image algebra functions)
- convolution filtering
- neighborhood analyses (analyzing a pixel based on the values of neighboring pixels)
- subsetting and mosaicking
- principal components analysis
- proximity analysis
- contiguity analysis
- descriptor table manipulation

Conventions

The following conventions are used in this On-Line Help document:

Words enclosed in < >

When you see words with < > around them, you need to substitute these words with the proper information. For example, when you see the following in this document:

```
SCALAR <name>;
```

you would substitute the actual name of the variable so that the actual statement syntax would be something like:

```
SCALAR scalefactor;
```

Strings

A string is a group of words or characters. You must use quotation marks to enclose a string.

Numbers

A number can either be a floating-point number or an integer and can either be positive or negative (i.e. 2.5, -2.5, 2 or -2).

Introduction to SML

Statements

A model consists primarily of one or more [statements](#). Each statement falls into one of the following categories:

- [Declarations](#) - define objects to be manipulated within the model
- [Assignments](#) - assign a value to an object
- [SHOW](#) and [VIEW](#) statements - allow you to see and interpret results from the model
- [SET](#) statements - define the scope of the model or establish default values used by the Spatial Modeler
- [Macro Definitions](#) - define substitution text associated with a macro name.
- [QUIT](#) statement - end execution of the model

The Spatial Modeler Language also includes [flow control](#) structures, so that you can use conditional branching and looping in your models and [statement block](#) structures, which cause a set of statements to be executed as a group.

Object Types

The basic entities which can be manipulated within the Spatial Modeler Language are called objects. Each object falls into one of the following categories:

- [SCALAR](#) - a single numeric value, color, or character string.
- [TABLE](#) - a series of numeric values, colors, or character strings. A table has one column and a fixed number of rows. Tables are typically used to store columns from a descriptor table or a list of values which pertains to the individual layers of a multi-layer image file. For example, a table with 4 rows could be used to store the maximum value from each layer of a 4-layer image file.
- [MATRIX](#) - a set of numbers arranged in a two-dimensional array. A matrix has a fixed number of rows and columns. Matrices may be used to store convolution kernels or the neighborhood definition used in neighborhood functions. They can also be used to store covariance matrices, eigenvector matrices, or matrices of linear combination coefficients.
- [RASTER](#) - a single layer or multi-layer array of pixel data. Rasters are typically used to contain and manipulate data from image files.
- [VECTOR](#) - vector data in either a vector coverage or annotation layer can be read directly into the modeler, converted from vector to raster, then processed similarly to raster data. The modeler cannot write to coverages or annotation layers.

The size of a RASTER is defined by a window, which will be discussed in [Windows](#).

Data Types

Each object within the Spatial Modeler stores data in one of the following data types:

- [BINARY](#) - either 0 (FALSE) or 1 (TRUE)
- [INTEGER](#) - integer values from -2,147,483,648 to 2,147,483,647 (signed 32-bit integer)
- [FLOAT](#) - floating point data (double precision)
- [COMPLEX](#) - complex data (double precision)
- [COLOR](#) - three floating point numbers in the range 0.0 to 1.0 representing intensity of red, green, and blue
- [STRING](#) - a character string

Variables

Variables are objects in the Spatial Modeler which have been associated with a name using a [Declaration statement](#). The declaration statement defines the data type and object type of the variable. The declaration may also associate a raster variable with certain layers of an image file or a table variable with a descriptor table. [Assignment statements](#) are used to set or change the value of a variable.

Windows

A **window** is used to define the size and resolution used for a raster object. A window is a rectangle defined by an upper left (x,y) coordinate pair and a lower right (x,y) coordinate pair. These coordinates may be in either map units for georeferenced data or pixel units for non-georeferenced data.

If the coordinates are in map units, the window also contains an x and y cell size which defines the resolution of each pixel. If the coordinates are in pixel units, the cell size is assumed to be 1 pixel.

Working Window

A window called the **Working Window** defines the size and resolution for all raster objects in the model. Each raster object, regardless of the size or resolution of the file with which it may be associated, is treated as having the size and resolution defined by the Working Window. When the file is read by the Spatial Modeler, the input data will be resampled, truncated, or padded with background values (normally 0) as needed so that the raster object matches the Working Window.

By default, the Working Window will be the union of the extents of all input layers in the model. The default cell size of the Working Window will be the minimum of the input cell sizes from all input layers.

The Working Window can be changed using the [SET WINDOW](#) and [SET CELLSIZE](#) statements.

Setting Windows on Layers

When you declare a variable which is associated with an existing file, you have the option of setting a window for the variable. If the file is georeferenced, you may set the window in either map or pixel coordinates. If a pixel coordinate window is specified for a georeferenced file, the pixel coordinates are converted to map coordinates, and the resulting map coordinate window is used. If the file is not georeferenced, you may set the window in pixel coordinates only.

If the Spatial Modeler tries to read data outside the window you have set, the background value will be read. If you do not set a window when declaring a variable, the effective window for each layer of the variable will be the extent of the corresponding layer in the image file.



Introduction to SML

Bin Functions

Purpose of Bin Functions

At the completion of a model, the Spatial Modeler will compute statistics and a histogram for each output raster layer. The histogram will be stored in a descriptor table for the output layer. The model may also output other descriptor columns to a descriptor table. See [Associating Table Variables With Descriptors and Color Tables](#).

Bins are used to group ranges of data values together for better manageability. [Histograms](#) and other descriptor columns for 1, 2, 4 and 8-bit data are easy to handle, since they contain a maximum of 256 rows. However, to have a row in a descriptor table for every possible data value in floating point, complex, and 32-bit data would yield an enormous amount of information.

Example of a Bin Function

Suppose we have a floating point data layer with values ranging from 0.0 to 1.0. We could set up a descriptor table with 100 rows with each row or bin corresponding to a data range of .01 in the layer.

The bins would look like the following:

Bin Number	Data Range
0	$X < 0.01$
1	$0.01 \leq X < 0.02$
2	$0.02 \leq X < 0.03$
.	
.	
.	
98	$0.98 \leq X < 0.99$
99	$0.99 \leq X$

Then, for example, row 23 of the histogram table would contain the number of pixels in the layer whose value fell between .023 and .024.

Types of Bin Functions

The **bin function** establishes the relationship between data values and rows in the descriptor table. There are four types of bin functions used in IMAGINE image layers:

- **DIRECT** - one bin per integer value. Used by default for 1, 2, 4, and 8-bit integer data, but may be used for other data types as well. The direct bin function may include an offset for negative data, or data in which the minimum value is greater than zero. For example, a direct bin with 900 bins and an offset of -601 and would look like the following:

Bin Number	Data Range
0	$X \leq -600.5$
1	$-600.5 < X \leq -599.5$
.	
.	
.	
599	$-2.5 < X \leq -1.5$
600	$-1.5 < X \leq -0.5$
601	$-0.5 < X < 0.5$
602	$0.5 \leq X < 1.5$
603	$1.5 \leq X < 2.5$
.	
.	
.	
898	$296.5 \leq X < 297.5$
899	$297.5 \leq X$

- **LINEAR** - establishes a linear mapping between data values and bin numbers, as in our first example, mapping the data range 0.0 to 1.0 to bin numbers 0 to 99.

The bin number is computed by:

$\text{bin} = \text{numbins} * (x - \text{min}) / (\text{max} - \text{min})$
if (bin < 0) bin = 0
if (bin >= numbins) bin = numbins - 1

where:

	bin	=	resulting bin number
	numbins	=	number of bins
	x	=	data value
	min	=	lower limit (usually minimum data value)
	max	=	upper limit (usually maximum data value)

Introduction to SML

- LOG - establishes a logarithmic mapping between data values and bin numbers.

The bin number is computed by:

$\text{bin} = \text{numbins} * (\ln (1.0 + ((x - \text{min})/(\text{max} - \text{min}))) / \ln (2.0))$
if (bin < 0) bin = 0
if (bin >= numbins) bin = numbins - 1

- EXPLICIT - explicitly defines mapping between each bin number and data range.



Explicit bin functions are not accessible from the current version of the Spatial Modeler Language. They are only accessible through the C Programmers' Toolkit.

Default Bin Function for Output Data File Types

- 1, 2, 4, and 8-bit integer layers:
Direct binning. Number of bins equals 2, 4, 16, and 256 respectively.
- 16 and 32-bit integer layers:
If the difference between the maximum and minimum data values is less than 256, uses direct binning with an offset of the minimum data value; number of bins is: maximum - minimum + 1

If the difference is greater or equal to 256, linear binning from min to max with 256 bins is used by default.
- Floating point layers:
Linear binning from min to max, 256 bins.
- Complex layers:
Linear binning from minimum magnitude to maximum magnitude, 256 bins. The complex values are converted to magnitude before computing bin number.

The declaration statement has options to override the default binning by setting the desired bin function type and number of bins for the output layer. See the section [Bin Function Specification](#) under Raster Declarations.

Modeler Language Statements

The Spatial Modeler Language includes several basic statements that you will use together to create models. These statements include:

- [Declaration](#) statements
- [Assignment](#) statements
- [Visual output](#) statements
- [SET](#) statements
- [Macro Definitions](#)
- [QUIT](#) statement

[Expressions](#) are used in building statements.

General Syntax

A model may contain an almost unlimited number of lines of text.

The indentation of the lines that are used in the example models here are used for clarity only—they are not necessary. However, if used, they will make your models more readable.

Statements

Each statement in a model makes one definition or calculation. A statement may occupy more than one line. A semicolon must mark the end of each statement. The following statement is legal:

```

YYYYYYYYYYYYY = max (
AAAAAAAAAAAAA ,
BBBBBBBBBBBBB ,
CCCCCCCCCCCCC ,
DDDDDDDDDDDD ,
EEEEEEEEEEEEEE ,
FFFFFFFFFFFFFFF ) + 127 ;

```



Most syntax errors are caused when the semicolon is left off the end of a line. The line that is missing the semicolon is the line preceding the one in which the error is detected. For example, if a semicolon is missing on line 7, the error will appear on line 8.

Comments

Any line that begins with a **#** character is considered a comment line, and will be ignored by the Modeler. An exception is the [Macro Definition](#) statement, which begins with **#define**. Comments can also be on lines with valid statements. Any text after a **#** is a comment (unless **#** is followed by **define**). Comment lines can be embedded anywhere within the model.

A comment does not need to end with a semicolon. However, if a comment is on the same line as a statement, the statement must end with a semicolon, and the **#** character must follow the semicolon.



Declaration Statements

A **declaration** statement establishes a variable in the model. It defines:

- the name and size of the variable,
- its data type and
- object type.

There are five types of declarations statements used in the Modeler:

SCALAR

TABLE

MATRIX

RASTER

VECTOR

Declaration Statement Format

The basic format of the declaration statement is:

`<datatype> <objecttype> <name> <sizedef> <other parameters>;`

where:

<datatype>

Data type, either BINARY, INTEGER, FLOAT, COMPLEX, COLOR, or STRING. If the data type is omitted, the data type defaults to INTEGER.

<objecttype>

Object type, either SCALAR, TABLE, MATRIX, RASTER, or VECTOR. If the object type is omitted, the object type defaults to SCALAR.

<name>

The name you specify for the variable. The name must start with a letter and consist entirely of letters, numbers, and the underscore character (_). You cannot use any of the [keywords](#) defined in the Spatial Modeler Language as variable names.

<sizedef>

Sets the number of rows in a table, the number of rows and columns of a matrix, or the number of layers in a raster variable. The **<sizedef>** may also include a origin specifier, which specifies what index is to be used to identify the first row, column, or layer. The origin is usually either 0 or 1.

The **<sizedef>** component is not used for scalars or vectors, and may be omitted in the declaration of other types. If the variable is associated with existing file data, the size will be derived from the file. Otherwise, the size will be set on the first assignment to this variable.

<other parameters>

May be used to associate a raster variable with an image file, or a table with a descriptor column, or set various parameters used when creating a new image file.

The **<other parameters>** used in associating rasters and tables with files are discussed in the following pages of this document.

SCALAR Declarations

SCALAR variables may be declared with one of the following statements:

```
<datatype> SCALAR <name>;
```

or

```
SCALAR <name>;
```

or

```
<datatype> name;
```

If **<datatype>** is omitted, the data type defaults to **INTEGER**. If a variable is declared without an object type, then it defaults to **SCALAR**.

TABLE Declarations

The basic format of a **TABLE** declaration is the following:

```
<datatype> TABLE <name> <sizedef>;
```

<datatype> and/or **<sizedef>** may be omitted. If **<datatype>** is omitted, data type defaults to **INTEGER**.

<sizedef> is one of the following:

```
[<size>]  
[<origin>:<size>]  
[<origin>:]
```

<origin> is an integer constant specifying the index for the first row of the table.

For example, if a table is declared as:

```
TABLE bob [0:7];
```

then **bob [0]** is the first row of the table and **bob [6]** is the last row.

If **<origin>** is omitted, the origin will be the default table origin. The pre-set default table origin is zero. The default table origin may be reset using the [SET DEFAULT ORIGIN TABLE](#) statement, or may be set using the [Preference Editor](#).



<origin> must be a constant.

Associating Table Variables With Descriptors and Color Tables

<size> is a numeric scalar expression specifying the number of rows in the table variable. If **<size>** is non-integer, it will be converted to integer. If **<size>** is omitted, the number of rows will be set either by the descriptor table size or by the first assignment to the variable. If the table is associated with a descriptor table as described in the next section, the size of the descriptor table may determine the size of the declared table.

TABLE variables may be associated in the declaration statement with a descriptor table column from an image file. The descriptor table column may already exist or be created by the model.

- **If the descriptor table column already exists**, the column is read into the table variable from the file. At the end of model execution, the table will be written back to the file with any new values that may have been assigned to the table by the model.
- **If the descriptor table column does not already exist**, the column will be created at the end of model execution and values written into the file.

See [Statistics Computation and Descriptor Column Output](#) for more information on writing descriptor table columns.

The format for declaring a table associated with a descriptor column or color scheme is:

```
<datatype> TABLE <name> <sizedef> DESCRIPTOR <raster-exp> ::
<descriptorname>;
```

or

```
COLOR TABLE <name> <sizedef> COLORTABLE <raster-exp>;
```

<datatype> and **<sizedef>** are described in the previous section, [Basic Table Declarations](#).

<raster-exp> is a raster expression which must represent a single layer raster object associated with a layer of an image file. The layer may already exist, or it may be a layer which will be created by this model. The **<raster-exp>** must be a reference to a raster variable, or a single layer of a raster variable, i.e., **<raster-exp>** is either:

```
<raster-variable>
```

or

```
<raster-variable> (<layer-number>)
```

The **<layer-number>**, if present, must be an integer constant.

<descriptorname> is a string constant which specifies the name of the descriptor column to be associated with this variable. This may be an existing descriptor column, or the name of a new column to be created by the Modeler. If the descriptor column exists, the data type of the column must match the data type specified by the declaration. If the column does not exist, it will be created using the data type of this table variable.



*The current version of the Modeler always defaults to **INTEGER** type, regardless of the data type of the associated descriptor table, and the type of the variable must match the type of the descriptor column.*



If either **DESCRIPTOR** or **COLORTABLE** is present in the declaration, and the descriptor table exists for the layer associated with the **<raster-exp>**, the number of rows will be set to the number of rows in the descriptor table for that layer. Otherwise, the number of rows for the table will be deferred until the first assignment statement assigning a value to the variable name. At that point, the number of rows in the expression being assigned to this variable determines the size.

The **COLORTABLE** keyword may be used only with a table of **COLOR** data type. The values found in the “Red,” “Green,” and “Blue” descriptor columns will be read into this table, or if these columns do not exist, the Modeler will create them.

Only one table variable may be associated with each descriptor column for a single layer. If you try to associate more than one table variable with the same descriptor column of the same layer, an error occurs. Also, since the **COLORTABLE** keyword associates a table variable with the “Red,” “Green,” and “Blue” columns of a descriptor table, you may not associate another table variable with any of these three descriptor columns for the same layer.

Examples of Table Declarations



Please read the following sections before continuing with these examples:

- [RASTER Declarations](#)
- [Variable References](#)
- [Raster Layer Stacks](#)
- [Bin Function Specification](#)

Examples

```
TABLE tom;
```

This declares an **INTEGER TABLE** named **tom** of undefined size. The number of rows in **tom** will be defined when an assignment is made to **tom**.

```
RASTER in FILE OLD INPUT "infile.img";  
STRING TABLE cnames DESCRIPTOR in :: "Class_Names";
```

This declares the table **cnames** and associates it with the **Class_Names** column of the descriptor table for the single layer in the existing file **infile.img**. The number of rows in **cnames** will be the number of bins in the descriptor table. The file **infile.img** must contain exactly one layer, or the Modeler will report an error.

```
RASTER in FILE OLD INPUT "mobbay.img";  
FLOAT TABLE hist DESCRIPTOR in (3) :: "Histogram";
```

This time, the table **hist** is associated with the histogram for layer 3 of **mobbay.img**.



*The current version of the Modeler always defaults to **INTEGER** type, regardless of the data type of the associated file data.*

```
RASTER out FILE NEW OUTPUT "newfile.img";  
STRING TABLE cnames DESCRIPTOR out :: "Class_Names";
```

In this example, **newfile.img** is a new file, so the descriptor table has not been created yet. Thus the size of **clnames** is undefined after the declaration, and will be defined by the first assignment to **clnames**. The descriptor table for **newfile.img** is not created until the statistics are computed at the end of model execution.

After the descriptor table is created, the **Class_Names** column is added, and the data in **clnames** is written to the column. If there are more rows in **clnames** than bins in the descriptor table, only the rows up to the number of bins are written out. If there are more bins in the descriptor table than rows in **clnames**, the remaining rows in the descriptor column will be initialized to "".

Also, note that the number of layers in the **RASTER** variable **out** was left undefined in its declaration. The declaration of **clnames** assumes that **out** has only one layer, and in fact actually defines the number of layers in **out** to be one.

```

FLOAT TABLE hist DESCRIPTOR out (3) :: "Histogram";

```

This time, however, the number of layers in **out** is undefined when **out (3)** is referenced in the declaration of **hist**. This causes an error to be reported.

```

RASTER in FILE NEW INPUT "infile.img";
COLOR TABLE clrtab COLORTABLE in;

```

This declares the color table **clrtab** to be associated with the Red, Green, and Blue columns of the descriptor table for the single layer in the existing file **infile.img**.

Associating Table Variables With Vector Attributes

TABLE variables may also be associated in the declaration statement with an attribute of a vector coverage or an annotation layer. This is very similar to associating a table with a descriptor column from a raster image file. For vector coverages, the attribute may already exist may or be created by the model:

- **If the attribute already exists**, the attribute data is read into the table variable from the file. At the end of model execution, the table will be written back to the attribute file with any new values that may have been assigned to the table by the model.
- **If the attribute does not already exist**, the attribute will be created at the end of model execution and values written into the file.

For annotation layers, only the Name and Description can be treated as attributes. Either of these may be read from and/or written into. No new attributes for annotation can be created.

The format for declaring a table associated with a attribute is:

```

<datatype> TABLE <name> <sizedef> ATTRIBUTE <vector-variable>
:: <attributename>;

```

<datatype> and **<sizedef>** are described in the section [Basic Table Declarations](#).

The default data type is **INTEGER**, regardless of the type of attribute.

<vector-variable> is the variable name of a vector object. See [VECTOR Declarations](#).

<attributename> is a string constant which specifies the name of the attribute to be associated with this variable. For vector coverages, this may be an existing attribute, or the name of a new attribute to be created by the Modeler. For annotation layers, **<attributename>** must be either “Name” or “Description”;

If **ATTRIBUTE** is present in the declaration, and the attribute table exists for the coverage or layer associated with the **<vector-exp>**, the number of rows will be set to the number of rows in the attribute table. Otherwise, the number of rows for the table will be deferred until the first assignment statement assigning a value to the variable name. At that point, the number of rows in the expression being assigned to this variable determines the size.

Only one table variable may be associated with each attribute for a single coverage or layer. If you try to associate more than one table variable with the same attribute, an error occurs.

MATRIX Declarations

The basic form of a matrix declaration is:

```
<datatype> MATRIX <name> <sizedef>;
```

<datatype> and/or **<sizedef>** may be omitted. If **<datatype>** is omitted, data type defaults to **INTEGER**.



*Matrix objects may not be declared as **COLOR** or **STRING** data types.*

<sizedef> is one of the following:

```
[<rows>, <columns>]
[<roworigin>:<rows>, <columnorigin>:<columns>]
[<roworigin>:, <columnorigin>:]
```

<roworigin> and **<columnorigin>** are integer constants specifying the index for the first row and first column of the table. For example, if a matrix is declared as:

```
MATRIX biff [1:3, 0:4];
```

then the elements of the matrix would be arranged as:

biff [1,0]	biff [1,1]	biff [1,2]	biff [1,3]
biff [2,0]	biff [2,1]	biff [2,2]	biff [2,3]
biff [3,0]	biff [3,1]	biff [3,2]	biff [3,3]

If **<roworigin>** and **<columnorigin>** are omitted, the origin for both rows and columns will be the default matrix origin. The pre-set default table origin is zero. The default matrix origin may be reset using the [SET DEFAULT ORIGIN MATRIX](#) statement or may be set using the [Preference Editor](#).



***<roworigin>** and **<columnorigin>** must be constants.*

<rows> and **<columns>** are numeric scalar expressions specifying the number of rows and columns in the matrix variable. If **<rows>** or **<columns>** is non-integer, it will be converted to integer. If **<rows>** and **<columns>** are omitted, the number of rows and columns will be deferred until the first assignment statement assigning a value to the variable name. At that point, the number of rows and columns in the expression being assigned to this variable determines the size of the matrix.

RASTER Declarations

The most basic format of a raster declaration is:

```
<datatype> RASTER <name> <sizedef>;
```

<datatype> and/or **<sizedef>** are optional. If **<datatype>** is omitted, data type defaults to **INTEGER**.



*Raster objects may not be declared as **COLOR** or **STRING** data types.*

<sizedef> is one of the following:

```
( <size> )
( <origin> : <size> )
( <origin> : )
```

<origin> is an integer constant specifying the index for the first layer of the raster. For example, if a raster is declared as:

```
RASTER buffy (0:7);
```

then **buffy (0)** is the first layer of the raster and **buffy (6)** is the last layer.

If **<origin>** is omitted, the origin will be the default raster origin. The pre-set default raster origin is one. The default table origin may be reset using the [SET DEFAULT ORIGIN RASTER](#) statement or may be set using the [Preference Editor](#).



***<origin>** must be a constant.*

<size> is a numeric scalar expression specifying the number of layers in the raster variable. If **<size>** is non-integer, it will be converted to integer.

If **<size>** is omitted, the number of layers will be set by the first assignment to the variable. If the variable is associated with a raster file as described in the next section, the number of layers may be determined by the file.

Using Files

RASTER variables may be associated with one or more layers of an image file within the declaration statement. The image file may be either an already existing file, a new file, or new layers within a file which the Modeler will create. There is a variety of keywords which you may use in the declaration to control the data type and various other parameters used when creating a new image file.



The format for declaring a raster variable associated with a file is:

```
<datatype> RASTER <name> <sizedef> FILE <file parameters>
<layernames>;
```

<datatype> and **<sizedef>** are described in the previous section, [Basic Raster Declarations](#).

If **<size>** is omitted from **<sizedef>**, the number of layers will be set either by the number of layers specified in **<layernames>** or by the first assignment to the variable. If the **<layernames>** component is present in the declaration, the number of layers for the raster may be determined by **<layernames>**. (See the examples at the end of this section.) Otherwise, the number of layers for the raster will be deferred until the first assignment statement assigning a value to the variable name. At that point, the number of layers in the expression being assigned to this variable determines the size.

<file parameters> may be any combination of parameters which specify how new layers should be created, criteria to test against existing layers, or how layers are to be read. File parameters include:

OLD	64 BIT
NEW	128 BIT
DELETE_IF_EXISTING	SINGLE
INPUT	DOUBLE
OUTPUT	THEMATIC
INTEGER	CATEGORICAL
FLOAT	ATHEMATIC
COMPLEX	CONTINUOUS
SIGNED	NEAREST NEIGHBOR
UNSIGNED	BILINEAR INTERPOLATION
1 BIT	CUBIC CONVOLUTION
2 BIT	WINDOW <windowspec>
4 BIT	AOI <aoi-specification>
8 BIT	BIN <binfunctionspect>
16 BIT	EDGE REFLECT
32 BIT	EDGE FILL
U1	UNSIGNED_1_BIT
U2	UNSIGNED_2_BIT
U4	UNSIGNED_4_BIT
U8	UNSIGNED_8_BIT
U16	UNSIGNED_16_BIT
U32	UNSIGNED_32_BIT

S8	SIGNED_8_BIT
S16	SIGNED_16_BIT
S32	SIGNED_32_BIT
F32	FLOAT_SINGLE
F64	FLOAT_DOUBLE
C64	COMPLEX_SINGLE
C128	COMPLEX_DOUBLE

These parameters will be discussed in detail in the next section.

<layernames> is a **STRING** constant which may contain either the name of a file or the name of a file followed by a list of layers from the file.

For example:

```
" /usr/data/mobbay.img"
```

would specify all layers in the file **/usr/data/mobbay.img**.

```
" /usr/data/mobbay.img(:Layer_4,:Layer_2,:Layer_1)"
```

specifies layers 4, 2, and 1 (in that order) from **/usr/data/mobbay.img**.

If explicit layer names are included in the **<layernames>** component, and a size is specified in the declarations (with **<sizedef>**), the number of layers must match the number specified in the size. If no layer names are specified, the total number of layers in the file must match the specified size (from **<sizedef>**). See [Examples of Raster Declarations](#).



The Modeler will create temporary files for raster variables which are not associated with file layers. These temporary files will be deleted when the Modeler finishes executing the model.

*These temporary files will be created in the "Temporary File Directory" specified in the [Preference Editor](#). The default is **/tmp**. If there is not enough space in **/tmp** for these files, you may wish to change the directory in which the Modeler creates temporary files by changing the preference.*

File Parameters

The following keywords and parameters may be inserted in any order between the **FILE** keyword and the **<layernames>** parameter. Generally, these keywords specify how new files or layers are to be created, or test conditions on existing files.

Existence Parameters

Existence parameters specify whether the layers named in **<layernames>** are expected to already exist at the time the model is run. They include:

OLD	If OLD is present, the layers must already exist. Otherwise, an error is reported.
-----	--



NEW	If NEW is present, the layers must not exist. If they do, an error is reported.
DELETE_IF_EXISTING	If DELETE_IF_EXISTING is present, and the layers already exist, they will be deleted immediately, and then recreated by the model.

If none of the existence parameters is present, the Modeler will open the layers if they exist, or create them if they do not exist.

Access Parameters

Access parameters specify access to the layers. They include:

INPUT	Specifies that only read access is allowed to these layers. An error occurs if the model assigns a value to the associated variable. INPUT and NEW , or INPUT and DELETE_IF_EXISTING are incompatible.
OUTPUT	Specifies read and write access to layers.

If no access parameter is specified, read and write access are permitted.

Data Type Parameters

The **data type parameters** control which of the following data types is used for the layers specified in the declaration:

INTEGER	Specifies one of the integer data types.
FLOAT	Specifies one of the floating point data types.
COMPLEX	Specifies one of the complex data types.
SIGNED	Specifies a signed integer type.
UNSIGNED	Specifies an unsigned integer type.
1 BIT	1-bit unsigned integer.
2 BIT	2-bit unsigned integer.
4 BIT	4-bit unsigned integer.
8 BIT	8-bit integer data (signed or unsigned).
16 BIT	16-bit integer data (signed or unsigned).
32 BIT	32-bit integer data (signed or unsigned) or single precision float.
64 BIT	Specifies double precision float or single precision complex.
128 BIT	Specifies double precision complex.
SINGLE	Specifies single precision float or complex. FLOAT is used unless COMPLEX is specified or default is COMPLEX type.
DOUBLE	Specifies double precision float or complex. FLOAT is used unless COMPLEX is specified or default is COMPLEX type.

U1 UNSIGNED_1_BIT	1-bit unsigned integer.
U2 UNSIGNED_2_BIT	2-bit unsigned integer.
U4 UNSIGNED_4_BIT	4-bit unsigned integer.
U8 UNSIGNED_8_BIT	8-bit unsigned integer.
U16 UNSIGNED_16_BIT	16-bit unsigned integer data.
U32 UNSIGNED_32_BIT	32-bit unsigned integer data.
S8 SIGNED_8_BIT	8-bit signed integer.
S16 SIGNED_16_BIT	16-bit signed integer data.
S32 SIGNED_32_BIT	32-bit signed integer data.
F64 FLOAT_DOUBLE	Specifies double precision float.
C64 COMPLEX_SINGLE	Specifies single precision complex.
C128 COMPLEX_DOUBLE	Specifies double precision complex.

These data type parameters may be used together in any consistent combination. If any ambiguity about the data type persists after all data type parameters are specified, the default data types are used to resolve the ambiguity.

An error is reported if contradictory data type parameters are specified such as **SIGNED FLOAT**, **32 BIT COMPLEX**, or **SINGLE 16 BIT**. An error will also be reported if redundant data type parameters are used together such as **SIGNED S32**.

Layers of image files in IMAGINE may be any of the following data types:

1-bit unsigned integer
2-bit unsigned integer
4-bit unsigned integer
8-bit unsigned integer
8-bit signed integer
16-bit unsigned integer
16-bit signed integer
32-bit unsigned integer
32-bit signed integer
32-bit (single precision) floating point
64-bit (double precision) floating point
64-bit (single precision) complex
128-bit (double precision) complex

If the specified layers already exist, these parameters are checked against the data type of the layers. If the layer data type is incompatible with the data type parameter, an error is reported.

Default Data Types

If the specified layers do not exist, the data type of the variable defines the default data type for the new layers. The default data types are:

- **BINARY** variable: 1-bit unsigned integer
- **INTEGER** variable: 8-bit unsigned integer
- **FLOAT** variable: 32-bit (single precision) floating point
- **COMPLEX** variable: 64-bit (single precision) floating point

The default file data types for each modeler data type may be altered using the [SET DEFAULT <datatype>](#) statement or using the [Preference Editor](#).

Layer Type Parameters

Layer type parameters include:

THEMATIC
CATEGORICAL
ATHEMATIC
CONTINUOUS

These parameters identify layers as either thematic (categorical), or athematic (continuous). Various programs in IMAGINE will treat thematic and athematic data differently.

- **THEMATIC** or **CATEGORICAL** - use thematic layers
- **ATHEMATIC** or **CONTINUOUS** - use athematic layers
- **THEMATIC** and **CATEGORICAL** are incompatible with signed integer, floating point, and complex data, and these combinations will cause errors.

Interpolation Parameters

Interpolation parameters determine the resampling method that will be used if an existing layer is not the same resolution as the Working Window.

```
NEAREST NEIGHBOR
BILINEAR INTERPOLATION
CUBIC CONVOLUTION
```

The default interpolation is **NEAREST NEIGHBOR**. The default interpolation type may be changed using the [SET DEFAULT INTERPOLATION](#) statement or using the [Preference Editor](#). New layers are always the same resolution as the Working Window, so interpolation is not used for new layers.

Window Specification

The **window specification** sets a window on input layers. The format of the window specification is:

```
WINDOW <upper-left-x>, <upper-left-y> : <lower-right-x>
<lower-right-y> MAP
```

or

```
WINDOW <upper-left-x>, <upper-left-y> : <lower-right-x>
<lower-right-y> PIXEL
```

or

```
WINDOW <upper-left-x>, <upper-left-y> : <lower-right-x>
<lower-right-y> FILE
```

The coordinates in the window specifications must be constants. Window specifications are ignored for output layers.

See [Setting Windows](#) for more information about layer windows.

Area of Interest Specification

The **AOI specification** sets an area of interest on input layers. The format of the AOI specification is:

```
AOI <filename>
```

<filename> is a string constant containing the name of a file that contains an area of interest. When the data file is read, areas outside the AOI will be set to the background value. The AOI specification is ignored if used in the declaration of an output file.

```
AOI NONE
```

This indicates that no area of interest is to be used.



See also the [SET AOI](#) statement for setting an area of interest on a model. Setting an AOI on an input file rather than on the model changes when the AOI is applied. For example, using the SEARCH function on a layer with an AOI would cause the function to search only from search class pixels inside the AOI. On the other hand, SEARCH applied to a file without an AOI, but inside a model with an AOI, would search from all search class pixels in the Working Window. The AOI is then applied to the output of the SEARCH function.

Statistics Parameters

Statistics parameters determine which values in the data file will be used or ignored for the computing of final statistics or in [Global functions](#). Statistics parameters include:

```
USEALL <backgroundvalue>
```

Use all values present in the data file for computing statistics or global functions. **<backgroundvalue>** is optional and is not used if present.

```
IGNORE <backgroundvalue>
```

Ignore **<backgroundvalue>** when computing stats and global functions. **<backgroundvalue>** may be omitted, in which case zero values are ignored.

<backgroundvalue>, if present, must be a numeric constant.

The default for statistics is **USEALL**. The default statistics computation may be changed using the [SET DEFAULT STATISTICS](#) statement, or using the [Preference Editor](#).

Bin Function Specification

The **bin function specification** controls the bin function used in new output layers. The bin function specification may indicate:

- the type of bin function,
- the type and number of bins or type,
- number of bins, minimum and maximum.

If the bin function is completely specified, the descriptor table for each layer is created when the layer is created. If the bin function is only partially specified, the descriptor table is not created until the end of model execution, at which time statistics are computed, and the minimum and maximum used for the bin function are derived from the statistics.

See [Bin Functions](#) for an explanation of how bin functions are used.

The general format of a bin function specification is:

```
BIN <binfunctionspec>
```

However, the bin function specification may be any of the following formats:

```
BIN DIRECT DEFAULT
```

Use direct binning. The number of bins and offset are derived from statistics. The offset will be the minimum value (rounded to integer if necessary), the number of bins will be:

maximum (rounded if necessary) - minimum + 1

Exception: if THEMATIC is present, zero is used as the minimum, rather than the minimum file value. In this case the number of bins is:

maximum + 1

```
BIN DIRECT <numbins> BINS
```

Use direct binning, with an offset of zero, and **<numbins>** bins.

```
BIN DIRECT <numbins> BINS FROM <min> TO <max>
```

Use direct binning, offset **<min>**, **<numbins>** bins. **<numbins>** must equal **<max>** - **<min>** + 1.

```
BIN LINEAR <numbins> BINS
```

Use linear binning, **<numbins>** bins. Min and max are derived from statistics.

```
BIN LINEAR <numbins> BINS FROM <min> TO <max>
```

Use linear binning, **<numbins>** bins, **<min>** and **<max>** specified.

```
BIN LOG <numbins> BINS
```

Use logarithmic binning, **<numbins>** bins. Min and max are derived from statistics.

```
BIN LOG <numbins> BINS FROM <min> TO <max>
```

Use logarithmic binning, **<numbins>** bins, **<min>** and **<max>** specified.



<numbins>, **<min>** and **<max>** must be constants.

Edge Extension Specification

The **edge extension specification** specifies how the edge of the data file is to be handled by neighborhood functions. Since the focus or kernel used by the neighborhood function typically extends beyond the edge of the data, the function must generate data values for pixels outside the edge. The specification is:

```
EDGE REFLECT
```

or

```
EDGE FILL
```

EDGE REFLECT specifies that data file values should be reflected around the edge of the data file to generate pixels outside the edge. **EDGE FILL** specifies that pixels outside the edge of the data are given the background value. The default is **EDGE FILL**.

Examples of Raster Declarations

```
RASTER joe;
```



This declares a raster variable named **joe**, which is not associated with a file. The number of layers in **joe** will be determined when an assignment is made to **joe**.

```
RASTER bob (3);
```

This declares a raster variable named **bob** with three layers.

```
RASTER henry FILE OLD "/usr/data/mobbay.img";
```

This declares a raster variable named **henry**, which is associated with all the layers of the existing file **/usr/data/mobbay.img**. If **/usr/data/mobbay.img** does not exist, the file parameter **OLD** causes an error to occur. The variable **henry** has the same number of layers as the file **/usr/data/mobbay.img**.

```
RASTER bob (3) FILE OLD "/usr/data/mobbay.img";
```

This declares a raster variable named **bob**, which is associated with all the layers of the existing file **/usr/data/mobbay.img**. If **/usr/data/mobbay.img** does not exist, the file parameter **OLD** causes an error to occur. The variable **henry** is declared to have three layers. If **/usr/data/mobbay.img** does not have exactly three layers, an error occurs.

```
RASTER muddy FILE OLD  
"/usr/data/mobbay.img(:Layer_4,:Layer_2,:Layer_1)";
```

This declares a raster variable named **muddy**, which is associated with the listed layers of the existing file **/usr/data/mobbay.img**. The variable **muddy** will have three layers, since three layers were listed in the **<layernames>** component.

```
RASTER skipper (3) FILE OLD  
"/usr/data/mobbay.img(:Layer_1,:Layer_2)";
```

This will cause an error to occur, since **skipper** is declared to have three layers, but two layers were listed in the **<layernames>** component.

```
RASTER susie FILE NEW "/usr/data/newfile.img";
```

This will create a new image file named **/usr/data/newfile.img**. The number of layers in the file, and the number of layers in the variable **susie** will be determined when an assignment is made to the variable **susie**. If the file **/usr/data/newfile.img** already exists, the file parameter **NEW** will cause an error to occur.

```
RASTER jill FILE NEW  
"/usr/data/newfile.img(:Layer_1,:Layer_2)";
```

This declares the variable **jill** with 2 layers. If **:Layer_1** or **:Layer_2** is the name of an existing layer in the file **/usr/data/newfile.img**, an error occurs.

```
RASTER sam (3) FILE DELETE_IF_EXISTING  
"/usr/data/newfile.img";
```

This creates a new file named **/usr/data/newfile.img** with three layers. If there already existed a file called **/usr/data/newfile.img**, it is deleted first. Since no layer names are specified, the layers will be given the default names **:Layer_1**, **:Layer_2**, and **:Layer_3**.

Data Type Examples

Any data parameters in the declaration will modify the data type for the output layers based on the defaults. For example:

```
INTEGER RASTER a FILE NEW SIGNED "/usr/data/newfile.img";
```

Specifies that the layers of **/usr/data/newfile.img** will be 8-bit signed integer type.

You may specify any data type for the layers, regardless of the data type of the variable. For example, the following are valid declarations:

```
BINARY RASTER b FILE DOUBLE COMPLEX
"/usr/data/complexfile.img";
COMPLEX RASTER c FILE 1 BIT "/usr/data/binaryfile.img";
```

The data types are converted automatically when reading from or writing to the file layers.

VECTOR Declarations

Vector data in either a vector coverage or annotation layer can be read directly into the modeler, converted from vector to raster, then processed similarly to raster data. This is accomplished by declaring a VECTOR variable, which functions similarly to a read-only RASTER variable. The modeler cannot write to coverages or annotation layers.

VECTOR variables must be associated with an existing vector coverage or annotation layer within the declaration statement. The format for declaring a vector variable is:

```
<datatype> VECTOR <name> COVER <parameters> <covername>;
```

or

```
<datatype> VECTOR <name> ANNOTATION <parameters> <layername>;
```

<datatype> is optional. If **<datatype>** is omitted, data type defaults to **INTEGER**.



*Vector objects may not be declared as **COLOR** or **STRING** data types.*

Vector objects always have only one layer.

COVER indicates that **<covername>** is the name of a vector coverage. **ANNOTATION** indicates that **<layername>** is the name of a file containing an annotation layer.

<parameters> may be any combination of parameters which specify how vector layers are rasterized. Parameters include:



WINDOW <windowspec>
AOI <aoi-specification>
CELLSIZE <cellsizespec>
LINE
POINT
POLYGON
RENDER TO TEMPFILE
RENDER TO MEMORY



LINE, POINT and POLYGON parameters are used only for coverages, not annotation layers.

These parameters will be discussed in detail in the next section.

<covername> is a **STRING** constant which contains the name of a coverage.

<layername> is a **STRING** constant which contains the name of an annotation file.

Parameters

The following keywords and parameters may be inserted in any order between the **COVER** keyword and the **<covername>** parameter or between the **ANNOTATION** keyword and the **<layername>** parameter. Generally, these keywords specify how the vector data is to be rasterized.

Window Specification

The **window specification** sets a window on the input layer. The format of the window specification is:

```
WINDOW <upper-left-x>, <upper-left-y> : <lower-right-x>
<lower-right-y> MAP
```

The coordinates in the window specifications must be constants.

See [Setting Windows](#) for more information about layer windows.

Area of Interest Specification

The **AOI specification** sets an area of interest on the input layer. The format of the AOI specification is:

```
AOI <filename>
```

<filename> is a string constant containing the name of a file that contains an area of interest. When the data file is read, areas outside the AOI will be set to the background value. The AOI specification is ignored if used in the declaration of an output file.

```
AOI NONE
```

This indicates that no area of interest is to be used.

See also the [SET AOI](#) statement for setting an area of interest on a model. Setting an AOI on an input file rather than on the model changes when the AOI is applied. For example, using the SEARCH function on a layer with an AOI would cause the function to search only from search class pixels inside the AOI. On the other hand, SEARCH applied to a file without an AOI, but inside a model with an AOI, would search from all search class pixels in the Working Window. The AOI is then applied to the output of the SEARCH function.

Cellsize Specification

The **CELLSIZE specification** sets the default cell size to use in rendering the vector data. The format of the CELLSIZE specification is:

```
CELLSIZE <x-size> , <y-size> <units>
```

<x-size> and **<y-size>** are numeric constants.

<units> is either **METERS**, **FEET**, **INCHES**, **RADIANS**, or any other units listed in the file **\$IMAGINE_HOME/etc/units.dat**. Units and coordinates from input layers are converted to the units specified here, if necessary.

The vector data will always be rasterized to the cell size of the [Working Window](#). If there is a SET CELLSIZE statement in the model which sets an explicit cell size for the Working Window, this parameter will be ignored. Otherwise, this cell size specification is used in computing the Working Window cell size using the cell size rule. See [Setting Windows](#) for more info.

If there is no SET CELLSIZE statement specifying an explicit cell size, and no input RASTER layers to establish the cell size of the Working Window, a CELLSIZE specification is required in the VECTOR declaration to establish the cell size to use for the Working Window. If the Working Window cell size has not been established by the time the modeler attempts to read from the vector data, an error is reported.

Feature Type Specification

The **feature type specification** is one of the following:

```
LINE
POINT
POLYGON
```

The feature type specification determines which type of features are to be rasterized from a vector coverage. Feature type specifications may only be used with vector coverages, not with annotation layers. If this specification is not present the feature type to be rasterized is determined from the types of features present in the coverage.



Rendering Method Specification

The **rendering method specification** specifies how the vector data should be rendered by the modeler. The two options are

RENDER TO TEMPFILE

RENDER TO MEMORY

RENDER TO TEMPFILE specifies that the entire vector coverage or annotation layer will be rasterized into a temporary file up front by the modeler, and subsequently is treated as any other raster temp file. **RENDER TO MEMORY** specifies that the vectors or annotation are rendered tile by tile into memory without using any temporary disk space. Rendering tile by tile may be efficient enough when there is a relatively small number of relatively simple vector features to be rendered. However, if there is a large number of complicated features each with a large geographic extent, it is likely that rendering to a temporary file will be much more efficient, although it would require more disk space. If no rendering method is specified, **RENDER TO MEMORY** is used by default.



*The temporary files will be created in the “Temporary File Directory” specified in the [Preference Editor](#). The default is **/tmp**. If there is not enough space in **/tmp** for these files, you may wish to change the directory in which the Modeler creates temporary files by changing the preference.*

Expressions

Expressions are the basic building blocks of most Modeler statements. Expressions consist of constants and variables linked together by operators and functions. Every expression represents an object in the Modeler. An expression has a **data type** and an **object type**. The data type and object type for an expression are determined by the types of the constants and variables it is built from, together with the **Standard Rules** for combining types for each operation or function involved.



*It is possible to create expressions which have data and object type combinations which are not supported in variable **declarations**, such as color matrices and string rasters.*

Constants

Constants are **SCALAR** objects with a fixed value. Constants may be any data type. There are six types of constants:

- Binary
- Integer
- Float
- Complex
- Color
- String

Each type is explained in the following:

Binary Constants

TRUE	value of 1 or "true" logical value
DEFAULT	value of 1 or "true" logical value
FALSE	value of 0 or "false" logical value

Integer Constants

Any numeric value between -2,147,483,648 and 2,147,483,647 which does not contain a decimal point or scientific notation is an integer constant.

Examples:

601
-43

Float Constant

A numeric value containing a decimal point, in scientific notation, or outside the 32-bit signed integer range.



Examples:

```
1.  
-.000345  
4e3
```



PI is also recognized as a float constant **3.141592653589793**.

Complex Constants

Complex constants have the form (**<real>**, **<imaginary>**), where **<real>** and **<imaginary>** are float or integer constants.

Examples:

```
(1, 0)  
(.07, 4e13)
```

Color Constants

Color constants have the form (**<red>**, **<green>**, **<blue>**), where **<red>**, **<green>**, and **<blue>** are float or integer constants.

Examples:

```
(1, 1, 0)  
(.5, .3, .1)
```

The values for red, green, and blue should be in the range 0.0 to 1.0 when the color is going to be used in a color table by the Modeler. However, values outside this range are allowed.

For example, you can create color constants such **(255, 0, 255)** and **(128, 255, 0)**. If the 0-255 scale is used, at some point later in the model, the values should be divided by 255 before they are output to a color table.

Example:

```
RASTER out NEW OUTPUT "/usr/data/newfile.img";  
COLOR TABLE ct COLORTABLE out;  
.  
.  
.  
# initialize table using colors in 0-255 range  
ct = TABLE ((0,0,0),  
(255,255,0),  
(0,255,128),  
(240,120,120),  
(0,220,255) );  
.  
.  
.  
ct = ct / 255.;  
#divide by 255 so that output is in 0-1 range
```


String Constants

String constants are any sequence of characters enclosed in double quotes.

Examples:

```
"water"
"deciduous forest land"
```

Variable References

Variable references are expressions which retrieve the value stored in a variable. A variable reference is simply the variable name.

For example, if the variable **bob** is declared as:

```
INTEGER RASTER bob FILE OLD
"/usr/data/mobbay.img(:Layer_4,:Layer_2,:Layer_1)";
```

Then the expression:

```
bob
```

would cause data to be read from the specified layers of **/usr/data/mobbay.img** into the expression object.

In most cases, you may not use a variable of **undefined size** in an expression. A variable has undefined size if its size was not specified in its declaration, and no assignment has been made to the variable that would determine its size. There is one exception to this rule:

- in a [TABLE declaration](#) when you associate a table with a descriptor column that will be created by the model.

In this case, one layer is assumed.



Previous versions of the modeler required variable references to be prefaced with the \$ character. This is now optional. In the current version, any valid expression may be preceded by \$, which will be ignored.

Using Operators and Functions

Constants and variable references are combined in expressions using **operators** and **functions**. For example:

If **biff** is declared as

```
MATRIX biff [3, 4];
```

then

```
biff + 1
```

creates a new 3 row and 4 column matrix object with one added to the value of each element of **biff**.



If **sid** and **tom** are declared as:

```
TABLE sid [10];  
TABLE tom [10];
```

then

```
max (sid, tom)
```

creates a new 10 row table. The value in the first row of the new table will be the maximum of the first row of **sid** and the first row of **tom**, the second row contains the maximum of the second rows of **sid** and **tom**, and so forth.



See [Model Function Categories](#) for a description of each operator and function in the *Modeler*.

Table Subexpressions

Table subexpressions define new objects which are copied from portions of a table expression. Table subexpressions have one of the following forms:

```
<table-expression> [<index>]
```

This expression results in a **SCALAR** object.

```
<table-expression> [<start> : <end>]
```

This expression results in a **TABLE** object with **<end> - <start> + 1** rows.

<index>, **<start>**, and **<end>** are **SCALAR** numeric expressions. They are converted to **INTEGER** data type if necessary.

Examples:

If **sid** is declared as:

```
FLOAT TABLE sid [0:10];
```

then **sid** is a table with 10 rows numbered 0 to 9, and

```
sid [9]
```

will be a floating point **SCALAR**. The value will be copied from the last row of **sid**.

```
sid [0:3]
```

will be a floating point table with four rows copied from the first four rows of **sid**.



*You may not use a **TABLE** variable of undefined size in a subexpression.*

Matrix Subexpressions

Matrix subexpressions define new objects which are copied from portions of a matrix expression. Matrix subexpressions have one of the following forms:

```
<matrix-expression> [<row>, <column>]
```

This expression results in a **SCALAR** object.

```
<matrix-expression> [<startrow> , <startcolumn> : <endrow> ,
<endcolumn>]
```

This expression results in a **MATRIX** object with $\text{<endrow>} - \text{<startrow>} + 1$ rows and $\text{<endcolumn>} - \text{<startcolumn>} + 1$ columns.

<row>, **<column>**, **<startrow>**, **<startcolumn>**, **<endrow>**, and **<endcolumn>** are scalar numeric expressions. They are converted to **INTEGER** data type if necessary.

Examples:

If **fred** is declared as:

```
COMPLEX MATRIX fred [12, 9];
```

then

```
fred [6, 7]
```

will be a complex scalar.

```
fred [3, 4:8, 7]
```

will be a complex matrix with six rows and four columns.

You may not use a matrix variable of undefined size in a subexpression.

Raster Layer Stacks

Raster layer stacks define new objects which are copied from portions of a raster expression. Raster layer stacks have one of the following forms.

```
<raster-expression> (<layer-list>)
```

<layer-list> is a list of ranges of layer numbers separated by commas:

```
<layer-range>, <layer-range>, ...
```

Each **<layer-range>** may either be a single layer number or a **<startlayer>** and **<endlayer>** separated by a colon:

```
<layer>
```

or

```
<startlayer>:<endlayer>
```

<layer>, **<startlayer>**, and **<endlayer>** are scalar numeric expressions. They are converted to **INTEGER** data type if necessary.

The new raster will be a copy of the layers specified in layer list in the order listed.

Example:



If **biff** is declared as:

```
RASTER biff (12);
```

then

```
biff (8, 2:6, 1, 10:11)
```

will be a raster with 9 layers: layers 8, 2, 3, 4, 5, 6, 1, 10, and 11 of **biff** in that order.



You may not use a raster variable of undefined size in a raster layer stack.

Assignment Statements

An **assignment statement** is used to assign the value of an [expression](#) to a [variable](#). The basic form of an assignment statement is:

```
<variable> = <expression>;
```

You can also make assignment to [subexpressions](#) of tables and matrices, or to [raster layer stacks](#):

```
<table-variable> [<index>] = <expression>;
<table-variable> [<start> : <end>] = <expression>;
<matrix-variable> [<row>, <column>] = <expression>;
<matrix-variable> [<startrow> : <startcolumn> , <endrow> :
<endcolumn>] = <expression>;
<raster-variable> (<layer-list>) = <expression>;
```

Example Assignments

```
STRING TABLE bob [10];
bob = "hello, world";
bob [9] = "goodbye, cruel world";
```

These two assignment statements assign the string constant **"hello, world"** to all rows of the **table bob**, then the last row is changed to **"goodbye, cruel world."**

```
RASTER biff (12);
RASTER buff (9);
biff (8, 2:6, 1, 10:11) = buff;
```

This assignment copies all layers of **buff** to the layers listed for **biff**.

Data Type Assignment Compatibility

The following are the rules for assigning expressions of particular [data types](#) to variables:

- An expression may be assigned to a variable of the same data type (assuming object type compatibility - see next section).
- A numeric expression (**BINARY**, **INTEGER**, **FLOAT**, or **COMPLEX**) may be assigned to a numeric variable. Numeric conversion will be performed.
- A numeric expression may be assigned to a **COLOR** variable. The red, green, and blue components will all receive the same value.
- A **COLOR** expression may not be assigned to a numeric variable, with one exception noted in the next section.

STRING expressions may be assigned only to **STRING** variables.

Object Type Assignment Compatibility

The following are the rules for assigning expressions of particular [object types](#) to variables:

- An expression may be assigned to a variable (or subtable, submatrix, or raster layer stack) of the same object type and size.
- An expression may be assigned to a variable of the same object type whose size has not yet been defined. The assignment statement will define the size of the variable, and all subsequent assignments to this variable must conform to this size.



- A **SCALAR** expression may be assigned to a variable of any object type. All elements of the variable are assigned the same value.
- A **SCALAR** expression may be assigned to a variable of another object type of undefined size. The variable size will be set to one (i.e., one row for a table, one row and one column for a matrix, and one layer for a raster).
- A one layer raster expression may be assigned to a multiple layer raster variable. The single layer is copied to each layer of the variable.
- A table expression may be assigned to a raster variable which has the same number of layers as the table has rows. Each layer of the raster variable is filled with the value from the corresponding row of the table expression.
- A **COLOR SCALAR** expression may be assigned to a three layer **RASTER** variable. The red value of the expression fills the first layer of the variable, the green value fills the second layer, and the blue value fills the third.

Any assignment statement which does not conform to these rules will cause an error to be reported.

ASCII Input-Output Statements

The **SHOW**, **READ**, and **WRITE** statements allow you to input and output objects to and from ASCII format.

- The **SHOW** statement prints the values in a **SCALAR**, **MATRIX**, or **TABLE** object to standard output.
- The **READ** statement reads a **SCALAR**, **MATRIX**, or **TABLE** object from an input ASCII file.
- The **WRITE** statement writes a **SCALAR**, **MATRIX**, or **TABLE** object to an output ASCII file.

SHOW Statement

The **SHOW** statement has the form:

```
SHOW <expression>;
```

or

```
SHOW <expression>, <expression>, ... <expression>;
```

The contents of each expression object are printed to the standard output. If the Modeler is run from IMAGINE, the standard output is the Session Log. All expressions in a **SHOW** statement must be a **SCALAR**, **TABLE**, or **MATRIX** object type.

READ Statement

The **READ** statement reads a **SCALAR**, **MATRIX**, or **TABLE** object from an input ASCII file.

The form of the **READ** statement is:

```
READ <expression> FROM <filename>;
```

<expression> must be a scalar, matrix, or table object. If the size of the object has not yet been defined, it will be determined from the number of columns and lines of data in the ASCII file.

<filename> is a string constant containing the name of an ASCII file. The file should contain the same number of rows as the object. The individual elements of a single row should be separated by white space (spaces or tabs). There must not be any extra blank lines in the file, or an error will occur. Only one object must be contained in the file. **READ** statements cannot read multiple objects from the same file.

WRITE Statement

The **WRITE** statement writes a **SCALAR**, **MATRIX**, or **TABLE** object to an output ASCII file.

The form of the **WRITE** statement is:

```
WRITE <expression> TO <filename>;
```

<expression> must be a scalar, matrix, or table object.

<filename> is a string constant containing the name of an ASCII file. The file will contain the same number of rows as the object. The individual elements of a single row will be separated by spaces. Only one object can be written to an ASCII file using the **WRITE** statement. Any previous contents of the ASCII file will be deleted when a **WRITE** statement is encountered.



VIEW Statement



*The **VIEW** statement, which was supported in earlier versions of the modeling language is no longer supported.*

Setting Windows

By default, the [Working Window](#) at any point in a model will be the union of the windows for all layers from existing files associated with variables declared up to that point in the model. If input files are georeferenced, the Working Window will be the union of the map coordinate windows defined for each input layer. If any input file is georeferenced, all input files must be georeferenced using the same projection. The default cell size is the minimum of the cell sizes of all input layers.

If all input files are non-georeferenced, the default Working Window will be as wide as the widest input layer window and as long as the longest input layer window. The upper left corners of each input layer window are overlaid.

The **SET WINDOW** and **SET CELLSIZE** statements are used to change the Working Window. The formats of these statements are explained below.

SET WINDOW

```
SET WINDOW INTERSECTION;
```

If this is the first statement in the model, the Working Window will be set to the intersection of the windows from all existing layers declared in the model.

If existing input layers have already been declared, the current Working Window will already have been set to the union of the already declared layers' windows. After this statement, declaring any more input layers will intersect the Working Window with the new windows.

```
SET WINDOW UNION;
```

This statement sets the Working Window computation rule back to union, the default.

The “Default Window Rule” preference can be changed using the [Preference Editor](#) to change the default from Union to Intersection.

```
SET WINDOW <upper-left-x>,<upper-left-y>:<lower-right-x>,<lower-right-y> MAP;
```

This statement sets the Working Window to the map coordinate area explicitly specified in the statement. All input layers must be georeferenced to the same projection.

```
SET WINDOW <upper-left-x>,<upper-left-y>:<lower-right-x>,<lower-right-y> PIXEL;
```

or

```
SET WINDOW <upper-left-x>,<upper-left-y>:<lower-right-x>,<lower-right-y> FILE;
```

This statement sets the Working Window to the pixel coordinates explicitly specified in the statement. **<upper-left-x>**, **<upper-left-y>**, **<lower-right-x>**, and **<lower-right-y>** are all numeric constants. You may set pixel windows for georeferenced or non-georeferenced input layers. If the input layers are georeferenced, then the specified pixel window is applied relative to the pixels of the current Working Window, and converted into a map coordinate window.



*Coordinates used in the **SET WINDOW** statement must be constants.*

SET CELLSIZE

```
SET CELLSIZE MAX;
```

This statement causes the cell size for the working window to be computed from the maximum cell size of input layers rather than the minimum.

```
SET CELLSIZE MIN;
```

This statement resets the cell size computation rule to use the minimum input cell size, which is the default.

The “Default Cellsize Rule” preference can be changed using the [Preference Editor](#) to change the default rule for the Working Window cell size from Minimum to Maximum.

```
SET CELLSIZE <x-size> , <y-size> <units>;
```

Set the cell size for the Working Window to the specified size.

<x-size> and **<y-size>** are numeric constants.

<units> is either **METERS**, **FEET**, **INCHES**, **RADIANS**, or any other units listed in the file **/usr/imagine/etc/units.dat**. Units and coordinates from input layers are converted to the units specified here, if necessary.

All forms of the **SET CELLSIZE** statement may be used only with georeferenced input layers.



*Cell sizes in the **SET CELLSIZE** statement must be constants.*

All new output layers created by the Modeler have size and georeferencing information determined by the current Working Window at the time of their creation. Output layers are created when the first assignment is made to the variable associated with the output layers. You may output to existing layers, but only if the size and resolution of the existing layer exactly matches the Working Window. Unexpected results may occur otherwise.

Other SET Statements

SET AOI

This statement sets an area of interest (AOI) for the model.

```
SET AOI <filename>;
```

<filename> is a string constant containing the name of a file that contains an area of interest. This sets the area of interest for the model. All functions will return 0 for pixels that are inside the Working Window but outside the area of interest.

Note that setting the AOI does not affect the Working Window. To set the Working Window to the bounding rectangle of the AOI, you must determine the bounding area of the AOI using a cursor box in the Viewer, and then use a [SET WINDOW](#) statement to set the Working Window.

```
SET AOI NONE;
```

This indicates that no area of interest is to be used.

See also **Area of Interest Specification** for [RASTER](#) or [VECTOR](#) file declarations.

SET DEFAULT <datatype>

This statement has the form:

```
SET DEFAULT <datatype> <data-type-parameters>
```

<datatype> is either **BINARY**, **INTEGER**, **FLOAT**, or **COMPLEX**.

<data-type-parameters> is any valid combination of data type or layer type parameters used in raster declarations.

This statement resets the default file layer data type associated with raster variables of a particular Modeler data type, based on the pre-set defaults. For example:

```
SET DEFAULT INTEGER SIGNED;
```

The pre-set default file layer data type for **INTEGER** variables is 8-bit unsigned integer. This statement changes the default to 8-bit signed integer.

The pre-set defaults for each data type may be changed using the [Preference Editor](#).

SET DEFAULT ORIGIN

This statement has the form:

```
SET DEFAULT ORIGIN <objecttype> <origin>
```

<objecttype> is either **TABLE**, **MATRIX**, or **RASTER**.

<origin> is an integer constant (usually 0 or 1).

This statement resets the default origin for table rows, matrix rows and columns, or raster layers. The pre-set default for tables and matrices is 0. The pre-set default for raster layers is 1.

The pre-set defaults for the origin of each object type may be changed using the [Preference Editor](#).

SET DEFAULT INTERPOLATION

This statement has the form:

```
SET DEFAULT INTERPOLATION <interpolation-type>
```

or

```
SET DEFAULT INTERPOLATION <filetype> <interpolation-type>
```

where:

<interpolation-type> is either **NEAREST NEIGHBOR**, **BILINEAR INTERPOLATION**, or **CUBIC CONVOLUTION**.

<filetype> is either **THEMATIC** or **ATHEMATIC**.

This sets the default interpolation type to be used when reading from existing layers of different resolution than the Working Window. The **<filetype>** parameter controls whether this default is set for thematic or athematic files. If **<filetype>** is omitted, the specified interpolation type is set for both file types.



The pre-set default for both types is **NEAREST NEIGHBOR**. The pre-set default may be changed using the [Preference Editor](#).

SET DEFAULT STATISTICS

This statement allows you to set a background value which will be ignored when statistics or global functions are computed, or to specify that all values be used.

```
SET DEFAULT STATISTICS USEALL <backgroundvalue>
```

Use all values present for computing statistics or global functions. **<backgroundvalue>** is optional and is not used if present.

```
SET DEFAULT STATISTICS IGNORE <backgroundvalue>
```

Ignore **<backgroundvalue>** when computing stats and global functions. **<backgroundvalue>** may be omitted, in which case zero values are ignored.

<backgroundvalue>, if present, must be a numeric constant.

The pre-set default for statistics is **USEALL**. The pre-set default statistics computation may be changed using the [Preference Editor](#).

SET TILESIZE

Raster objects are processed by the Modeler in tiles. The tile size determines how many pixels of raster data are processed in each tile. The default tile size is 64 by 64 pixels. You may change the tile size using the **SET TILESIZE** statement:

```
SET TILESIZE <rows> , <columns>;
```

<rows> and **<columns>** are integer constants.

SET RANDOM SEED

The [RANDOM](#) function normally generates numbers from a seed derived from the current time, so that each sequence of numbers generated may be different. To guarantee that the same sequence of random numbers is generated each time a model is run, you may set a seed for the random number sequence using the **SET RANDOM SEED** statement:

```
SET RANDOM SEED <seed>;
```

<seed> is an integer constant.

Whenever the same seed is used, the same sequence of pseudo-random numbers will be generated by the **RANDOM** function.

QUIT Statement

After the Modeler executes the **QUIT** statement, it does not execute any more statements. The **QUIT** statement has the form:

```
QUIT;
```

After the **QUIT** statement is encountered, [statistics](#) are computed for all output files, descriptor tables associated with table variables are written out, and all files are closed.

Statement Blocks

Statement blocks collect a group of statements into a block to be executed together. Statement blocks are created by enclosing a group of statements in braces:

```
{
<statement>
<statement>
.
.
.
<statement>
}
```

The most common use for statement blocks is to group a set of raster [Assignment statements](#) into a block. The entire block will be executed tile-by-tile rather than individual statements being executed tile-by-tile. For example, consider the follow two models:

Model A:

```
RASTER in OLD INPUT "mobbay.img";
RASTER out1 NEW OUTPUT "out1.img";
RASTER out2 NEW OUTPUT "out2.img";

out1 = in + 5;
out2 = in * 2;

QUIT;
```

Model B:

```
RASTER in OLD INPUT "mobbay.img";
RASTER out1 NEW OUTPUT "out1.img";
RASTER out2 NEW OUTPUT "out2.img";

{
out1 = in + 5;
out2 = in * 2;
}

QUIT;
```

In model A, each 64 by 64 pixel tile is read from **mobbay.img**. Then 5 is added to each pixel in the tile, and the tile is written into **out1.img**. After this is complete, the tiling starts again at the beginning of **mobbay.img** and multiplies each tile by 2, and writes the output to **out2.img**.

In model B, each tile from **mobbay.img** is read only once. 5 is added to the tile and written to **out1.img**; then the tile values from **mobbay.img** are multiplied by 2 and written to **out2.img**. Since each tile from **mobbay.img** is read only once, rather than twice, model B runs faster than model A.



*Not every function can be executed tile-by-tile. **Point functions** are executed tile by tile. **Neighborhood functions** on existing layers are executed tile-by-tile. **Neighborhood functions** operating on intermediate results require a temporary file to be created beforehand. **Global, Zonal, and Layer functions** do not operate tile-by-tile. See individual operators and functions to determine function type.*

Variables declared inside a statement block are only defined within the statement block. The variable name will not be defined after the end of the statement block.

Generally, statement blocks are used for grouping together raster assignments or used in **flow control**. There are certain combinations of statements you should avoid putting into statement blocks:



Do not declare a variable whose size is a non-constant expression in the same statement block in which an assignment is made to that variable.

*Do not put **SET WINDOW**, **SET CELLSIZE**, **SET DEFAULT**, or **SET TILESIZE** statements in the same block as raster assignments.*

It is most efficient to group raster assignments sequentially inside a statement block, rather than alternating raster assignments with assignments to other object types, or other types of statements.

Failure to follow these guidelines can result in errors, unexpected results, or inefficient model execution.

Flow Control

Flow control is used to control the execution of a model, using conditional branching or looping.

Conditional Branching

The various forms of conditional branching are described below:

IF

```
IF (<binary-scalar-expression>) <statement-block>
```

<binary-scalar-expression> is a numeric **SCALAR** expression. If zero, it is treated as **FALSE**; nonzero is treated as **TRUE**.

<statement-block> is a set of statements enclosed in braces. (See [Statement Blocks](#).)

If the **<binary-scalar-expression>** is **TRUE**, the statements in **<statement-block>** are executed. If **FALSE**, they are skipped.

IF ... ELSE

```
IF (<binary-scalar-expression>) <statement-block-1> ELSE
<statement-block-2>
```

If the **<binary-scalar-expression>** is **TRUE**, the statements in **<statement-block-1>** are executed and **<statement-block-2>** is skipped. If **FALSE**, **<statement-block-1>** is skipped, **<statement-block-2>** is executed.

```
IF (<binary-scalar-expression-1>) <statement-block-1>
ELSE IF (<binary-scalar-expression-2>) <statement-block-2>
ELSE IF (<binary-scalar-expression-3>) <statement-block-3>
.
.
.
```

The statement block corresponding to the first **SCALAR** expression which is **TRUE** is executed. All other statement blocks are skipped. If none of the expressions is true, none of the statement blocks is executed.

```
IF (<binary-scalar-expression-1>) <statement-block-1>
ELSE IF (<binary-scalar-expression-2>) <statement-block-2>
ELSE IF (<binary-scalar-expression-3>) <statement-block-3>
.
.
.
ELSE <statement-block-N>
```

The statement block corresponding to the first scalar expression which is **TRUE** is executed. All other statement blocks are skipped. If none of the expressions is true, **<statement-block-N>** is executed.

UNLESS

```
UNLESS (<binary-scalar-expression>) <statement-block>
```

If the **<binary-scalar-expression>** is **FALSE**, the statements in **<statement-block>** are executed. If **TRUE**, they are skipped.

Looping

The forms of looping are described below:



WHILE

```
WHILE (<binary-scalar-expression>) <statement-block>
```

If **<binary-scalar-expression>** is **TRUE**, the statements in **<statement-block>** are executed. After execution, **<binary-scalar-expression>** is evaluated again. If **TRUE**, **<statement-block>** is executed again. This is repeated until **<binary-scalar-expression>** is **FALSE**.

UNTIL

```
UNTIL (<binary-scalar-expression>) <statement-block>
```

If **<binary-scalar-expression>** is **FALSE**, the statements in **<statement-block>** are executed. After execution, **<binary-scalar-expression>** is evaluated again. If **FALSE**, **<statement-block>** is executed again. This is repeated until **<binary-scalar-expression>** is **TRUE**.



*Note that the **<binary-scalar-expression>** in all flow control structures must be SCALAR. to make decisions based on other object types, use the [EITHER-IF-OR-OTHERWISE](#), [PICK](#), or [CONDITIONAL](#) functions.*

The statement block in a flow control structure must be syntactically correct, even if it is not executed. Parse-time errors may be reported from statements in a statement block, even though those statements will not be executed.

Macro Definitions

Macro definitions allow you to associate a string of text with a name. Subsequently, whenever the name is encountered in the model, it is replaced with the string of text. The syntax of a macro definition is:

```
#define <name> <replacement-text>
```

Any subsequent occurrences of **<name>** will be replaced with **<replacement-text>**. For example:

```
raster in1 file old "/usr/data/input1.img";
raster in2 file old "/usr/data/input2.img";
raster out file new "/usr/data/output.img";
float matrix kernel [3, 3];

kernel = 1. / 9.;

#define average ((in1 + in2) / 2)

out = average * 2 - convolve (average, kernel);
```

The last statement in this model is equivalent to:

```
out = ((in1 + in2) / 2) * 2 - convolve (((in1 + in2) / 2),
kernel);
```

The replacement text consists of all characters, including spaces, following **<name>** on the line starting with **#define**. To continue a macro definition onto the next line, end the line with a backslash (\).

Running the Spatial Modeler

Running from IMAGINE

Models can be written, edited, and run from the Spatial Modeler component or directly from the command line as described below. To write or edit models within IMAGINE, left-click the [Spatial Modeler](#) icon from the ERDAS IMAGINE icon panel and then left-click the **Script Librarian** option. You then have access to models generated from Model Maker and the Spatial Modeler Language.

Model Maker

Models written with [Model Maker](#) can be run or edited using the tools in Model Maker. You can also generate a script from Model Maker that will let you edit the model using the Spatial Modeler Language. This may be helpful if you have a general idea for a model and want to “sketch” it out graphically. You can then add to the model with the Modeler language by choosing the **Edit** option from the [Model Maker Menu](#).

This technique may also help you learn how to use the Spatial Modeler Language. By generating a script from within Model Maker, you can see the correct syntax for generating a particular model.



Although Model Maker is a powerful modeling tool, it does not include all the functionality available with the Spatial Modeler Language.

Running from the Command Line

You can run the Spatial Modeler directly from the command line. The syntax for the Spatial Modeler is:

```
modeler <model-file> <model-args> <options>
```

<model-file> The name of the text file containing the model.

<model-args> Arguments to be passed into the model. The model substitutes these arguments for the strings **ARG1**, **ARG2**, **ARG3**, etc., in the model as described below.

<options> Any of a set of options described below which control the Spatial Modeler execution.



Model Arguments

<model-args> is a list of arguments:

```
<argument1> <argument2> <argument3> ...
```

Each argument may be any string of characters. When run from the command line, an argument which contains a space should be enclosed in double quotes ("). The shell will remove the quotes before passing the argument to the Modeler.

The model may contain the strings **ARG1**, **ARG2**, **ARG3**, etc. The Spatial Modeler will substitute **<argument1>** for **ARG1** in the model, **<argument2>** for **ARG2**, and so forth. The string **ARGCOUNT** may also be used in the model. The integer constant representing the number of arguments passed in on the command line will be substituted for **ARGCOUNT**.

The Spatial Modeler will automatically enclose certain arguments in double quotes before passing them to the model, unless the **-nq option** is included on the command line. Any argument will be enclosed in quotes before being passed to the model, unless it meets one of the following conditions:

- The argument already starts and ends with the double quote character (").
- The argument is a valid floating point number.
- The argument contains only letters, numbers, or underscore (_) characters.
- The argument does not contain any letters, numbers, or underscore characters.

So, for example:

```
bob.img or 12*10
```

would be enclosed in quotes, while

```
34.2e12, bob, or +
```

would not be.

Example:

Suppose the file **binaryop.mdl** contains:

```
RASTER a FILE OLD INPUT ARG2;  
RASTER b FILE OLD INPUT ARG4;  
RASTER c FILE NEW OUTPUT ARG1;  
  
c = a ARG3 b;  
  
QUIT;
```

Then running the Spatial Modeler using the command line:

```
modeler binaryop.mdl out.img in1.img + in2.img
```

would result in this model being executed after substitutions were made:

```
RASTER a FILE OLD INPUT "in1.img";
RASTER b FILE OLD INPUT "in2.img";
RASTER c FILE NEW OUTPUT "out.img";

c = a + b;

QUIT;
```

Note that **ARG2**, **ARG4**, and **ARG1** were enclosed in quotes, while **ARG3** was not.

ARGCOUNT is useful for making sure that the model is executed with the correct number of arguments. In the previous example, we could add as the first line of the model:

```
IF (ARGCOUNT < 4) { QUIT; }
```

This would cause the model to stop executing after the first line if less than 4 model arguments were included on the command line.

Command Line Options

<options> may be any combination of the following:

```
-s or -state
-m or -meter
-nq
```

These are described below.

```
-s or -state
```

When run from the command line, the **-s** or **-state** option causes the Spatial Modeler to print to standard output a status message which indicates the current status of model execution.

```
-m or -meter
```

When run from the command line, the **-m** or **-meter** option causes the Spatial Modeler to print to standard output a message which indicates the progress of the current stage of model execution as a percent of total time for that stage. This message has the form:

```
% = <percent-done>
```

For example if the model **binaryop.mdl** in the previous example were run with the command line:

```
modeler binaryop.mdl out.img in1.img + in2.img -s -m
```

the following would be sent to standard output:

```
Initializing...
Processing points
% = 0 (increases to 100% as file is processed)
Computing stats, file: out.img
% = 0 (increases to 100% as statistics are computed)
All done
```



When you run from an [ERDAS Macro Language](#) macro, you can use the **-status** and **-meter** options to output the status and percent completion to a progress meter window.

-nq

The **-nq** option disables the automatic enclosure of [Model Arguments](#) in double quotes as described in the previous section.

Statistics Computation and Descriptor Column Output

At the completion of model execution, the Spatial Modeler will compute statistics and a [histogram](#) for each raster layer output by the model. The histogram will be stored in a descriptor table for the layer. The statistics for each layer include:

- min
- max
- mean
- standard deviation
- mode
- median

The histogram, mode, and median will depend on the bin function for the layer.



See [Bin Functions](#) and [Raster Declarations](#) for more information on bin functions and how to change them.

The statistics calculation will depend on whether or not you have specified a background value to be ignored during statistics calculation. You have the option of specifying [Statistics Parameters](#) when you declare the raster variable for the output layers, which specify whether or not a background value is to be ignored. If no statistics parameters were specified on declaration, the default statistics option is used. You can change the default statistics option using either the **SET DEFAULT STATISTICS** statement, or the [Preference Editor](#).

After the statistics and histogram have been written out, any tables which are associated with descriptor columns of the layer are also written out to the file. If the table has more rows than the number of bins in the descriptor table, only the rows up to the number of bins will be written out. If the table has less rows than the number of bins in the descriptor table, the remaining rows in the descriptor column will be set to **0** for numeric columns, or **" "** for **STRING** columns.

See [Associating Table Variables With Descriptors and Color Tables](#) for more information.

Errors

Syntax Errors

When the Spatial Modeler encounters an error, it prints an error message or list of error messages explaining what error occurred. It also prints the line number in the model, and the word, symbol, or string at which it recognized the error. This may be at or after where the actual problem in the model is located.

For example, suppose **newmodel.mdl** has as its first two lines:

```
RASTER a FILE "file1.img"
RASTER b FILE "file2.img";
```

The first statement is missing the terminating semicolon. The Spatial Modeler would print the following error:

```
***ERROR NUMBER 1 IN FUNCTION elox_Parse***
>>>Error in file newmodel.mdl, line 2
parse error: at or near [RASTER]
<<<
```

The error is reported as being at the beginning of the second line rather than at the end of the first line. This is because statements can be spread over more than one line in the model, so it would be OK if the second line started with a semicolon. However the second line starts with **RASTER** which is not what the Spatial Modeler expected, so it reports that the error occurred there.

Processing Errors

Sometimes, errors can occur during the execution of a model rather than during the parsing of the model. In this case, the model will be syntactically correct, but some error occurs during processing. The Spatial Modeler will report the error as being at the end of the statement or statement block in which the error occurred. If you have trouble identifying the statement in a statement block which was responsible for the error, you may need to remove the statements from the statement block, so that each statement is executed individually, to determine the problem statement.

When the Spatial Modeler is run from IMAGINE, the error messages are displayed in the Session Log. When run from the command line, the error messages are printed to standard output.

Common Causes of Errors

Some common causes of errors are listed below.

- Missing semicolon at the end of a statement.
- Mismatched parentheses.
- Mixing up parentheses and brackets. Parentheses must be used for [raster size declarations](#) and [raster layer stacks](#). Brackets must be used for [TABLE](#) and [matrix declarations](#) and [subexpressions](#).
- [Subexpression](#) index or [layer number](#) out of valid range. Check the [origin](#) and size for the table, matrix, or raster.
- Layers specified as [NEW](#) already exist.
- Attempting to use a file of incorrect format as input, or a corrupted or incomplete file.



- Out of disk space for output images.
- Out of disk space in **/tmp** directory. You may be able to prevent this error by changing the “Temporary File Directory” preference in the [Preference Editor](#).
- Wrong data type or object type for function or operation.
- Invalid mixture of object types as inputs to a function.
- [Assignment](#) of expression of one object type or size to an incompatible object type or size.
- Using variable with [undefined size](#) in an expression.
- Wrong number of arguments to a function.
- Mixing input files which are not georeferenced to the same projection type, or mixing georeferenced and non-georeferenced files.
- Misspelled name for [descriptor columns](#).
- Using a [reserved keyword](#) as a variable name.
- Using non-scalar expressions where a scalar expression is required.
- Using an [expression](#) where a constant is required.
- Attempting to use a variable declared inside a [statement block](#) outside that statement block.
- Parse-time errors in statements inside the statement block of a [flow control structure](#). The statement block must parse correctly, even if the condition for the control structure is such that the statement block will not be executed.
- [Data type](#) for **CLUMP** output file is too limited for output data range. Try a larger data type, preferably 32-bit unsigned.

Standard Rules for Combining Different Types

Most operators and functions can take inputs of different data and object types. Many of the operators and functions follow a set of standard rules for combining different data and object types, which are described here.



These rules apply to both [Model Maker](#) and the [Spatial Modeler Language](#). However, some capabilities mentioned may be accessible only through the [Spatial Modeler Language](#).



These rules are very similar to the rules governing assigning an expression to a variable of a different type described in [Assignment Statements](#).

Data Types

In general, most functions and operators will accept a mixture of numeric [data types](#) as inputs. The inputs are “promoted” to the maximum numeric data type present, in the order **BINARY**, **INTEGER**, **FLOAT**, **COMPLEX**. This maximum data type is considered the input data type for the function.

For example, if a function has two inputs, one **BINARY** and one **COMPLEX**, the **BINARY** input will be converted to **COMPLEX**, and **COMPLEX** will be the input data type for the function. If an operator or function has an input of type **COLOR**, the inputs are promoted in the same way, in the order **BINARY**, **INTEGER**, **FLOAT**, **COLOR**.



*You should not combine **COLOR** and **COMPLEX** inputs into a function, since this will usually result in an undefined output data type.*

Most functions will not accept a combination of numeric and **STRING** inputs.

For most operators and functions, the Data Types section includes a chart which shows the output data type for each input data type to the function. For example:

Input	Output
BINARY	not supported
INTEGER	FLOAT
FLOAT	FLOAT
COMPLEX	COMPLEX
COLOR	COLOR
STRING	not supported

Now using the promotion rules above and this chart, the following can be determined:

- If the inputs are all **INTEGER**, this function returns **FLOAT**.
- If the inputs are **BINARY** and **COMPLEX**, this function returns **COMPLEX**.
- If the inputs are all **BINARY**, an error is reported.
- If any of the inputs is **STRING**, an error is reported.



In this chart, “not supported” means that an error will be reported if this data type is input into this function. “Undefined” means that the return data type is not one of the defined data types.

COLOR Data Type

The **COLOR** data type is implemented as a “stack” of three **FLOAT**s. Any function which returns a **FLOAT** on **FLOAT** input will return **COLOR** for **COLOR** input by simply applying the function to the red, green, and blue components individually.

However, if a function returns anything other than **FLOAT** for **FLOAT** input, the output type for **COLOR** input is "undefined," or a data type that is not fully supported by the Modeler. For example, the Data Type chart for the "==" (test for equality) operator is :

Input	Output
BINARY	BINARY
INTEGER	BINARY
FLOAT	BINARY
COMPLEX	BINARY
COLOR	undefined
STRING	BINARY

Since **FLOAT** input returns **BINARY** output, **COLOR** input will return a "stack" of three **BINARY** values, which is not a supported data type. However, a few functions will accept this "undefined" data type and return a supported data type. The **ISALLTRUE** function will accept this type and return a **BINARY SCALAR**. Also, the **UNSTACK** function will accept the stack of 3 binary values and return a **BINARY TABLE** with 3 rows.

Object Types

The following are the standard rules for combining various **object types** as inputs to most operators and functions:

- Objects that are the same object type and size can be combined and will produce an output of that same object type and size. Each element of the first object will be combined with the element at the same row, column, and layer number of the second object to produce the corresponding element of the output object.
- Any object type may be combined with a **SCALAR**. The output object will be the type and size of the larger object. The **SCALAR** will be combined with every element of the larger object.
- A **RASTER** with multiple layers may be combined with a **RASTER** with a single layer. The output will be a **RASTER** with the same number of multiple layers. The function will combine the single layer input with each layer of the multiple layer input.
- A **TABLE** of any data type other than **COLOR** may be combined with a multiple layer **RASTER** which has the same number of layers as the **TABLE** has rows. The output will be the same size as the **RASTER**. The function combines each row of the **TABLE** with each element of the corresponding layer of the **RASTER**.

Standard Rules for Combining Different Types

- A **TABLE** of any data type other than **COLOR** may be combined with a single layer **RASTER**. The output will be a **RASTER** with as many layers as the **TABLE** has rows. The function combines each row of the **TABLE** with each element of the **RASTER** to produce the corresponding layer of the output **RASTER**.
- A **COLOR SCALAR** may be combined with a three layer **RASTER**. The output is a three layer **RASTER**. The red component of the **COLOR SCALAR** is combined with the first layer of the **RASTER**, the green with the second, and the blue with the third.

Any other combination of object types will cause an error to be reported for most functions.

Any function which does not conform to these rules will have its object type rules explicitly stated in the function description.





Section II

SML Function Syntax

Function Types

There are five function types in the Modeler:

- Point
- Neighborhood
- Global
- Zonal
- Layer

Point Functions

Point functions operate "point by point." The [standard rules](#) for object type combinations generally apply to most point functions. Point functions usually operate on any object type. Point functions operating on a **RASTER** consider the value of only one pixel in each input layer in determining the value for a single pixel of output. Point functions make up the majority of functions supported by the Modeler.

An example point function is:

```
MAX (<arg1>, <arg2>, <arg3>, ...)
```

Each element of the output of **MAX** is the maximum of the corresponding element in **<arg1>**, **<arg2>**, **<arg3>**, etc.

Neighborhood Functions

Neighborhood functions operate on a **RASTER** using a neighborhood which is defined in a **MATRIX**. Neighborhood functions involve each pixel's "neighbors," or nearby pixels, in the calculations.

An example neighborhood function is:

```
FOCAL MAX (<raster>, <focus>)
```

Each pixel in the output of **FOCAL MAX** is the maximum of the neighbors of the corresponding pixel in the input **<raster>**, using the **MATRIX <focus>** to define the size and shape of the neighborhood.

Global Functions

Global functions operate on an entire object or layer, and return a single value for that object or layer.

An example neighborhood function is

```
GLOBAL MAX (<arg1>)
```

If **<arg1>** is a **TABLE** or **MATRIX**, **GLOBAL MAX** returns the maximum value in the entire **TABLE** or **MATRIX**. If **<arg1>** is a **RASTER**, **GLOBAL MAX** returns a **TABLE** containing the maximum value of each entire layer in the **RASTER**.



SML Function Syntax

Zonal Functions

Zonal functions operate on two input **RASTER** layers, and return either a **MATRIX** or **TABLE**. One of the two input layers is called the zone layer, and the values of this layer (which must be unsigned integer data type) are called zones. The second input layer is called the class layer or value layer. Statistics are computed from the class or value layer for each zone in the zone layer, and the results are returned in the output **TABLE** or **MATRIX**. Each row of the returned object corresponds to one zone.

SUMMARY is a Zonal function which returns a **MATRIX**. Zonal functions which return a **TABLE** include the versions of the **ZONAL MAX**, **ZONAL MIN**, **ZONAL MEAN**, **ZONAL RANGE**, and **ZONAL SD** which have two raster layers as input. The other versions of these functions use the output of **SUMMARY** as input, and so are actually Point functions rather than Zonal functions, since their input is a **MATRIX**, not two **RASTER** layers.

Layer Functions

Layer functions operate on an entire **RASTER**. Layer functions output a **RASTER**. The value for each pixel in the output **RASTER** may depend on the arrangement of pixel values over the entire input layer.

Layer functions include **SEARCH** and **CLUMP**.

Combination Functions

Some functions in the modeler are actually combinations of simpler functions. An example is **PRINCIPAL COMPONENTS**, which combines **COVARIANCE**, **EIGENMATRIX**, **MATTRANS**, and **LINEARCOMB** into one function.

COVIARANCE is a Global function, while the others are Point functions. Other combination functions include **HISTOEQ**, **STRETCH**, and **RASTERMATCH**.

Modeler Function Categories

For your convenience, the Spatial Modeler Language functions and operators have been divided into several categories. These categories are listed below with a brief description of some of the capabilities included in each:

Analysis	Includes convolution filtering, histogram matching, contrast stretch, principal components, and more.
Arithmetic	Perform basic arithmetic functions including addition, subtraction, multiplication, division, factorial, and modulus.
Bitwise	Use bitwise and, or, exclusive or, and not.
Boolean	Perform logical functions including and, or, and not.
Color	Manipulate colors to and from RGB and IHS.
Conditional	Run logical tests using conditional statements and either...if...or...otherwise.
Data Generation	Create raster layers from map coordinates, column numbers, or row numbers. Create a matrix or table from a list of scalars.
Descriptor	Read descriptor information and map a raster through a descriptor column.
Distance	Perform distance functions including proximity analysis.
Exponential	Use exponential operators including natural and common logarithms, power, and square root.
Focal (Scan)	Several neighborhood analysis functions are available including boundary, density, diversity, majority, mean, minority, rank, standard deviation, sum, and others.
Global	Perform global operations including diversity, maximum, mean, minimum, standard deviation, sum, and more.
Matrix	Matrix functions allow you to multiply, divide and transpose matrices, as well as convert a matrix to table and vice versa.
Other	A host of miscellaneous functions provide data type conversion, various tests, and other utilities.
Relational	Relational operators include equality, inequality, greater than, less than, greater than or equal, less than or equal, and others.
Size	Measure cell X and Y size, layer width and height, number of rows and columns, etc.
Stack	Perform operations over a stack of layers including diversity, majority, max, mean, median, min, minority, standard deviation, and sum.
Statistical	Local statistical operations include density, diversity, majority, mean, rank, standard deviation, and more.
String	Concatenate strings and convert text to upper or lower case.



Surface

Surface functions allow you to calculate aspect and degree or percent slope.

Trigonometric

Use common trigonometric functions including sine/arcsine, cosine/arccosine and tangent/arctangent, and hyperbolic arcsine, arccosine, cosine, sine and tangent.

Zonal

Perform zonal operations including summary, diversity, majority, max, mean, min, range, and standard deviation.

Analysis

[CLUMP](#) (*Clump - Contiguity Analysis*)

[CONVOLVE](#) (*Convolution*)

[CORRELATION](#) (*Correlation Matrix from Covariance Matrix*)

[CORRELATION](#) (*Correlation Matrix from Raster*)

[COVARIANCE](#) (*Covariance Matrix*)

[DELROWS](#) (*Delete Rows from Sieved Descriptor Column*)

[DIRECT LOOKUP](#) (*Map Integer Values Through Lookup Table*)

[EIGENMATRIX](#) (*Compute Matrix of Eigenvectors*)

[EIGENVALUES](#) (*Compute Table of Eigenvalues*)

[HISTMATCH](#) (*Histogram Matching*)

[HISTOEQ](#) (*Histogram Equalization*)

[HISTOGRAM](#) (*Histogram*)

[LINEARCOMB](#) (*Linear Combination*)

[LOOKUP](#) (*Map Input Values Through Lookup Table Using Bin Function*)

[PRINCIPAL COMPONENTS](#) (*Principal Components*)

[RASTERMATCH](#) (*Raster Matching*)

[SIEVETABLE](#) (*Get Sieve Lookup Table*)

[STRETCH](#) (*Stretch*)

For more information see [Standard Rules](#).



CLUMP (Clump - Contiguity Analysis)

Syntax: `CLUMP (<raster>, <neighborcount>)`

Function Type: [Layer](#)

Description: Performs a contiguity analysis on **<raster>**, a single layer **RASTER**. Each separate raster region, or clump, is recoded to a separate class. The output is a single layer raster in which the contiguous areas are numbered sequentially. **<neighborcount>** is a **SCALAR** which determines the number of neighbors around a pixel used for determining contiguity. The only currently supported values for **<neighborcount>** are 4 and 8.

Data Types:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	INTEGER	FLOAT inputs converted to INTEGER
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: **<raster>** is a single layer **RASTER**. **<neighborcount>** is a **SCALAR**. The output is a single layer **RASTER**.

Example Model: `# This model produces a clumped output, then copies the class name for each # clump from the original class of the clump.`

```
RASTER in FILE OLD INPUT "/usr/imagene/examples/lnlandc.img";
RASTER cl FILE NEW OUTPUT 32 BIT UNSIGNED THEMATIC BIN DIRECT
DEFAULT
  "/usr/imagene/examples/clump.img";

STRING TABLE clnames DESCRIPTOR cl :: "Class_Names";

# do the clump
cl = CLUMP (in, 8);

# get class names for each clump
clnames = LOOKUP (cl :: "Original Value", in :: "Class_Names");

QUIT;
```

Notes: If the output from **CLUMP** is associated with a file layer, the layer should be declared at least 16-bit integer, or preferably 32-bit integer. The **CLUMP** function will use the output file layer to store temporary results. If the layer's data type is not sufficient to store the data range of the temporary results, an error is reported.

The **CLUMP** function will create a descriptor column called "Original Value" which contains the input file value for each clump in the output file. **CLUMP** is the only function in the Spatial Modeler Language which automatically generates descriptor columns, and after which the new descriptor columns are available to be used later in the model. All other descriptors columns created by models are created when the **QUIT** statement is encountered.

See the example for the **DELROWS** function. **SIEVETABLE** can be used to filter out small clumps from a layer processed with **CLUMP**.

See Also:

[SIEVETABLE](#)

[DELROWS](#)



CONVOLVE (Convolution)

Syntax: CONVOLVE (<raster>, <kernel>)

Function Type: [Neighborhood](#)

Description: Performs convolution on <raster> using <kernel> as convolution kernel.

Convolution filtering is a method of spatial filtering that analyzes pixels based on neighboring pixels. A convolution kernel is a matrix of numbers that is moved across the image to determine the output value of each pixel in the raster.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	not supported	
STRING	not supported	

Object Types: <raster> is a **RASTER**; <kernel> is a **MATRIX**.

Output is a **RASTER** with same number of layers as <raster>.

Notes: The output data are not normalized. To normalize, divide the <kernel> by the sum of its coefficients before using **CONVOLVE**. This can be done by using:

<kernel> / GLOBAL SUM (<kernel>)

Make sure that **GLOBAL SUM (<kernel>)** is non-zero to avoid division by zero.

See Also: [MATRIX](#)
[GLOBAL SUM](#)
[FOCAL functions](#)

CORRELATION (Correlation Matrix From Covariance Matrix)

Syntax: `CORRELATION (<covariance_matrix>)`

Function Type: [Point](#)

Description: Computes the correlation matrix from the covariance matrix.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	not supported	
STRING	not supported	

Object Types: `<covariance_matrix>` is a square **MATRIX**. The output is the same size as the input.

See Also: [CORRELATION](#) (from raster)

[COVARIANCE](#)



CORRELATION (Correlation Matrix From Raster)

Syntax:

```
CORRELATION (<raster>)  
  
or  
  
CORRELATION (<raster>, <ignoreoption>)  
  
or  
  
CORRELATION (<raster>, <ignoreoption> <backgroundvalue>)
```

Function Type:

Global

Description:

Returns the correlation matrix of **<raster>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value. If the pixel value for any of the **<raster>** layers is equal to the background value, that pixel is not included in the correlation calculation.

If **<ignoreoption>** is not present, the computation depends on whether **<raster>** is a previously existing file, or a **RASTER** created within the model. If **<raster>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<raster>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<raster>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	not supported	
STRING	not supported	

Object Types:

<raster> is a **RASTER**. The output is a square **MATRIX**. The number of rows and columns in the output is the same as the number of layers in **<raster>**.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

Notes:

If your model will be using both the covariance matrix and the correlation matrix, it is more efficient to calculate the covariance matrix first, using the **COVARIANCE** function, then use the **CORRELATION** from covariance matrix function, rather than using this function.

This function is equivalent to:

```
CORRELATION (COVARIANCE (<raster>, <ignoreoption>
<backgroundvalue>))
```

See Also:

[CORRELATION \(from covariance matrix\)](#)

[COVARIANCE](#)

[SET DEFAULT STATISTICS](#)



COVARIANCE (Covariance Matrix)

Syntax:

```
COVARIANCE (<raster>)  
  
or  
  
COVARIANCE (<raster>, <ignoreoption>)  
  
or  
  
COVARIANCE (<raster>, <ignoreoption> <backgroundvalue>)
```

Function Type:

Global

Description:

Returns the covariance matrix of **<raster>**. The covariance matrix is an $n \times n$ matrix containing all the variance and covariance within n bands of data. The covariance matrix can be used in matrix equations such as principal components analysis.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value. If the pixel value for any of the **<raster>** layers is equal to the background value, that pixel is not included in the covariance calculation.

If **<ignoreoption>** is not present, the computation depends on whether **<raster>** is a previously existing file, or a **RASTER** created within the model. If **<raster>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<raster>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<raster>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	not supported	
STRING	not supported	

Object Types:

<raster> is a **RASTER**. The output is a square **MATRIX**. The number of rows and columns in the output is the same as the number of layers in **<raster>**.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

[EIGENMATRIX](#)

[MATTRANS](#)

[PRINCIPAL COMPONENTS](#)

[SET DEFAULT STATISTICS](#)

[CORRELATION](#)



DELROWS (Delete Rows from Sieved Descriptor Column)

Syntax: `DELROWS (<dsctable>, <sievetable>)`

Function Type: [Point](#)

Description: **DELROWS** returns a **TABLE**, which is the same data type and size and **<dsctable>**. The table is initialized to zero if numeric, or "" if it is **STRING** type. Then, for each row **i** of the table:

```
if <sievetable> [i] > 0
then <result> [<sievetable> [i]] = <dsctable> [i]
```

DELROWS is typically used after the **SIEVETABLE** function. The output of **SIEVETABLE** is used as **<sievetable>**. A descriptor table from the input clumped layer whose histogram was input to **SIEVETABLE** would be used as **<dsctable>**. Then **DELROWS** outputs a table where the rows corresponding to the "sieved" values have been deleted. This allows you to copy the appropriate descriptor information from the clumped file to the sieved file.

Data Types: **<sievetable>** is **INTEGER**. **<dsctable>** may be any type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	STRING	

Object Types: **<sievetable>** and **<dsctable>** are **TABLE** type with the same number of rows. The output is also a **TABLE** with the same number of rows.

Example Model:

```
# This model uses a clumped file as input (see example model for
CLUMP), then # produces a sieved output using a threshold of 5.
The Original Values and
# Class Names are input to DELROWS to remove the descriptor table
rows for
# classes removed by the sieve.

RASTER cl FILE OLD INPUT "/usr/data/clump.img";
RASTER sv FILE NEW OUTPUT 32 BIT UNSIGNED THEMATIC BIN DIRECT
DEFAULT
  "/usr/data/sieve.img";

STRING TABLE clnames DESCRIPTOR cl :: "Class_Names";
STRING TABLE svnames DESCRIPTOR sv :: "Class_Names";
INTEGER TABLE svoriginal DESCRIPTOR sv :: "Original Value";

TABLE svtable;
INTEGER threshold; #threshold for sieve

threshold = 5;
# get the sieve table
svtable = SIEVETABLE (threshold, histogram (cl));

# do the sieve
sv = LOOKUP (cl, svtable);

# use DELROWS to get Original Value and Class Names for sieved file
svoriginal = DELROWS (cl :: "Original Value", svtable);
svnames = DELROWS (clnames, svtable);

QUIT;
```

See Also:

[CLUMP](#)
[SIEVETABLE](#)
[LOOKUP](#)



DIRECT LOOKUP (Map Integer Values Through Lookup Table)

Syntax: `DIRECT LOOKUP (<arg1>, <table>)`

Function Type: [Point](#)

Description: Map integer values in **<arg1>** through lookup table **<table>**. For example, if **<arg1>** is **SCALAR**, the result will be the value in row **<arg1>** of **<table>**, i.e., **<table>[<arg1>]** (assuming the origin for **<table>** is 0).

Data Types: **<arg1>** must be **INTEGER**.
The output will be the same data type as **<table>**:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	STRING	

Object Types: **<arg1>** may be any object type. **<table>** is a **TABLE**. The output will be the same object type and size as **<arg1>**.

Exception: If **<arg1>** is single layer **RASTER** and **<table>** is **COLOR**, output will be a three layer **RASTER**. If **<arg1>** is **RASTER** and **<table>** is **COLOR**, the number of layers in **<arg1>** must be one or three.

Notes: **DIRECT LOOKUP** differs from **LOOKUP** in that **LOOKUP** uses the bin functions associated with the lookup table and **DIRECT LOOKUP** does not.

See Also: [LOOKUP](#)

EIGENMATRIX (Compute Matrix of Eigenvectors)

Syntax: `EIGENMATRIX (<matrix1>)`

Function Type: [Point](#)

Description: The input must be a square matrix, typically the result of the **COVARIANCE** function. The output is the matrix of eigenvectors derived from the input matrix. An eigenvector matrix is often used in image processing functions such as principal components analysis.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	not supported	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

Object Types: The input must be a square **MATRIX**. The output will be a **MATRIX** the same size as the input.

See Also: [COVARIANCE](#)
[MATTRANS](#)
[LINEARCOMB](#)
[EIGENVALUES](#)
[PRINCIPAL COMPONENTS](#)



EIGENVALUES (Compute Table of Eigenvalues)

Syntax: `EIGENVALUES (<matrix1>)`

Function Type: [Point](#)

Description: The input must be a square matrix, typically the result of the **COVARIANCE** function. The output is the eigenvalues of the matrix returned as a table.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	not supported	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

Object Types: The input must be a square **MATRIX**. The output will be a **TABLE** with the same number of rows as the input.

See Also: [EIGENMATRIX](#)
[COVARIANCE](#)
[MATTRANS](#)
[LINEARCOMB](#)

HISTMATCH (Histogram Matching)

Syntax: `HISTMATCH (<hist1>, <hist2>)`

Function Type: [Point](#)

Description: Histogram matching is the process of determining a lookup table that will convert the histogram of one object to resemble the histogram of another object. The inputs are two histogram tables.

This function creates a lookup table. If the data used to generate **<hist1>** are passed through this lookup table, the histogram of the new data will approximate the shape of **<hist2>**.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	not supported	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: **<hist1>** and **<hist2>** are **TABLE**s. The output is a **TABLE** the same size as **<hist1>**.

Notes: For rasters, **HISTMATCH** is useful for matching data of the same or adjacent scenes which are slightly different due to sun angle or atmospheric effects.

See Also: [LOOKUP](#)
[HISTOGRAM](#)
[HISTOEQ](#)
[RASTERMATCH](#)
[Bin Functions](#)



HISTOEQ (Histogram Equalization)

Syntax:

```
HISTOEQ (<raster>, <bincount>)  
  
or  
  
GLOBAL MEAN (<raster>, <bincount>, <ignoreoption>)  
  
or  
  
GLOBAL MEAN (<raster>, <bincount>, <ignoreoption>  
<backgroundvalue>)
```

Function Type:

Combination ([Global](#) and [Point](#))

Description:

Computes histogram equalization of **<raster>** using **<bincount>** bins.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value.

If **<ignoreoption>** is not present, the computation depends on whether **<raster>** is a previously existing file, or a **RASTER** created within the model. If **<raster>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<raster>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<raster>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

<bincount> is numeric and is converted to **INTEGER**. **<raster>** may be any numeric type; the output is **INTEGER**:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	INTEGER	
COMPLEX	INTEGER	
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.

Object Types:

<raster> is a **RASTER**, **<bincount>** is a scalar. The result is a **RASTER** with the same number of layers as **<raster>**.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

[HISTMATCH](#)

[HISTOGRAM](#)

[LOOKUP](#)

[SET DEFAULT STATISTICS](#)



HISTOGRAM (Histogram)

Syntax:

```
HISTOGRAM (<arg1>)  
  
or  
  
HISTOGRAM (<arg1>, <ignoreoption>)  
  
or  
  
HISTOGRAM (<arg1>, <ignoreoption> <backgroundvalue>)
```

Function Type

: [Global](#)

Description:

Returns the histogram of **<arg1>**.

A histogram is a graph which represents data distribution. For a single band of data, the horizontal axis of the graph is the range of all possible data file values. The vertical axis is the number of pixels that have each value.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value. **<ignoreoption>** and **<backgroundvalue>** may be used only if **<arg1>** is a **RASTER** object.

If **<ignoreoption>** is not present and **<arg1>** is a **RASTER** object, the computation depends on whether **<arg1>** is a previously existing file, or a **RASTER** created within the model. If **<arg1>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<arg1>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<arg1>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

The type of **<arg1>** determines the output type:

Input	Output	Comments
BINARY	FLOAT	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.

Object Types:

<arg1> may be any object type. If **<arg1>** is a **RASTER**, **<arg1>** must have only one layer. The result is a **TABLE**. The number of rows in the table is determined by the bin function for the table. If **<arg1>** is a **RASTER** associated with a file layer, then the bin function from that file layer's descriptor table is used. Otherwise, the default bin function is used based on the data type of **<arg1>** and the minimum and maximum data value of **<arg1>**.

BINARY	Direct, offset = 0
INTEGER, max - min < 256	Direct, offset = min
INTEGER, max - min >= 256	Linear, 256 bins
FLOAT	Linear, 256 bins
COMPLEX	Linear, 256 bins, use magnitude

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**. **<ignoreoption>** and **<backgroundvalue>** may be present only if **<arg1>** is a **RASTER**.

See Also:

[HISTMATCH](#)

[SET DEFAULT STATISTICS](#)

[Bin Functions](#)

LINEARCOMB (Linear Combination)

Syntax: `LINEARCOMB (<raster>, <arg2>)`

Function Type: [Point](#)

Description: Computes linear combination of **<raster>** using **<arg2>** as a transformation matrix. For example:

```
RASTER x (3);  
RASTER y (2);  
MATRIX m [1:2, 1:3];  
y = LINEARCOMB (x, m);
```

is equivalent to:

```
y (1) = x (1) * m [1,1] + x (2) * m [1,2] + x (3) * m [1,3];  
y (2) = x (1) * m [2,1] + x (2) * m [2,2] + x (3) * m [2,3];
```

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	not supported	
STRING	not supported	

Object Types: **<raster>** is a **RASTER**. **<arg2>** must have as many columns as **<raster>** has layers. **<arg2>** is normally a **MATRIX**, but may be a **TABLE** or even **SCALAR** if **<raster>** has only one layer. The result is a **RASTER** with as many layers as **<arg2>** has rows.

LOOKUP (Map Input Values Through Lookup Table Using Bin Function)

Syntax: LOOKUP (<arg1>, <table>)

Function Type: [Point](#)

Description: If <table> has an associated bin function, the values in <arg1> will be converted to bin numbers, then the bin number will be used as an index into the lookup table <table>.

If <table> does not have an associated bin function, <arg1> is converted to **INTEGER**, and this number is used as an index to the lookup table.

Data Types: If <arg1> is **BINARY**, **INTEGER**, **FLOAT**, or **COMPLEX**, the output will be the same data type as <table>:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	STRING	

If <arg1> is **COLOR**, and <table> is **FLOAT** or **COLOR**, output will be **COLOR**. If <arg1> is **COLOR** and <table> is any other type, output is undefined type.

STRING type for <arg1> is not supported.

Object Types: <arg1> may be any object type. <table> is a **TABLE**. The output will be the same object type and size as <arg1>.

Exception: If <arg1> is single layer **RASTER** and <table> is **COLOR**, output will be a three layer **RASTER**. If <arg1> is **RASTER** and <table> is **COLOR**, the number of layers in <arg1> must be one or three.

Notes: Tables will have an associated [Bin Function](#) if they are associated with a descriptor column, or if they are the result of the [HISTOGRAM](#) or [HISTMATCH](#) functions. In either case the bin function for the descriptor table is used. All other table objects do not have associated bin functions.

See Also: [DIRECT LOOKUP](#) and [Map Raster through Descriptor Column](#)



PRINCIPAL COMPONENTS (Principal Components)

Syntax:

```
PRINCIPAL COMPONENTS (<raster>, <count>)  
or  
PRINCIPAL COMPONENTS (<raster>, <ignoreoption>)  
or  
PRINCIPAL COMPONENTS (<raster>, <ignoreoption> <backgroundvalue>)
```

Function Type:

Combination ([Global](#) and [Point](#))

Description:

Computes the first **<count>** principal components of **<raster>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation of the covariance matrix used in the principal components transform, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value. If the pixel value for any one of the layers of **<raster>** is equal to the background value, that pixel is not included in the covariance calculation.

If **<ignoreoption>** is not present, the computation depends on whether **<raster>** is a previously existing file, or a **RASTER** created within the model. If **<raster>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<raster>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<raster>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

<count> is numeric, and is converted to **INTEGER**.

The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.

Object Types:

<count> is **SCALAR**. **<raster>** is a **RASTER** and must have at least **<count>** layers.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

The output is **RASTER** with **<count>** layers.

Notes:

Equivalent to:

```
LINEARCOMB (<raster>, MATTRANS (EIGENMATRIX (COVARIANCE
(<raster>)) [0, 0: NUMLAYERS (<raster>) - 1, <count> -1]))
```

See Also:

[EIGENMATRIX](#)
[COVARIANCE](#)
[MATTRANS](#)
[LINEARCOMB](#)
[SET DEFAULT STATISTICS](#)



RASTERMATCH (Raster Matching)

Syntax:

```
RASTERMATCH (<raster1>, <raster2>)  
  
or  
  
RASTERMATCH (<raster1>, <ignore-params1>, <raster2>)  
  
or  
  
RASTERMATCH (<raster1>, <raster2>, <ignore-params2>)  
  
or  
  
RASTERMATCH (<raster1>, <ignore-params1>, <raster2>, <ignore-params2>)
```

Function Type:

Combination ([Global](#) and [Point](#))

Description:

Maps **<raster1>** through a lookup table so that the histogram of each layer of the returned **RASTER** will have approximately the same shape as the histogram of the corresponding layer of **<raster2>**.

<ignore-params1> and **<ignore-params2>** determine how histograms are computed for **<raster1>** and **<raster2>** respectively. Each of **<ignore-params1>** and **<ignore-params2>** consists of **<ignoreoption>** or **<ignoreoption> <backgroundvalue>**, as in the [HISTOGRAM](#) function.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value.

If **<ignoreoption>** is not present, the computation depends on whether the associated **RASTER** (**<raster1>** or **<raster2>**) is a previously existing file, or a **RASTER** created within the model. If it is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the **SET DEFAULT STATISTICS** statement.

If the **RASTER** (**<raster1>** or **<raster2>**) is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of the **RASTER** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option is used, as set by the [Preference Editor](#) or the **SET DEFAULT STATISTICS** statement.

Data Types:

<raster1> may be any numeric type. The type of **<raster2>** determines the output type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.

Object Types:

<raster1> and **<raster2>** are both **RASTER**. They must either have the same number of layers, or either may have one layer.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

The output is a **RASTER** which has the same number of layers as the maximum of **<raster1>** and **<raster2>**.

Notes:

If **<raster1>** and **<raster2>** both have one layer, this is equivalent to:

```
LOOKUP (<raster1>, HISTMATCH (HISTOGRAM (<raster1>), HISTOGRAM
(<raster2>)))
```

See Also:

[HISTMATCH](#)
[HISTOEQ](#)
[HISTOGRAM](#)
[LOOKUP](#)
[SET DEFAULT STATISTICS](#)



SIEVETABLE (Get Sieve Lookup Table)

Syntax: `SIEVETABLE (<threshold>, <hist>)`

Function Type: [Point](#)

Description: **SIEVETABLE** produces a lookup table which can be used to filter out small clumps from a layer which is the output of **CLUMP**. **<hist>** is a table which contains the histogram from the **CLUMP**ed layer. **<threshold>** is a **SCALAR** which specifies the minimum clump size to retain.

You can use the **CLUMP**ed layer and the output of **SIEVETABLE** as input into the **LOOKUP** function to create a new layer with the small clumps filtered out. The small clumps will be recoded to value zero.

Data Types: **<threshold>** may be any numeric type, and is converted to **FLOAT**. **<hist>** must be **FLOAT**. The output is **INTEGER**.

Input	Output	Comments
BINARY	not supported	
INTEGER	not supported	
FLOAT	INTEGER	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: **<raster>** is a single layer **RASTER**. **<threshold>** is a **SCALAR**. The output is a single layer **RASTER**.

Notes: See the example for **DELROWS**.

See Also: [CLUMP](#)
[LOOKUP](#)
[DELROWS](#)

STRETCH (Stretch)

Syntax:

```
STRETCH (<raster>, <stdcount>, <min>, <max>)
```

or

```
STRETCH (<raster>, <stdcount>, <min>, <max>, <ignoreoption>)
```

or

```
STRETCH (<raster>, <stdcount>, <min>, <max>, <ignoreoption>  
<backgroundvalue>)
```

Function Type:

Combination ([Global](#) and [Point](#))

Description:

Performs a linear scale and shift on the input **<raster>** such that each layer of the output **RASTER** meets the following conditions:

$$\text{mean (output)} - \text{<stdcount>} * \text{standard deviation (output)} = \text{<min>}$$

$$\text{mean (output)} + \text{<stdcount>} * \text{standard deviation (output)} = \text{<max>}$$

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values from **<raster>** in the computing the mean and standard deviation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value.

If **<ignoreoption>** is not present, the computation depends on whether **<raster>** is a previously existing file, or a **RASTER** created within the model. If **<raster>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<raster>** is a previously existing raster file, and **<ignoreoption>** is not present, the result is normally be computed from the statistics stored with the file. If all layers of **<raster>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file are used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.



Data Types:

The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	not supported	
STRING	not supported	

<stdcount>, **<min>**, and **<max>** may be any numeric type.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.

Object Types:

<raster> is a **RASTER**. **<stdcount>**, **<min>**, and **<max>** must be either **SCALAR** or a **TABLE** with as many rows as **<raster>** has layers.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

Notes:

Equivalent to:

$$\begin{aligned} & \text{<min>} + (\text{<raster>} - \text{GLOBAL MEAN (<raster>)}) + \text{GLOBAL SD (<raster>) } * \\ & \text{<stdcount>} * (\text{<max>} - \text{<min>}) / (\text{GLOBAL SD (<raster>) } * 2. * \\ & \text{<stdcount>}) \end{aligned}$$

See Also:

[GLOBAL MEAN](#)

[GLOBAL SD](#)

[SET DEFAULT STATISTICS](#)

Arithmetic

+ (Addition)

- (Subtraction)

- (Negation)

***** (Multiplication)

/ (Division)

MOD (*Modulus*)

! (Factorial)



For more information see [Standard Rules](#).



+ (Addition)

Syntax:

`<arg1> + <arg2>`

Function Type:

[Point](#)

Description:

Add **<arg1>** and **<arg2>**.

Data Types:

Input	Output	Comments
BINARY	BINARY	Equivalent to: <arg1> OR <arg2>
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

See Also:

[SUM](#)

[FOCAL SUM](#)

[GLOBAL SUM](#)

- (Subtraction)**Syntax:** `<arg1> - <arg2>`**Function Type:** [Point](#)**Description:** Subtract **<arg2>** from **<arg1>**.**Data Types:**

Input	Output	Comments
BINARY	BINARY	Equivalent to: <arg1> AND NOT <arg2>
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.**See Also:** [- \(Negation\)](#)
[+ \(Addition\)](#)

- (Negation)

Syntax:

- <arg1>

Function Type:

[Point](#)

Description:

Negative of <arg1>.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

See Also:

[- \(Subtraction\)](#)

[~ \(Bitwise Not\)](#)

[NOT](#)

*** (Multiplication)**

Syntax: `<arg1> * <arg2>`

Function Type: [Point](#)

Description: Multiply **<arg1>** by **<arg2>**.

Data Types:

Input	Output	Comments
BINARY	BINARY	Equivalent to: <arg1> AND <arg2>
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [MATMUL](#)



/ (Division)

Syntax:

<arg1> / <arg2>

Function Type:

[Point](#)

Description:

Divide <arg1> by <arg2>.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

Notes:

Integer division by 0 will cause error to be reported. Floating point division by zero results in floating point "Infinity" value.

See Also:

[MATDIV](#)

MOD (Modulus)

Syntax:

<arg1> MOD <arg2>

Function Type:

Point

Description:

Returns the remainder (modulus) when **<arg1>** is divided by the **<arg2>**. The result is the same sign as **<arg1>**.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

Example:

3 MOD 2 equals 1

-3 MOD 2 equals -1

3 MOD -2 equals 1

-3 MOD -2 equals -1

Notes:

Integer modulus by 0 will cause error to be reported. Floating point modulus by zero results in floating point "NaN" (not a number) value.

See Also:

[/ \(Division\)](#)



! (Factorial)

Syntax: `<arg1> !`

Function Type: [Point](#)

Description: Computes **<arg1>** factorial.
Factorial is commonly used in statistical analyses.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [GAMMA](#)



Bitwise

& (Bitwise And)

| (Bitwise Or)

^ (Bitwise Exclusive Or)

~ (Bitwise Not)



For more information see [Standard Rules](#).



& (Bitwise And)

Syntax:

<arg1> & <arg2>

Function Type

: [Point](#)

Description:

Compute bitwise and of <arg1> and <arg2>.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	INTEGER	FLOAT inputs converted to INTEGER
COMPLEX	not supported	
COLOR	undefined	
STRING	not supported	

Object Types:

All object types, standard rules.

See Also:

[| \(Bitwise Or\)](#)

[&& \(Logical And\)](#)

| (Bitwise Or)**Syntax:**

<arg1> | <arg2>

Function Type:

Point

Description:

Computes bitwise or of <arg1> and <arg2>.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	INTEGER	FLOAT inputs converted to INTEGER
COMPLEX	not supported	
COLOR	undefined	
STRING	not supported	

Object Types:

All object types, standard rules.

See Also:[^ \(Bitwise Exclusive Or\)](#)[& \(Bitwise And\)](#)[|| \(Logical Or\)](#)

^ (Bitwise Exclusive Or)

Syntax: `<arg1> ^ <arg2>`

Function Type: [Point](#)

Description: Computes bitwise exclusive or of **<arg1>** and **<arg2>**.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	INTEGER	FLOAT inputs converted to INTEGER
COMPLEX	not supported	
COLOR	undefined	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [| \(Bitwise Or\)](#)

~ (Bitwise Not)**Syntax:**

~ <arg1>

Function Type:

Point

Description:

Reverse bits of <arg1>.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	INTEGER	FLOAT inputs converted to INTEGER
COMPLEX	not supported	
COLOR	undefined	
STRING	not supported	

Object Types:

All object types, standard rules.

See Also:

NOT

- (Subtraction)





Boolean

[AND](#) (*Logical And*)

[&&](#) (*Logical And*)

[OR](#) (*Logical Or*)

[||](#) (*Logical Or*)

[NOT](#) (*Logical NOT*)



For more information see [Standard Rules](#).



AND (Logical And)

Syntaxes:

`<arg1> AND <arg2>`

or

`<arg1> && <arg2>`

Function Type:

[Point](#)

Description:

True if **<arg1>** and **<arg2>** are both non-zero, false otherwise.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	BINARY	
COLOR	undefined	
STRING	not supported	

Object Types:

All object types, standard rules.

Notes:

Identical to **&&**.

See Also:

[& \(Bitwise And\)](#)

[OR \(Logical Or\)](#)



OR (Logical Or)

Syntaxes:

```
<arg1> OR <arg2>
```

or

```
<arg1> || <arg2>
```

Function Type:

Point

Description:

True if either **<arg1>** or **<arg2>** is non-zero, false otherwise.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	BINARY	
COLOR	undefined	
STRING	not supported	

Object Types:

All object types, standard rules.

Notes:

Identical to ||.

See Also:

[| \(Bitwise Or\)](#)

[AND \(Logical And\)](#)



NOT (Logical NOT)

Syntax:

NOT <arg1>

Function Type:

[Point](#)

Description:

True if **<arg1>** is zero, false otherwise.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	BINARY	
COLOR	undefined	
STRING	not supported	

Object Types:

All object types, standard rules.

Notes:

Equivalent to **DELTA (<arg1>)**.

See Also:

[DELTA](#)

[~ \(Bitwise Not\)](#)

[ISALLTRUE](#)

[ISNONZERO](#)



Color

COLOR (*Create Color Scalar*)

HUE (*Get Hue from RGB*)

IHSTOBLU (*Get Blue from Intensity, Hue, and Saturation*)

IHSTOGRN (*Get Green from Intensity, Hue, and Saturation*)

IHSTORED (*Get Red from Intensity, Hue, and Saturation*)

IHSTORGB (*Get Red, Green, and Blue from Intensity, Hue, and Saturation*)

INTENS (*Get Intensity from RGB*)

RGBTOIHS (*Get Intensity, Hue, and Saturation from Red, Green, and Blue*)

SATUR (*Get Saturation from RGB*)

STACK (*Convert FLOAT TABLE to COLOR TABLE*)

UNSTACK (*Convert COLOR SCALAR to FLOAT TABLE*)



For more information see [Standard Rules](#).



COLOR (Create Color Scalar)

Syntax:

```
COLOR (<colorname>)  
  
or  
  
COLOR (<redval>, <greenval>, <blueval>)
```

Function Type:

[Point](#)

Description:

Converts either the color name string constant in **<colorname>**, or the red, green, and blue values input into a **COLOR SCALAR**.

Data Types:

<colorname> is a **STRING** constant.

<redval>, **<greenval>**, and **<blueval>** are numeric type, and are converted to **FLOAT**.

The output is **COLOR**.

Object Types:

<colorname> is a constant.

<redval>, **<greenval>**, and **<blueval>** are **SCALAR**.

Output is **SCALAR**.

Notes:

The color names and associated RGB values available to use for **<colorname>** are contained in the file **/usr/imagene/etc/color.txt**.

See Also:

[STACK](#)

[UNSTACK](#)

[:: \(Read Descriptor Column or Color Table\)](#)



HUE (Get Hue from RGB)

Syntax:

```
HUE (<redval>, <greenval>, <blueval>)
```

or

```
HUE (<redval>, <greenval>, <blueval>, <maxval>)
```

or

```
HUE (<redval>, <greenval>, <blueval>, <redmax>, <greenmax>,
    <bluemax>)
```

Function Type:

Point

Description:

Computes hue from red, green, and blue values. If **<maxval>** is present, the range of input data values for red, green, and blue is assumed to be zero to **<maxval>**. If **<redmax>**, **<greenmax>**, and **<bluemax>** are present, red input values are assumed to range from zero to **<redval>**, green input values from zero to **<greenval>**, and blue input values from zero to **<blueval>**.

If neither **<maxval>** nor **<redmax>**, **<greenmax>**, and **<bluemax>** are present, the data range for all three inputs is assumed to be zero to the default RGB maximum, which is 255. The output data values will range from 0.0 to 360.0. The values of hue are as follows:

Blue	0
Magenta	60
Red	120
Yellow	180
Green	240
Cyan	300

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types:

<redval>, **<greenval>**, and **<blueval>** may be any object type. The standard rules apply for the result type. **<maxval>**, **<redmax>**, **<greenmax>**, and **<bluemax>**, if present, must be **SCALAR**.



See Also:

[INTENS](#)

[SATUR](#)

[RGBTOIHS](#)

[IHSTORGB](#)

[IHSTORED](#)

[IHSTOGRN](#)

[IHSTOBLU](#)



IHSTOBLU (Get Blue from Intensity, Hue, and Saturation)

Syntax:

```
IHSTOBLU (<intensity>, <hue>, <saturation>)
```

or

```
IHSTOBLU (<intensity>, <hue>, <saturation>, <maxval>)
```

or

```
IHSTOBLU (<intensity>, <hue>, <saturation>, <intensmax>, <huemax>, <saturmax>)
```

Function Type: [Point](#)

Description: Computes blue from intensity, hue, and saturation values.

If **<maxval>** is present, the range of input data values for intensity, hue, and saturation is assumed to be zero to **<maxval>**. If **<intensmax>**, **<huemax>**, and **<saturmax>** are present, intensity input values are assumed to range from zero to **<intensmax>**, hue input values from zero to **<huemax>**, and saturation input values from zero to **<saturmax>**.

If neither **<maxval>** nor **<intensmax>**, **<huemax>**, and **<saturmax>** are present, the data range for the inputs is assumed to be zero to the default maximums for IHS, which are 1.0 for intensity, 360.0 for hue, and 1.0 for saturation. The output data values will range from 0.0 to 1.0.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: **<intensity>**, **<hue>**, and **<saturation>** may be any object type. The standard rules apply for the result type. **<maxval>**, **<intensmax>**, **<huemax>**, and **<saturmax>**, if present, must be **SCALAR**.

See Also:

[INTENS](#)[HUE](#)[SATUR](#)[RGBTOIHS](#)

[IHSTORGB](#)[IHSTORED](#)[IHSTOGRN](#)

IHSTOGRN (Get Green from Intensity, Hue, and Saturation)

Syntax:

```
IHSTOGRN (<intensity>, <hue>, <saturation>)  
  
or  
  
IHSTOGRN (<intensity>, <hue>, <saturation>, <maxval>)  
  
or  
  
IHSTOGRN (<intensity>, <hue>, <saturation>, <intensmax>, <huemax>, <saturmax>)
```

Function Type:

[Point](#)

Description:

Computes green from intensity, hue, and saturation values.

If **<maxval>** is present, the range of input data values for intensity, hue, and saturation is assumed to be zero to **<maxval>**. If **<intensmax>**, **<huemax>**, and **<saturmax>** are present, intensity input values are assumed to range from zero to **<intensmax>**, hue input values from zero to **<huemax>**, and saturation input values from zero to **<saturmax>**.

If neither **<maxval>** nor **<intensmax>**, **<huemax>**, and **<saturmax>** are present, the data range for the inputs is assumed to be zero to the default maximums for IHS, which are 1.0 for intensity, 360.0 for hue, and 1.0 for saturation. The output data values will range from 0.0 to 1.0.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types:

<intensity>, **<hue>**, and **<saturation>** may be any object type. The standard rules apply for the result type. **<maxval>**, **<intensmax>**, **<huemax>**, and **<saturmax>**, if present, must be **SCALAR**.

See Also:

[INTENS](#) [HUE](#) [SATUR](#) [RGBTOIHS](#)
[IHSTORGB](#) [IHSTORED](#) [IHSTOBLU](#)



IHSTORED (Get Red from Intensity, Hue and Saturation)

Syntax:

```
IHSTORED (<intensity>, <hue>, <saturation>)
```

or

```
IHSTORED (<intensity>, <hue>, <saturation>, <maxval>)
```

or

```
IHSTORED (<intensity>, <hue>, <saturation>, <intensmax>, <huemax>, <saturmax>)
```

Function Type: [Point](#)

Description: Computes red from intensity, hue, and saturation values.

If **<maxval>** is present, the range of input data values for intensity, hue, and saturation is assumed to be zero to **<maxval>**. If **<intensmax>**, **<huemax>**, and **<saturmax>** are present, intensity input values are assumed to range from zero to **<intensmax>**, hue input values from zero to **<huemax>**, and saturation input values from zero to **<saturmax>**.

If neither **<maxval>** nor **<intensmax>**, **<huemax>**, and **<saturmax>** are present, the data range for the inputs is assumed to be zero to the default maximums for IHS, which are 1.0 for intensity, 360.0 for hue, and 1.0 for saturation. The output data values will range from 0.0 to 1.0.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: **<intensity>**, **<hue>**, and **<saturation>** may be any object type. The standard rules apply for the result type. **<maxval>**, **<intensmax>**, **<huemax>**, and **<saturmax>**, if present, must be **SCALAR**.

See Also:

[INTENS](#)[HUE](#)[SATUR](#)[RGBTOIHS](#)

[IHSTORGB](#)[IHSTOGRN](#)[IHSTOBLU](#)

IHSTORGB (Get Red, Green and Blue from Intensity, Hue and Saturation)

Syntax:

```
IHSTORGB (<ihs>)
or
IHSTORGB (<ihs>, <maxval>)
or
IHSTORGB (<ihs>, <intensmax>, <huemax>, <saturmax>)
```

Function Type:

Point

Description:

Computes red, green, and blue from intensity, hue, and saturation values contained in **<ihs>**.

<ihs> may be a three layer **RASTER**, in which case the first layer is used for intensity input values, the second for hue, and the third for saturation. In this case a three layer **RASTER** is output, with the layers representing red, green, and blue.

<ihs> may also be a **COLOR SCALAR** or **COLOR TABLE**, in which case the three values for each element of **<ihs>** are considered to be intensity, hue, and saturation, rather than red, green, and blue. The output in this case is a **COLOR** object the same size and type as the input.

If **<maxval>** is present, the range of input data values for intensity, hue, and saturation is assumed to be zero to **<maxval>**. If **<intensmax>**, **<huemax>**, and **<saturmax>** are present, intensity input values are assumed to range from zero to **<intensmax>**, hue input values from zero to **<huemax>**, and saturation input values from zero to **<saturmax>**.

If neither **<maxval>** nor **<intensmax>**, **<huemax>**, and **<saturmax>** are present, the data range for the inputs is assumed to be zero to the default maximums for IHS, which are 1.0 for intensity, 360.0 for hue, and 1.0 for saturation. The output data values will range from 0.0 to 1.0.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	Three layer RASTER only
FLOAT	FLOAT	Three layer RASTER only
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

Object Types:

<ihs> may be either a three layer **RASTER** or any object of data type **COLOR**. The output will be the same object type and size as **<ihs>**. **<maxval>**, **<intensmax>**, **<huemax>**, and **<saturmax>**, if present, must be **SCALAR**.

See Also:

INTENS HUE SATUR RGBTOIHS
IHSTORED IHSTOGRN IHSTOBLU



INTENS (Get Intensity from RGB)

Syntax:

```
INTENS (<redval>, <greenval>, <blueval>)
```

or

```
INTENS (<redval>, <greenval>, <blueval>, <maxval>)
```

or

```
INTENS (<redval>, <greenval>, <blueval>, <redmax>, <greenmax>, <bluemax>)
```

Function Type: [Point](#)

Description: Computes intensity from red, green, and blue values.

If **<maxval>** is present, the range of input data values for red, green, and blue is assumed to be zero to **<maxval>**. If **<redmax>**, **<greenmax>**, and **<bluemax>** are present, red input values are assumed to range from zero to **<redval>**, green input values from zero to **<greenval>**, and blue input values from zero to **<blueval>**.

If neither **<maxval>** nor **<redmax>**, **<greenmax>**, and **<bluemax>** are present, the data range for all three inputs is assumed to be zero to the default RGB maximum, which is 255. The output data values will range from 0.0 to 1.0.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: **<redval>**, **<greenval>**, and **<blueval>** may be any object type. The standard rules apply for the result type. **<maxval>**, **<redmax>**, **<greenmax>**, and **<bluemax>**, if present, must be **SCALAR**.

See Also:

[HUE](#)[SATUR](#)[RGBTOIHS](#)[IHSTORGB](#)

[IHSTORED](#)[IHSTOGRN](#)[IHSTOBLU](#)

RGBTOIHS (Get Intensity, Hue and Saturation from Red, Green and Blue)

Syntax:

```
RGBTOIHS (<rgb>)
or
RGBTOIHS (<rgb>, <maxval>)
or
RGBTOIHS (<rgb>, <redmax>, <greenmax>, <bluemax>)
```

Function Type:

Point

Description:

Computes intensity, hue, and saturation from red, green, and blue values contained in **<rgb>**.

<rgb> may be a three layer **RASTER**, in which case the first layer is used for red input values, the second for green, and the third for blue. In this case a three layer **RASTER** is output, with the layers representing intensity, hue, and saturation.

<rgb> may also be a **COLOR SCALAR** or **COLOR TABLE**. The output in this case is a **COLOR** object the same size and type as the input, where the three values for each element of the output are considered to be intensity, hue, and saturation, rather than red, green, and blue.

If **<maxval>** is present, the range of input data values for red, green, and blue is assumed to be zero to **<maxval>**. If **<redmax>**, **<greenmax>**, and **<bluemax>** are present, red input values are assumed to range from zero to **<redval>**, green input values from zero to **<greenval>**, and blue input values from zero to **<blueval>**.

If neither **<maxval>** nor **<redmax>**, **<greenmax>**, and **<bluemax>** are present, the data range for all three inputs is assumed to be zero to the default RGB maximum, which is 255. The output data values will range from 0.0 to 1.0 for intensity, 0.0 to 360.0 for hue, and 0.0 to 1.0 for saturation.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	Three layer RASTER only
FLOAT	FLOAT	Three layer RASTER only
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

Object Types:

<rgb> may be either a three layer **RASTER** or any object of data type **COLOR**. The output will be the same object type and size as **<rgb>**. **<maxval>**, **<redmax>**, **<greenmax>**, and **<bluemax>**, if present, must be **SCALAR**.

See Also:

INTENS HUE SATUR IHSTORGB
IHSTORED IHSTOGRN IHSTOBLU



SATUR (Get Saturation from RGB)

Syntax:

```
SATUR (<redval>, <greenval>, <blueval>)
```

or

```
SATUR (<redval>, <greenval>, <blueval>, <maxval>)
```

or

```
SATUR (<redval>, <greenval>, <blueval>, <redmax>, <greenmax>, <bluemax>)
```

Function Type: [Point](#)

Description: Computes saturation from red, green, and blue values.

If **<maxval>** is present, the range of input data values for red, green, and blue is assumed to be zero to **<maxval>**. If **<redmax>**, **<greenmax>**, and **<bluemax>** are present, red input values are assumed to range from zero to **<redval>**, green input values from zero to **<greenval>**, and blue input values from zero to **<blueval>**.

If neither **<maxval>** nor **<redmax>**, **<greenmax>**, and **<bluemax>** are present, the data range for all three inputs is assumed to be zero to the default RGB maximum, which is 255. The output data values will range from 0.0 to 1.0.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: **<redval>**, **<greenval>**, and **<blueval>** may be any object type. The standard rules apply for the result type. **<maxval>**, **<redmax>**, **<greenmax>**, and **<bluemax>**, if present, must be **SCALAR**.

See Also:

[INTENS](#)[HUE](#)[RGBTOIHS](#)[IHSTORGB](#)

[IHSTORED](#)[IHSTOGRN](#)[IHSTOBLU](#)

STACK (Convert FLOAT TABLE to COLOR SCALAR)

Syntax: `STACK (<table1>)`

Function Type: [Point](#)

Description: **STACK** converts the RGB values from a float table to a color scalar.

If input is **FLOAT TABLE** with three rows, output will be **COLOR SCALAR**.

If input is a **TABLE** of data type **BINARY**, **INTEGER**, **FLOAT**, **COMPLEX**, or **STRING** with one row, output is **SCALAR** of same type.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	COLOR	
COMPLEX	COMPLEX	
COLOR	not supported	
STRING	STRING	

Object Types: Input must be a **TABLE**; output is **SCALAR**.

Notes: Inverse of **UNSTACK**.

An input **TABLE** of any size not listed in the Description above will have undefined result.

COLOR data type input or input of any object type other than **TABLE** will result in an error.

See Also: [UNSTACK](#)

UNSTACK (Convert COLOR SCALAR to FLOAT TABLE)

Syntax: `UNSTACK (<scalar1>)`

Function Type: [Point](#)

Description: If input is **COLOR SCALAR**, output will be **FLOAT TABLE** with three rows. Input of any other data type will be converted to **TABLE** with one row.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	FLOAT	
STRING	STRING	

Object Types: Input must be **SCALAR**, output is **TABLE**.

Notes: Inverse of **STACK**.

See Also: [STACK](#)





Conditional

[CONDITIONAL](#) (*Conditional*)

[EITHER...IF...OR....OTHERWISE](#) (*Select on Binary Test*)

[INDEX](#) (*Index - Find Matching Item on List*)

[PICK](#) (*Pick - Get nth Item on List*)



For more information see [Standard Rules](#).



CONDITIONAL (Conditional)

Syntax:

```
CONDITIONAL {(<test1>) <arg1>, (<test2>) <arg2>, (<test3>) <arg3>,  
...}
```

Function Type:

[Point](#)

Description:

<test1> is converted to **BINARY**. If true, **<arg1>** is returned. Otherwise, **<test2>** is converted to **BINARY**. If true, **<arg2>** is returned, etc. The first expression that is true will determine the output value. If none of the test objects is true, zero is returned, or "" (the empty string) if the type of the **<args>** is **STRING**.

For rasters, a conditional statement is often used to determine the output class value (recode) by testing the input class value.

Data Types:

<test1>, **<test2>**, etc., are numeric, and are converted to **BINARY**.

<arg1>, **<arg2>**, etc., may be any type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	STRING	

Object Types:

All object types, standard rules.

Notes:

CONDITIONAL operates element by element on tables, matrices, and rasters, like most other point functions.

See Also:

[EITHER...IF...OR...OTHERWISE](#)

[PICK](#)



EITHER...IF...OR....OTHERWISE (Select on Binary Test)

Syntax: `EITHER <arg1> IF (<test>) OR <arg2> OTHERWISE`

Function Type: [Point](#)

Description: **<test>** is converted to **BINARY**. If true, **<arg1>** is returned. Otherwise, **<arg2>** is returned.

Data Types: **<test>** is numeric, and is converted to **BINARY**.
<arg1> and **<arg2>** may be any type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	STRING	

Object Types: All object types, standard rules.

Notes: **EITHER...IF...OR....OTHERWISE** operates element by element on tables, matrices, and rasters, like most other point functions.

See Also: [CONDITIONAL](#)
[PICK](#)
[Flow Control](#)



INDEX (Index - Find Matching Item on List)

Syntax: `INDEX (<test>) {<arg1>, <arg2>, <arg3>, ...}`

Function Type: [Point](#)

Description: If **<test>** equals **<arg1>**, 1 is returned. If **<test>** equals **<arg2>**, 2 is returned, etc. If **<test>** is not equal to any of the arguments on the right, 0 is returned.

Data Types:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	INTEGER	
COMPLEX	INTEGER	
COLOR	INTEGER	
STRING	INTEGER	

Object Types: All object types, standard rules.

Notes: **INDEX** operates element by element on tables, matrices, and rasters, like most other point functions.

See Also: [PICK](#)

PICK (Pick - Get nth Item on List)

Syntax: `PICK (<number>) {<arg1>, <arg2>, <arg3>, ...}`

Function Type: [Point](#)

Description: If **<number>** is 1, **<arg1>** is returned. If **<number>** is 2, **<arg2>** is returned, etc. If **<number>** is less than one or greater than the number of arguments on the right, zero is returned, or if the arguments are **STRING**, "" (the empty string) is returned.

Data Types: **<number>** must be **BINARY**, **INTEGER**, or **FLOAT**, and is converted to **INTEGER**.
<arg1>, **<arg2>**, etc., may be any data type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	STRING	

Object Types: All object types, standard rules.

Notes: **PICK** operates element by element on tables, matrices, and rasters, like most other point functions.

See Also: [CONDITIONAL](#)
[EITHER...IF...OR...OTHERWISE](#)
[INDEX](#)





Data Generation

MAPX (*Create Raster Containing X Map Coordinates*)

MAPY (*Create Raster Containing Y Map Coordinates*)

MATRIX (*Create Matrix from List of Scalars*)

MATRIX (*Read Matrix from Kernel Library*)

MATRIX SERIES (*Create Matrix Containing 2-D Series*)

PIXELX (*Create Raster Containing Column Number*)

PIXELY (*Create Raster Containing Row Number*)

RANDOM (*Generate Random Value*)

STACKLAYERS (*Stack Raster Layers*)

TABLE (*Create Table from List of Scalars*)

TABLE SERIES (*Create Table Containing Series*)



For more information see [Standard Rules](#).



MAPX (Create Raster Containing X Map Coordinates)

Syntax: MAPX

Function Type: [Point](#)

Description: Returns a raster in which each pixel contains the X map coordinate corresponding to its position.

Data Types: Output is **FLOAT**.

Object Types: Output is single layer **RASTER**.

Notes: If Working Window is in pixel coordinates, returns pixel coordinates converted to **FLOAT**.

See Also: [MAPY](#)
[PIXELX](#)

MAPY (Create Raster Containing Y Map Coordinates)

Syntax: `MAPY`

Function Type: [Point](#)

Description: Returns a raster in which each pixel contains the Y map coordinate corresponding to its position.

Data Types: Output is **FLOAT**.

Object Types: Output is single layer **RASTER**.

Notes: If Working Window is in pixel coordinates, returns pixel coordinates converted to **FLOAT**.

See Also: [MAPX](#)
[PIXELY](#)



MATRIX (Create Matrix from List of Scalars)

Syntax: `MATRIX (<rows>, <columns>: <arg1>, <arg2>, <arg3>, ...)`

Function Type: [Point](#)

Description: Returns a matrix **<rows>** rows by **<columns>** columns containing the scalar arguments in the order listed across successive rows. For example:

```
MATRIX (2, 3: 1, 2, 3, 4, 5, 6)
```

results in the matrix:

```
1 2 3
4 5 6
```

Data Types: **<rows>** and **<columns>** are **INTEGER** constants. **<arg1>**, **<arg2>**, etc., may be any type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	STRING	

Object Types: **<rows>** and **<columns>** are constants. All input objects are **SCALAR**. Output is **MATRIX** with having **<rows>** rows and **<columns>** columns. The number of arguments **<arg1>**, **<arg2>**, etc., must equal **<rows> * <columns>**.

See Also: [MATRIX](#) (Read Matrix from Kernel Library)

[MATRIX SERIES](#)

[TABLE](#)



MATRIX (Read Matrix from Kernel Library)

Syntax:

```
MATRIX (<kernelname>)  
or  
MATRIX (<libraryname>, <kernelname>)
```

Function Type:[Point](#)**Description:**

Returns a matrix read from a kernel library. If **<libraryname>** is present, the kernel is read from this library. Otherwise the default kernel library **<\$IMAGINE_HOME>/etc/default.klb** is used.

Data Types:

<libraryname> and **<kernelname>** are **STRING** constants.

The output is type FLOAT.

Object Types:

<libraryname> and **<kernelname>** are **SCALAR** constants. Output is **MATRIX**. The size of the output matrix is determined by the input file.

See Also:

[MATRIX](#) (Create Matrix from List of Scalars)
[CONVOLUTION](#)



MATRIX SERIES (Create Matrix Containing 2-D Series)

Syntax:

```
MATRIX SERIES (<rows>, <columns>, <initval>, <rowincrement>,  
<columnincrement>)
```

Function Type:

[Point](#)

Description:

Returns a matrix having **<rows>** rows and **<columns>** columns. The first element of the returned matrix contains **<initval>**. Each successive column in the matrix is incremented by **<columnincrement>**, and each successive row by **<rowincrement>**. For example:

```
MATRIX SERIES (2, 3, 10, 2, .3)
```

results in the matrix:

```
10 10.3 10.6  
12 12.3 12.6
```

Data Types:

<rows> and **<columns>** are **BINARY**, **INTEGER**, or **FLOAT**, and are converted to **INTEGER**. **<initval>**, **<rowincrement>**, and **<columnincrement>** may be any type other than **STRING**:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types:

All input objects are **SCALAR**. Output is **MATRIX** having **<rows>** rows and **<columns>** columns.

See Also:

[MATRIX](#) (Create Matrix from List of Scalars)

[MATRIX](#) (Read Matrix from Kernel Library)

[TABLE SERIES](#)

PIXELX (Create Raster Containing Column Number)

Syntax: `PIXELX`

Function Type: [Point](#)

Description: Returns a raster in which each pixel contains its column position in the Working Window. Column positions start at zero.

Data Types: Output is **INTEGER**

Object Types: Output is single layer **RASTER**.

See Also: [PIXELY](#)
[MAPX](#)



PIXELY (Create Raster Containing Row Number)

Syntax: `PIXELY`

Function Type: [Point](#)

Description: Returns a raster in which each pixel contains its row position in the Working Window. Row positions start at zero.

Data Types: Output is **INTEGER**

Object Types: Output is single layer **RASTER**.

See Also: [PIXELX](#)
[MAPY](#)

RANDOM (Generate Random Values)

Syntax: `RANDOM(<arg1>)`

Function Type: [Point](#)

Description: Returns object filled with pseudo-randomly generated floating point values greater than or equal to 0 and less than 1. The data type and value of the input is ignored. The input object determines the object type of the result.

Data Types:

Input	Output	Comments
BINARY	FLOAT	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	COLOR	
STRING	FLOAT	

Object Types: Any object type may be input. Output object type is the same as input.

See Also: [SET RANDOM SEED](#)



STACKLAYERS (Stack Raster Layers)

Syntax:

```
STACKLAYERS (<arg1>, <arg2>, <arg3>, ...)
```

Function Type:

[Point](#)

Description:

Outputs **RASTER** which includes all the layers from **<arg1>**, **<arg2>**, **<arg3>**, etc. For example, if **<arg1>** has three layers, and **<arg2>** has four layers, layers 1 through 3 of the output would be copied from **<arg1>**, layers 4 through 7 would be copied from **<arg2>**, and so forth.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	not supported	
STRING	not supported	

Object Types:

All inputs must be either **RASTER** or **SCALAR**. Each **RASTER** input may have any number of layers. The output is a **RASTER**. The number of layers in the output is the sum of the number of layers from all inputs.

See Also:

[Raster Layer Stacks](#)

TABLE (Create Table from List of Scalars)**Syntax:**

```
TABLE (<count>: <arg1>, <arg2>, <arg3>, ...)
```

or

```
TABLE (<arg1>, <arg2>, <arg3>, ...)
```

Function Type:

[Point](#)

Description:

Creates a table containing the scalar arguments input in the order listed. If **<count>** is present, the number of subsequent arguments must equal **<count>**.

Data Types:

<count> is an **INTEGER** constant. **<arg1>**, **<arg2>**, etc., may be any type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	STRING	

Object Types:

<count> is a constant. All input objects are **SCALAR**. Output is **TABLE** having number of rows equal to number of arguments **<arg1>**, **<arg2>**, etc. If **<count>** is present, number of rows equals **<count>**.

See Also:

[LOOKUP](#)

[TABLE SERIES](#)

[MATRIX](#)



TABLE SERIES (Create Table Containing Series)

Syntax: `TABLE SERIES (<count>, <initval>, <increment>)`

Function Type: [Point](#)

Description: Creates a table containing **<count>** elements. The first element of the returned table contains **<initval>**. Each successive element in the table contains its predecessor + **<increment>**. For example:

```
TABLE SERIES (4, 10, 2)
```

results in the table:

```
10
12
14
16
```

Data Types: **<count>** is **BINARY**, **INTEGER**, or **FLOAT**, and is converted to **INTEGER**. **<initval>** and **<increment>** may be any type other than **STRING**:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All input objects are **SCALAR**. Output is **TABLE** having **<count>** rows.

See Also: [TABLE](#)
[MATRIX SERIES](#)

Descriptor

. (Map Raster Through Descriptor Column)

:: (Read Descriptor Column or Color Table)



For more information see [Standard Rules](#).



. (Map Raster Through Descriptor Column)

Syntax: `<raster> . <name>`

Function Type: [Point](#)

Description: Maps the single layer **RASTER** `<raster>` through the descriptor column `<name>` from the descriptor table for the file layer associated with `<raster>`.

The pixel values of the input `<raster>` are converted to bin numbers using the descriptor table's bin function. These bin numbers are then used as an index into the table read from the named descriptor column. The result is a single layer raster whose data type is the same as the named descriptor column.

Data Types: `<raster>` may be any data type. `<name>` is a **STRING** constant. Output is the same type as the descriptor column named in `<name>`.

Object Types: `<raster>` is a single layer **RASTER**. `<name>` is a constant. Output is single layer **RASTER**.

Notes: `<raster>` must be a raster variable associated with an existing single layer file or an explicit layer from an existing file:

`<raster-variable> (<integer-constant>)` is correct, but

`<raster-variable> (<integer-variable>)` is not allowed.

This function is equivalent to:

```
LOOKUP (<raster>, <raster> :: <name>)
```

If a **TABLE** variable has been declared to be associated with the descriptor column referenced by this operation, and that variable has been modified, this operation will not use the modified value stored in the variable. It will read the original unmodified values from the file layer's descriptor table.

See Also: [:: \(Read Descriptor Column or Color Table\)](#)
[LOOKUP](#)

:: (Read Descriptor Column or Color Table)**Syntax:**

```
<raster> :: <name>
or
<raster> :: COLORTABLE
```

Function Type:

Point

Description:

Reads and returns a descriptor column or the color table from the descriptor table for the file layer associated with **<raster>**.

If **<name>** is present, reads the descriptor column named **<name>** and returns a **TABLE** whose type is the same as the descriptor column.

If **COLORTABLE** is present, reads the "Red", "Green," and "Blue" columns of the descriptor table and returns a **COLOR TABLE**.

<raster> must be a single layer **RASTER** associated with a file layer having a descriptor table.

Data Types:

<raster> may be any data type. **<name>** is a **STRING** constant. Output is the same type as the named descriptor column if **<name>** is used. Output is **COLOR** if **COLORTABLE** present.

Object Types:

<raster> is a single layer **RASTER**. **<name>** is a constant. Output is **TABLE**. The number of rows is the number of bins in the descriptor table.

Notes:

<raster> must be a raster variable associated with an existing single layer file, or an explicit layer from an existing file:

<raster-variable> (<integer-constant>) is correct, but

<raster-variable> (<integer-variable>) is not allowed.

If a **TABLE** variable has been declared to be associated with the descriptor column referenced by this operation, and that variable has been modified, this operation will not return the modified value stored in the variable. It will read the original unmodified values from the file layer's descriptor table.

See Also:

[. \(Map Raster through Descriptor Column\)](#)





Distance

[CIRC](#) (*Test if Inside Unit Circle*)

[DIST](#) (*Distance*)

[RECT](#) (*Rectangle*)

[SEARCH](#) (*Search - Proximity Analysis*)

[TRI](#) (*Triangle*)



For more information see [Standard Rules](#).



CIRC (Test if Inside Unit Circle)

Syntax: `CIRC (<arg1>, <arg2>)`

Function Type: [Point](#)

Description: Returns true if inside unit circle, false otherwise. For example:
True if $\langle \text{arg1} \rangle^2 + \langle \text{arg2} \rangle^2 \leq 1$.
False if $\langle \text{arg1} \rangle^2 + \langle \text{arg2} \rangle^2 > 1$.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	not supported	
COLOR	undefined	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [RECT](#)
[TRI](#)
[DIST](#)

DIST (Distance)

Syntax: `DIST (<arg1>, <arg2>)`

Function Type: [Point](#)

Description: Computes distance from origin:

SQRT (<arg1> ** 2 + <arg2> ** 2)

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [SQRT](#)



RECT (Rectangle)

Syntax: `RECT (<arg1>)`

Function Type: [Point](#)

Description: For **INTEGER**, **FLOAT**, **COLOR**—
Returns: **ABS (<arg1>) <= 0.5**, i.e.:

FALSE	if	<arg1> < -0.5
TRUE	if	-0.5 <= <arg1> <= 0.5
FALSE	if	0.5 > <arg1>

For **COMPLEX**—
Returns **ABS (REAL (<arg1>)) <= 0.5 AND ABS (IMAG (<arg1>)) <= 0.5**

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	BINARY	
COLOR	undefined	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [TRI](#)
[CIRC](#)

SEARCH (Search - Proximity Analysis)

Syntax:

```
SEARCH (<raster>, <dist>, <table>)
```

or

```
SEARCH (<raster>, <dist>, <class1>, <class2>, <class3>, ...)
```

Function Type:

Layer

Description:

Performs a proximity analysis on **<raster>**, a single layer **RASTER**. The distance in pixels to search is specified by **<dist>**, a numeric **SCALAR**. The classes in **<raster>** from which to search are either listed as arguments **<class1>**, **<class2>**, etc., or are contained in **<table>**. The output is a single layer **RASTER**, where the data value at each pixel is the distance in pixels from the nearest pixel whose value in **<raster>** belongs to the set of search classes.

Data Types:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	INTEGER	FLOAT inputs converted to INTEGER
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types:

<raster> is a single layer **RASTER**. **<dist>** is a **SCALAR**. **<table>** is a **TABLE**, **<class1>**, **<class2>**, etc., are **SCALAR**. The output is a single layer **RASTER**.



TRI (Triangle)

Syntax: TRI (<arg1>)

Function Type: Point

Description: For **INTEGER**, **FLOAT**, **COLOR**—
Computes **MAX (1. - ABS (<arg1>), 0.)**, i.e.:

0	if	<arg1> < -1
1 + <arg1>	if	-1 <= <arg1> <= 0
1 - <arg1>	if	0 <= <arg1> <= 1
0	if	1 < <arg1>

For **COMPLEX**—
Computes **MAX (1. - ABS (REAL (<arg1>)) - ABS (IMAG (<arg1>)), 0.)**

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

See Also:

MAX REAL ABS IMAG
CIRC RECT

Exponential

[EXP](#) (*Exponential*)

[LOG](#) (*Natural Logarithm*)

[LOG10](#) (*Common Logarithm*)

[POWER](#) (*Raise to Power*)

[**](#) (*Raise to Power*)

[SQRT](#) (*Square Root*)



For more information see [Standard Rules](#).



EXP (Exponential)

Syntax:

EXP (<arg1>)

Function Type:

Point

Description:

Computes **e** raised to the **<arg1>** power. The constant **e** equals 2.71828182845904, the base of the natural logarithm.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

Example:

EXP (1) equals 2.718281828 (the approximate value of **e**)

EXP (2) equals **e**² (7.389056099)

Notes:

EXP is the inverse of **LOG**, the natural logarithm of **<arg1>**.

See Also:

[LOG](#)

[POWER](#)



LOG (Natural Logarithm)

Syntax: `LOG (<arg1>)`

Function Type: [Point](#)

Description: Computes the natural logarithm of **<arg1>**.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

Example: LOG (e) equals 1

Notes: If **<arg1>** is **INTEGER** or **FLOAT**, and **<arg1>** is less than or equal to zero, NaN (not a number) is returned.

See Also: [LOG10](#)
[EXP](#)



LOG10 (Common Logarithm)

Syntax: LOG10 (<arg1>)

Function Type: [Point](#)

Description: Computes the common logarithm (base 10) of <arg1>.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

Example: LOG10 (10) equals 1
LOG10 (86) equals 1.9344985

Notes: If <arg1> is **INTEGER** or **FLOAT**, and <arg1> is less than or equal to zero, NaN (not a number) is returned.

See Also: [LOG](#)

POWER (Raise to Power)**Syntaxes:**

```
<arg1> POWER <arg2>  
<arg1> ** <arg2>
```

Function Type:

Point

Description:Raise **<arg1>** to **<arg2>** power.**Data Types:****<arg2>** cannot be **COMPLEX**. The type of **<arg1>** determines the output data type:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.



SQRT (Square Root)

Syntax: `SQRT (<arg1>)`

Function Type: [Point](#)

Description: Computes the square root of **<arg1>**.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

Example:
SQRT (16) equals 4
SQRT (-16) equals NaN (not a number)
SQRT ((-16, 0)) equals (0, 4)

Notes: If **<arg1>** is a negative **INTEGER** or **FLOAT**, NaN (not a number) will be returned.

See Also: [** \(Raise to Power\)](#)



Focal (Scan)

Focal functions are generally used for neighborhood analyses.

BOUNDARY (*Boundary*)

FOCAL DENSITY (*Focal Density*)

FOCAL DIVERSITY (*Focal Diversity*)

FOCAL MAJORITY (*Focal Majority*)

FOCAL MAX (*Focal Maximum*)

FOCAL MEAN (*Focal Mean*)

FOCAL MEDIAN (*Focal Median*)

FOCAL MIN (*Focal Minimum*)

FOCAL MINORITY (*Focal Minority*)

FOCAL RANK (*Focal Rank*)

FOCAL SD (*Focal Standard Deviation*)

FOCAL STANDARD DEVIATION (*Focal Standard Deviation*)

FOCAL SUM (*Focal Sum*)



For more information see [Standard Rules](#).



BOUNDARY (Boundary)

Syntax:

```
BOUNDARY (<raster>, <focus>)
```

or

```
BOUNDARY (<raster>, <focus>, <option1> <valuelist1>)
```

or

```
BOUNDARY (<raster>, <focus>, <option1> <valuelist1>, <option2>  
<valuelist2>)
```

Function Type:

Neighborhood

Description:

Returns 0 (**FALSE**) if all pixels in the focal window **<focus>** have the same value. Returns 1 (**TRUE**) if there is more than one value in the focal window.

<option1> and **<option2>**, if present, may be either a **<use_option>** or an **<apply_option>**. A **<use_option>** is one of the following:

```
IGNORE_VALUE  
USE_VALUE  
USE_LOOKUP_TABLE
```

An **<apply_option>** is one of the following:

```
NO_APPLY_AT_VALUE  
APPLY_AT_VALUE  
APPLY_LOOKUP_TABLE
```

If **<option1>** and **<option2>** are both present, one of them must be a **<use_option>** and the other must be an **<apply_option>**.

<valuelist1> and **<valuelist2>**, if present, may be either a scalar or table expression, or they may be:

```
{ <value1>, <value2>, ..., <valueN> }
```

where **<value1>**, **<value2>**, etc., are each scalar expressions.

A **<use_option>** and its **<valuelist>** specifies which values are to be used in computing the focal function. Pixels in the neighborhood whose values are specified as being ignored will not be counted in the calculation of the focal function. **IGNORE_VALUE** specifies that the values in **<valuelist>** will not be used. **USE_VALUE** specifies that only the values in **<valuelist>** are used, all other values are ignored. **USE_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not a value is used: **TRUE** indicates to use the value, **FALSE** to ignore it. If all values in the neighborhood of a pixel are ignored, the output value is 0.

An **<apply_option>** and its **<valuelist>** specifies whether or not the focal function is calculated for a particular input pixel value. If the option specifies that the function is not to be applied for the value of a particular input pixel, the focal function is not calculated at that pixel, and the function returns the value of the input pixel. **NO_APPLY_AT_VALUE** specifies that the function will not be applied at values in **<valuelist>**. **APPLY_AT_VALUE** specifies that the function is applied only at the values in **<valuelist>**, and at no other values. **APPLY_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not to apply the function at a value: **TRUE** indicates to apply at the value, **FALSE** indicates not to apply at the value.

Data Types:

<focus> may be any numeric type; it is converted to **BINARY**. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	INTEGER	input rounded to INTEGER
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	



<use_option> and **<apply_option>** may be used only with **INTEGER** input raster data.

Object Types:

<raster> is a **RASTER**; **<focus>** is a **MATRIX**. **<valuelist1>** and **<valuelist2>** are either **SCALAR**, **TABLE**, or a comma-separated list of **SCALAR**s enclosed in braces. Output is a **RASTER** with same number of layers as **<raster>**.

See Also:

[FOCAL DIVERSITY](#)



FOCAL DENSITY (Focal Density)

Syntax:

```
FOCAL DENSITY (<raster>, <focus>)  
  
or  
  
FOCAL DENSITY (<raster>, <focus>, <option1> <valuelist1>)  
  
or  
  
FOCAL DENSITY (<raster>, <focus>, <option1> <valuelist1>,  
<option2> <valuelist2>)
```

Function Type:

Neighborhood

Description:

Returns number of occurrences of the center pixel value in focal window **<focus>** around each pixel of **<raster>**.

<option1> and **<option2>**, if present, may be either a **<use_option>** or an **<apply_option>**. A **<use_option>** is one of the following:

```
IGNORE_VALUE  
USE_VALUE  
USE_LOOKUP_TABLE
```

An **<apply_option>** is one of the following:

```
NO_APPLY_AT_VALUE  
APPLY_AT_VALUE  
APPLY_LOOKUP_TABLE
```

If **<option1>** and **<option2>** are both present, one of them must be a **<use_option>** and the other must be an **<apply_option>**.

<valuelist1> and **<valuelist2>**, if present, may be either a scalar or table expression, or they may be:

```
{ <value1>, <value2>, ..., <valueN> }
```

where **<value1>**, **<value2>**, etc., are each scalar expressions.

A **<use_option>** and its **<valuelist>** specifies which values are to be used in computing the focal function. Pixels in the neighborhood whose values are specified as being ignored will not be counted in the calculation of the focal function. **IGNORE_VALUE** specifies that the values in **<valuelist>** will not be used. **USE_VALUE** specifies that only the values in **<valuelist>** are used, all other values are ignored. **USE_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not a value is used: **TRUE** indicates to use the value, **FALSE** to ignore it. If all values in the neighborhood of a pixel are ignored, the output value is 0.

An **<apply_option>** and its **<valuelist>** specifies whether or not the focal function is calculated for a particular input pixel value. If the option specifies that the function is not to be applied for the value of a particular input pixel, the focal function is not calculated at that pixel, and the function returns the value of the input pixel. **NO_APPLY_AT_VALUE** specifies that the function will not be applied at values in **<valuelist>**. **APPLY_AT_VALUE** specifies that the function is applied only at the values in **<valuelist>**, and at no other values. **APPLY_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not to apply the function at a value: **TRUE** indicates to apply at the value, **FALSE** indicates not to apply at the value.

Data Types:

<focus> may be any numeric type; it is converted to **BINARY**. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	



<use_option> and **<apply_option>** may be used only with **INTEGER** input raster data.

Object Types:

<raster> is a **RASTER**; **<focus>** is a **MATRIX**. **<valuelist1>** and **<valuelist2>** are either **SCALAR**, **TABLE**, or a comma-separated list of **SCALAR**s enclosed in braces. Output is a **RASTER** with same number of layers as **<raster>**.

See Also:

[DENSITY](#)



FOCAL DIVERSITY (Focal Diversity)

Syntax:

```
FOCAL DIVERSITY (<raster>, <focus>)  
  
or  
  
FOCAL DIVERSITY (<raster>, <focus>, <option1> <valuelist1>)  
  
or  
  
FOCAL DIVERSITY (<raster>, <focus>, <option1> <valuelist1>,  
<option2> <valuelist2>)
```

Function Type:

Neighborhood

Description:

Returns number of different values in focal window **<focus>** around each pixel of **<raster>**.

<option1> and **<option2>**, if present, may be either a **<use_option>** or an **<apply_option>**. A **<use_option>** is one of the following:

```
IGNORE_VALUE  
USE_VALUE  
USE_LOOKUP_TABLE
```

An **<apply_option>** is one of the following:

```
NO_APPLY_AT_VALUE  
APPLY_AT_VALUE  
APPLY_LOOKUP_TABLE
```

If **<option1>** and **<option2>** are both present, one of them must be a **<use_option>** and the other must be an **<apply_option>**.

<valuelist1> and **<valuelist2>**, if present, may be either a scalar or table expression, or they may be:

```
{ <value1>, <value2>, ..., <valueN> }
```

where **<value1>**, **<value2>**, etc., are each scalar expressions.

A **<use_option>** and its **<valuelist>** specifies which values are to be used in computing the focal function. Pixels in the neighborhood whose values are specified as being ignored will not be counted in the calculation of the focal function. **IGNORE_VALUE** specifies that the values in **<valuelist>** will not be used. **USE_VALUE** specifies that only the values in **<valuelist>** are used, all other values are ignored. **USE_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not a value is used: **TRUE** indicates to use the value, **FALSE** to ignore it. If all values in the neighborhood of a pixel are ignored, the output value is 0.

An **<apply_option>** and its **<valuelist>** specifies whether or not the focal function is calculated for a particular input pixel value. If the option specifies that the function is not to be applied for the value of a particular input pixel, the focal function is not calculated at that pixel, and the function returns the value of the input pixel. **NO_APPLY_AT_VALUE** specifies that the function will not be applied at values in **<valuelist>**. **APPLY_AT_VALUE** specifies that the function is applied only at the values in **<valuelist>**, and at no other values. **APPLY_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not to apply the function at a value: **TRUE** indicates to apply at the value, **FALSE** indicates not to apply at the value.

Data Types:

<focus> may be any numeric type; it is converted to **BINARY**. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	



<use_option> and **<apply_option>** may be used only with **INTEGER** input raster data.

Object Types:

<raster> is a **RASTER**; **<focus>** is a **MATRIX**. **<valuelist1>** and **<valuelist2>** are either **SCALAR**, **TABLE**, or a comma-separated list of **SCALAR**s enclosed in braces. Output is a **RASTER** with same number of layers as **<raster>**.

See Also:

[DIVERSITY](#)
[GLOBAL DIVERSITY](#)
[STACK DIVERSITY](#)
[BOUNDARY](#)
[ZONAL DIVERSITY](#)



FOCAL MAJORITY (Focal Majority)

Syntax:

```
FOCAL MAJORITY (<raster>, <focus>)

or

FOCAL MAJORITY (<raster>, <focus>, <option1> <valuelist1>)

or

FOCAL MAJORITY (<raster>, <focus>, <option1> <valuelist1>,
<option2> <valuelist2>)

or

FOCAL MAJORITY (<raster>, <focus>, <threshold>)

or

FOCAL MAJORITY (<raster>, <focus>, <threshold>, <option1>
<valuelist1>)

or

FOCAL MAJORITY (<raster>, <focus>, <threshold>, <option1>
<valuelist1>, <option2> <valuelist2>)
```

Function Type:

Neighborhood

Description:

Returns the most commonly occurring value in focal window **<focus>** around pixel of **<raster>**. If **<threshold>** is present, and the ratio of the number of occurrences of the majority value to the size of the focal window is greater than or equal to **<threshold>**, the returned pixel is the majority value. If this ratio is less than **<threshold>**, the returned pixel is the input pixel value.

<option1> and **<option2>**, if present, may be either a **<use_option>** or an **<apply_option>**. A **<use_option>** is one of the following:

```
IGNORE_VALUE
USE_VALUE
USE_LOOKUP_TABLE
```

An **<apply_option>** is one of the following:

```
NO_APPLY_AT_VALUE
APPLY_AT_VALUE
APPLY_LOOKUP_TABLE
```

If **<option1>** and **<option2>** are both present, one of them must be a **<use_option>** and the other must be an **<apply_option>**.

<valuelist1> and **<valuelist2>**, if present, may be either a scalar or table expression, or they may be:

```
{ <value1>, <value2>, ..., <valueN> }
```

where **<value1>**, **<value2>**, etc., are each scalar expressions.

A **<use_option>** and its **<valuelist>** specifies which values are to be used in computing the focal function. Pixels in the neighborhood whose values are specified as being ignored will not be counted in the calculation of the focal function. **IGNORE_VALUE** specifies that the values in **<valuelist>** will not be used. **USE_VALUE** specifies that only the values in **<valuelist>** are used, all other values are ignored. **USE_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not a value is used: **TRUE** indicates to use the value, **FALSE** to ignore it. If all values in the neighborhood of a pixel are ignored, the output value is the same as the input value.

An **<apply_option>** and its **<valuelist>** specifies whether or not the focal function is calculated for a particular input pixel value. If the option specifies that the function is not to be applied for the value of a particular input pixel, the focal function is not calculated at that pixel, and the function returns the value of the input pixel. **NO_APPLY_AT_VALUE** specifies that the function will not be applied at values in **<valuelist>**. **APPLY_AT_VALUE** specifies that the function is applied only at the values in **<valuelist>**, and at no other values. **APPLY_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not to apply the function at a value: **TRUE** indicates to apply at the value, **FALSE** indicates not to apply at the value.

Data Types:

<focus> may be any numeric type; it is converted to **BINARY**. **<threshold>** may be any numeric type, and is converted to **FLOAT**. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	



<use_option> and **<apply_option>** may be used only with **INTEGER** input raster data.

If **<valuelist1>** or **<valuelist2>** is present, they must be **INTEGER** or **BINARY**, and **<raster>** must be **INTEGER**.

Object Types:

<raster> is a **RASTER**; **<focus>** is a **MATRIX**; **<threshold>** is a **SCALAR**. **<valuelist1>** and **<valuelist2>** are either **SCALAR**, **TABLE**, or a comma-separated list of **SCALAR**s enclosed in braces. Output is a **RASTER** with same number of layers as **<raster>**.

See Also:

[MAJORITY](#)
[GLOBAL MAJORITY](#)
[STACK MAJORITY](#)
[ZONAL MAJORITY](#)



FOCAL MAX (Focal Maximum)

Syntax:

```
FOCAL MAX (<raster>, <focus>)

or

FOCAL MAX (<raster>, <focus>, <option1> <valuelist1>)

or

FOCAL MAX (<raster>, <focus>, <option1> <valuelist1>, <option2>
<valuelist2>)
```

Function Type:

Neighborhood

Description:

Returns the maximum of the data file values in focal window **<focus>** around each pixel of **<raster>**.

<option1> and **<option2>**, if present, may be either a **<use_option>** or an **<apply_option>**. A **<use_option>** is one of the following:

```
IGNORE_VALUE
USE_VALUE
USE_LOOKUP_TABLE
```

An **<apply_option>** is one of the following:

```
NO_APPLY_AT_VALUE
APPLY_AT_VALUE
APPLY_LOOKUP_TABLE
```

If **<option1>** and **<option2>** are both present, one of them must be a **<use_option>** and the other must be an **<apply_option>**.

<valuelist1> and **<valuelist2>**, if present, may be either a scalar or table expression, or they may be:

```
{ <value1>, <value2>, ..., <valueN> }
```

where **<value1>**, **<value2>**, etc., are each scalar expressions.

A **<use_option>** and its **<valuelist>** specifies which values are to be used in computing the focal function. Pixels in the neighborhood whose values are specified as being ignored will not be counted in the calculation of the focal function. **IGNORE_VALUE** specifies that the values in **<valuelist>** will not be used. **USE_VALUE** specifies that only the values in **<valuelist>** are used, all other values are ignored. **USE_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not a value is used: **TRUE** indicates to use the value, **FALSE** to ignore it. If all values in the neighborhood of a pixel are ignored, the output value is the same as the input value.

An **<apply_option>** and its **<valuelist>** specifies whether or not the focal function is calculated for a particular input pixel value. If the option specifies that the function is not to be applied for the value of a particular input pixel, the focal function is not calculated at that pixel, and the function returns the value of the input pixel. **NO_APPLY_AT_VALUE** specifies that the function will not be applied at values in **<valuelist>**. **APPLY_AT_VALUE** specifies that the function is applied only at the values in **<valuelist>**, and at no other values. **APPLY_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not to apply the function at a value: **TRUE** indicates to apply at the value, **FALSE** indicates not to apply at the value.

Data Types:

<focus> may be any numeric type; it is converted to **BINARY**. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	



<use_option> and **<apply_option>** may be used only with **INTEGER** input raster data.

Object Types:

<raster> is a **RASTER**; **<focus>** is a **MATRIX**. **<valuelist1>** and **<valuelist2>** are either **SCALAR**, **TABLE**, or a comma-separated list of **SCALAR**s enclosed in braces. Output is a **RASTER** with same number of layers as **<raster>**.

See Also:

[MAX](#)
[GLOBAL MAX](#)
[STACK MAX](#)
[ZONAL MAX](#)



FOCAL MEAN (Focal Mean)

Syntax:

```
FOCAL MEAN (<raster>, <focus>)

or

FOCAL MEAN (<raster>, <focus>, <option1> <valuelist1>)

or

FOCAL MEAN (<raster>, <focus>, <option1> <valuelist1>, <option2>
<valuelist2>)
```

Function Type:

Neighborhood

Description:

Returns mean of pixels in focal window **<focus>** around each pixel of **<raster>**.

<option1> and **<option2>**, if present, may be either a **<use_option>** or an **<apply_option>**. A **<use_option>** is one of the following:

```
IGNORE_VALUE
USE_VALUE
USE_LOOKUP_TABLE
```

An **<apply_option>** is one of the following:

```
NO_APPLY_AT_VALUE
APPLY_AT_VALUE
APPLY_LOOKUP_TABLE
```

If **<option1>** and **<option2>** are both present, one of them must be a **<use_option>** and the other must be an **<apply_option>**.

<valuelist1> and **<valuelist2>**, if present, may be either a scalar or table expression, or they may be:

```
{ <value1>, <value2>, ..., <valueN> }
```

where **<value1>**, **<value2>**, etc., are each scalar expressions.

A **<use_option>** and its **<valuelist>** specifies which values are to be used in computing the focal function. Pixels in the neighborhood whose values are specified as being ignored will not be counted in the calculation of the focal function. **IGNORE_VALUE** specifies that the values in **<valuelist>** will not be used. **USE_VALUE** specifies that only the values in **<valuelist>** are used, all other values are ignored. **USE_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not a value is used: **TRUE** indicates to use the value, **FALSE** to ignore it. If all values in the neighborhood of a pixel are ignored, the output value is the same as the input value.

An **<apply_option>** and its **<valuelist>** specifies whether or not the focal function is calculated for a particular input pixel value. If the option specifies that the function is not to be applied for the value of a particular input pixel, the focal function is not calculated at that pixel, and the function returns the value of the input pixel. **NO_APPLY_AT_VALUE** specifies that the function will not be applied at values in **<valuelist>**. **APPLY_AT_VALUE** specifies that the function is applied only at the values in **<valuelist>**, and at no other values. **APPLY_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not to apply the function at a value: **TRUE** indicates to apply at the value, **FALSE** indicates not to apply at the value.

Data Types:

<focus> may be any numeric type; it is converted to **BINARY**. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	not supported	
STRING	not supported	



<use_option> and **<apply_option>** may be used only with **INTEGER** input raster data.

Object Types:

<raster> is a **RASTER**; **<focus>** is a **MATRIX**. **<valuelist1>** and **<valuelist2>** are either **SCALAR**, **TABLE**, or a comma-separated list of **SCALAR**s enclosed in braces. Output is a **RASTER** with same number of layers as **<raster>**.

See Also:

[MEAN](#)
[GLOBAL MEAN](#)
[STACK MEAN](#)
[ZONAL MEAN](#)



FOCAL MEDIAN (Focal Median)

Syntax:

```
FOCAL MEDIAN (<raster>, <focus>)

or

FOCAL MEDIAN (<raster>, <focus>, <option1> <valuelist1>)

or

FOCAL MEDIAN (<raster>, <focus>, <option1> <valuelist1>, <option2>
<valuelist2>)
```

Function Type:

Neighborhood

Description:

Returns the median of values in focal window **<focus>** around each pixel of **<raster>**.

<option1> and **<option2>**, if present, may be either a **<use_option>** or an **<apply_option>**. A **<use_option>** is one of the following:

```
IGNORE_VALUE
USE_VALUE
USE_LOOKUP_TABLE
```

An **<apply_option>** is one of the following:

```
NO_APPLY_AT_VALUE
APPLY_AT_VALUE
APPLY_LOOKUP_TABLE
```

If **<option1>** and **<option2>** are both present, one of them must be a **<use_option>** and the other must be an **<apply_option>**.

<valuelist1> and **<valuelist2>**, if present, may be either a scalar or table expression, or they may be:

```
{ <value1>, <value2>, ..., <valueN> }
```

where **<value1>**, **<value2>**, etc., are each scalar expressions.

A **<use_option>** and its **<valuelist>** specifies which values are to be used in computing the focal function. Pixels in the neighborhood whose values are specified as being ignored will not be counted in the calculation of the focal function. **IGNORE_VALUE** specifies that the values in **<valuelist>** will not be used. **USE_VALUE** specifies that only the values in **<valuelist>** are used, all other values are ignored. **USE_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not a value is used: **TRUE** indicates to use the value, **FALSE** to ignore it. If all values in the neighborhood of a pixel are ignored, the output value is the same as the input value.

An **<apply_option>** and its **<valuelist>** specifies whether or not the focal function is calculated for a particular input pixel value. If the option specifies that the function is not to be applied for the value of a particular input pixel, the focal function is not calculated at that pixel, and the function returns the value of the input pixel. **NO_APPLY_AT_VALUE** specifies that the function will not be applied at values in **<valuelist>**. **APPLY_AT_VALUE** specifies that the function is applied only at the values in **<valuelist>**, and at no other values. **APPLY_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not to apply the function at a value: **TRUE** indicates to apply at the value, **FALSE** indicates not to apply at the value.

Data Types:

<focus> may be any numeric type; it is converted to **BINARY**. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	



<use_option> and **<apply_option>** may be used only with **INTEGER** input raster data.

Object Types:

<raster> is a **RASTER**; **<focus>** is a **MATRIX**. **<valuelist1>** and **<valuelist2>** are either **SCALAR**, **TABLE**, or a comma-separated list of **SCALAR**s enclosed in braces. Output is a **RASTER** with same number of layers as **<raster>**.

See Also:

[MEDIAN](#)

[GLOBAL MEDIAN](#)

[STACK MEDIAN](#)

[ZONAL MEDIAN](#)



FOCAL MIN (Focal Minimum)

Syntax:

```
FOCAL MIN (<raster>, <focus>)  
  
or  
  
FOCAL MIN (<raster>, <focus>, <option1> <valuelist1>)  
  
or  
  
FOCAL MIN (<raster>, <focus>, <option1> <valuelist1>, <option2>  
<valuelist2>)
```

Function Type:

Neighborhood

Description:

Returns the minimum of values in focal window **<focus>** around each pixel of **<raster>**.

<option1> and **<option2>**, if present, may be either a **<use_option>** or an **<apply_option>**. A **<use_option>** is one of the following:

```
IGNORE_VALUE  
USE_VALUE  
USE_LOOKUP_TABLE
```

An **<apply_option>** is one of the following:

```
NO_APPLY_AT_VALUE  
APPLY_AT_VALUE  
APPLY_LOOKUP_TABLE
```

If **<option1>** and **<option2>** are both present, one of them must be a **<use_option>** and the other must be an **<apply_option>**.

<valuelist1> and **<valuelist2>**, if present, may be either a scalar or table expression, or they may be:

```
{ <value1>, <value2>, ..., <valueN> }
```

where **<value1>**, **<value2>**, etc., are each scalar expressions.

A **<use_option>** and its **<valuelist>** specifies which values are to be used in computing the focal function. Pixels in the neighborhood whose values are specified as being ignored will not be counted in the calculation of the focal function. **IGNORE_VALUE** specifies that the values in **<valuelist>** will not be used. **USE_VALUE** specifies that only the values in **<valuelist>** are used, all other values are ignored. **USE_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not a value is used: **TRUE** indicates to use the value, **FALSE** to ignore it. If all values in the neighborhood of a pixel are ignored, the output value is the same as the input value.

An **<apply_option>** and its **<valuelist>** specifies whether or not the focal function is calculated for a particular input pixel value. If the option specifies that the function is not to be applied for the value of a particular input pixel, the focal function is not calculated at that pixel, and the function returns the value of the input pixel. **NO_APPLY_AT_VALUE** specifies that the function will not be applied at values in **<valuelist>**. **APPLY_AT_VALUE** specifies that the function is applied only at the values in **<valuelist>**, and at no other values. **APPLY_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not to apply the function at a value: **TRUE** indicates to apply at the value, **FALSE** indicates not to apply at the value.

Data Types:

<focus> may be any numeric type; it is converted to **BINARY**. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	



<use_option> and **<apply_option>** may be used only with **INTEGER** input raster data.

Object Types:

<raster> is a **RASTER**; **<focus>** is a **MATRIX**. **<valuelist1>** and **<valuelist2>** are either **SCALAR**, **TABLE**, or a comma-separated list of **SCALAR**s enclosed in braces. Output is a **RASTER** with same number of layers as **<raster>**.

See Also:

[MIN](#)
[GLOBAL MIN](#)
[STACK MIN](#)
[ZONAL MIN](#)



FOCAL MINORITY (Focal Minority)

Syntax:

```
FOCAL MINORITY (<raster>, <focus>)  
  
or  
  
FOCAL MINORITY (<raster>, <focus>, <option1> <valuelist1>)  
  
or  
  
FOCAL MINORITY (<raster>, <focus>, <option1> <valuelist1>,  
<option2> <valuelist2>)
```

Function Type:

Neighborhood

Description:

Returns the least commonly occurring value among in focal window **<focus>** around pixel of **<raster>**.

<option1> and **<option2>**, if present, may be either a **<use_option>** or an **<apply_option>**. A **<use_option>** is one of the following:

```
IGNORE_VALUE  
USE_VALUE  
USE_LOOKUP_TABLE
```

An **<apply_option>** is one of the following:

```
NO_APPLY_AT_VALUE  
APPLY_AT_VALUE  
APPLY_LOOKUP_TABLE
```

If **<option1>** and **<option2>** are both present, one of them must be a **<use_option>** and the other must be an **<apply_option>**.

<valuelist1> and **<valuelist2>**, if present, may be either a scalar or table expression, or they may be:

```
{ <value1>, <value2>, ..., <valueN> }
```

where **<value1>**, **<value2>**, etc., are each scalar expressions.

A **<use_option>** and its **<valuelist>** specifies which values are to be used in computing the focal function. Pixels in the neighborhood whose values are specified as being ignored will not be counted in the calculation of the focal function. **IGNORE_VALUE** specifies that the values in **<valuelist>** will not be used. **USE_VALUE** specifies that only the values in **<valuelist>** are used, all other values are ignored. **USE_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not a value is used: **TRUE** indicates to use the value, **FALSE** to ignore it. If all values in the neighborhood of a pixel are ignored, the output value is the same as the input value.

An **<apply_option>** and its **<valuelist>** specifies whether or not the focal function is calculated for a particular input pixel value. If the option specifies that the function is not to be applied for the value of a particular input pixel, the focal function is not calculated at that pixel, and the function returns the value of the input pixel. **NO_APPLY_AT_VALUE** specifies that the function will not be applied at values in **<valuelist>**. **APPLY_AT_VALUE** specifies that the function is applied only at the values in **<valuelist>**, and at no other values. **APPLY_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not to apply the function at a value: **TRUE** indicates to apply at the value, **FALSE** indicates not to apply at the value.

Data Types:

<focus> may be any numeric type; it is converted to **BINARY**. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	



<use_option> and **<apply_option>** may be used only with **INTEGER** input raster data.

Object Types:

<raster> is a **RASTER**; **<focus>** is a **MATRIX**. **<valuelist1>** and **<valuelist2>** are either **SCALAR**, **TABLE**, or a comma-separated list of **SCALAR**s enclosed in braces. Output is a **RASTER** with same number of layers as **<raster>**.

See Also:

[MINORITY](#)
[GLOBAL MINORITY](#)
[STACK MINORITY](#)



FOCAL RANK (Focal Rank)

Syntax:

```
FOCAL RANK (<raster>, <focus>)

or

FOCAL RANK (<raster>, <focus>, <option1> <valuelist1>)

or

FOCAL RANK (<raster>, <focus>, <option1> <valuelist1>, <option2>
<valuelist2>)
```

Function Type:

Neighborhood

Description:

Returns the number of pixels in the focal window **<focus>** whose value is less than the center pixel, for each pixel of **<raster>**.

<option1> and **<option2>**, if present, may be either a **<use_option>** or an **<apply_option>**. A **<use_option>** is one of the following:

```
IGNORE_VALUE
USE_VALUE
USE_LOOKUP_TABLE
```

An **<apply_option>** is one of the following:

```
NO_APPLY_AT_VALUE
APPLY_AT_VALUE
APPLY_LOOKUP_TABLE
```

If **<option1>** and **<option2>** are both present, one of them must be a **<use_option>** and the other must be an **<apply_option>**.

<valuelist1> and **<valuelist2>**, if present, may be either a scalar or table expression, or they may be:

```
{ <value1>, <value2>, ..., <valueN> }
```

where **<value1>**, **<value2>**, etc., are each scalar expressions.

A **<use_option>** and its **<valuelist>** specifies which values are to be used in computing the focal function. Pixels in the neighborhood whose values are specified as being ignored will not be counted in the calculation of the focal function. **IGNORE_VALUE** specifies that the values in **<valuelist>** will not be used. **USE_VALUE** specifies that only the values in **<valuelist>** are used, all other values are ignored. **USE_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not a value is used: **TRUE** indicates to use the value, **FALSE** to ignore it. If all values in the neighborhood of a pixel are ignored, the output value is 0.

An **<apply_option>** and its **<valuelist>** specifies whether or not the focal function is calculated for a particular input pixel value. If the option specifies that the function is not to be applied for the value of a particular input pixel, the focal function is not calculated at that pixel, and the function returns the value of the input pixel. **NO_APPLY_AT_VALUE** specifies that the function will not be applied at values in **<valuelist>**. **APPLY_AT_VALUE** specifies that the function is applied only at the values in **<valuelist>**, and at no other values. **APPLY_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not to apply the function at a value: **TRUE** indicates to apply at the value, **FALSE** indicates not to apply at the value.

Data Types:

<focus> may be any numeric type; it is converted to **BINARY**. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	INTEGER	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	



<use_option> and **<apply_option>** may be used only with **INTEGER** input raster data.

Object Types:

<raster> is a **RASTER**; **<focus>** is a **MATRIX**. **<valuelist1>** and **<valuelist2>** are either **SCALAR**, **TABLE**, or a comma-separated list of **SCALAR**s enclosed in braces. Output is a **RASTER** with same number of layers as **<raster>**.

See Also:

[RANK](#)



FOCAL SD (Focal Standard Deviation)

Syntax:

```
FOCAL SD (<raster>, <focus>)
```

or

```
FOCAL SD (<raster>, <focus>, <option1> <valuelist1>)
```

or

```
FOCAL SD (<raster>, <focus>, <option1> <valuelist1>, <option2>  
<valuelist2>)
```

Function Type:

Neighborhood

Description:

Returns standard deviation of pixels in focal window **<focus>** around each pixel of **<raster>**.

<option1> and **<option2>**, if present, may be either a **<use_option>** or an **<apply_option>**. A **<use_option>** is one of the following:

```
IGNORE_VALUE  
USE_VALUE  
USE_LOOKUP_TABLE
```

An **<apply_option>** is one of the following:

```
NO_APPLY_AT_VALUE  
APPLY_AT_VALUE  
APPLY_LOOKUP_TABLE
```

If **<option1>** and **<option2>** are both present, one of them must be a **<use_option>** and the other must be an **<apply_option>**.

<valuelist1> and **<valuelist2>**, if present, may be either a scalar or table expression, or they may be:

```
{ <value1>, <value2>, ..., <valueN> }
```

where **<value1>**, **<value2>**, etc., are each scalar expressions.

A **<use_option>** and its **<valuelist>** specifies which values are to be used in computing the focal function. Pixels in the neighborhood whose values are specified as being ignored will not be counted in the calculation of the focal function. **IGNORE_VALUE** specifies that the values in **<valuelist>** will not be used. **USE_VALUE** specifies that only the values in **<valuelist>** are used, all other values are ignored. **USE_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not a value is used: **TRUE** indicates to use the value, **FALSE** to ignore it. If all values in the neighborhood of a pixel are ignored, the output value is 0.

An **<apply_option>** and its **<valuelist>** specifies whether or not the focal function is calculated for a particular input pixel value. If the option specifies that the function is not to be applied for the value of a particular input pixel, the focal function is not calculated at that pixel, and the function returns the value of the input pixel. **NO_APPLY_AT_VALUE** specifies that the function will not be applied at values in **<valuelist>**. **APPLY_AT_VALUE** specifies that the function is applied only at the values in **<valuelist>**, and at no other values. **APPLY_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not to apply the function at a value: **TRUE** indicates to apply at the value, **FALSE** indicates not to apply at the value.

Data Types:

<focus> may be any numeric type; it is converted to **BINARY**. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	not supported	
STRING	not supported	



<use_option> and **<apply_option>** may be used only with **INTEGER** input raster data.

Object Types:

<raster> is a **RASTER**; **<focus>** is a **MATRIX**. **<valuelist1>** and **<valuelist2>** are either **SCALAR**, **TABLE**, or a comma-separated list of **SCALAR**s enclosed in braces. Output is a **RASTER** with same number of layers as **<raster>**.

Notes:

Equivalent to:

FOCAL STANDARD DEVIATION (**<raster>**, **<focus>**)

See Also:

[FOCAL STANDARD DEVIATION](#)

[SD](#)

[GLOBAL SD](#)

[STACK SD](#)

[ZONAL SD](#) (from summary)



FOCAL STANDARD DEVIATION (Focal Standard Deviation)

Syntax:

```
FOCAL STANDARD DEVIATION (<raster>, <focus>)

or

FOCAL STANDARD DEVIATION (<raster>, <focus>, <option1>
<valuelist1>)

or

FOCAL STANDARD DEVIATION (<raster>, <focus>, <option1>
<valuelist1>, <option2> <valuelist2>)
```

Function Type:

Neighborhood

Description:

Returns standard deviation of pixels in focal window **<focus>** around each pixel of **<raster>**.

<option1> and **<option2>**, if present, may be either a **<use_option>** or an **<apply_option>**. A **<use_option>** is one of the following:

```
IGNORE_VALUE
USE_VALUE
USE_LOOKUP_TABLE
```

An **<apply_option>** is one of the following:

```
NO_APPLY_AT_VALUE
APPLY_AT_VALUE
APPLY_LOOKUP_TABLE
```

If **<option1>** and **<option2>** are both present, one of them must be a **<use_option>** and the other must be an **<apply_option>**.

<valuelist1> and **<valuelist2>**, if present, may be either a scalar or table expression, or they may be:

```
{ <value1>, <value2>, ..., <valueN> }
```

where **<value1>**, **<value2>**, etc., are each scalar expressions.

A **<use_option>** and its **<valuelist>** specifies which values are to be used in computing the focal function. Pixels in the neighborhood whose values are specified as being ignored will not be counted in the calculation of the focal function. **IGNORE_VALUE** specifies that the values in **<valuelist>** will not be used. **USE_VALUE** specifies that only the values in **<valuelist>** are used, all other values are ignored. **USE_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not a value is used: **TRUE** indicates to use the value, **FALSE** to ignore it. If all values in the neighborhood of a pixel are ignored, the output value is 0.

An **<apply_option>** and its **<valuelist>** specifies whether or not the focal function is calculated for a particular input pixel value. If the option specifies that the function is not to be applied for the value of a particular input pixel, the focal function is not calculated at that pixel, and the function returns the value of the input pixel. **NO_APPLY_AT_VALUE** specifies that the function will not be applied at values in **<valuelist>**. **APPLY_AT_VALUE** specifies that the function is applied only at the values in **<valuelist>**, and at no other values. **APPLY_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not to apply the function at a value: **TRUE** indicates to apply at the value, **FALSE** indicates not to apply at the value.

Data Types:

<focus> may be any numeric type; it is converted to **BINARY**. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	not supported	
STRING	not supported	



<use_option> and **<apply_option>** may be used only with **INTEGER** input raster data.

Object Types:

<raster> is a **RASTER**; **<focus>** is a **MATRIX**. **<valuelist1>** and **<valuelist2>** are either **SCALAR**, **TABLE**, or a comma-separated list of **SCALAR**s enclosed in braces. Output is a **RASTER** with same number of layers as **<raster>**.

Notes:

Equivalent to:

```
FOCAL SD (<raster>, <focus>)
```

```
FOCAL SD (<raster>, <focus>, <option1> <valuelist1>)
```

```
FOCAL SD (<raster>, <focus>, <option1> <valuelist1>, <option2>  
<valuelist2>)
```

See Also:

[FOCAL SD](#)

[SD](#)

[GLOBAL STANDARD DEVIATION](#)

[STACK STANDARD DEVIATION](#)



FOCAL SUM (Focal Sum)

Syntax:

```
FOCAL SUM (<raster>, <focus>)  
  
or  
  
FOCAL SUM (<raster>, <focus>, <option1> <valuelist1>)  
  
or  
  
FOCAL SUM (<raster>, <focus>, <option1> <valuelist1>, <option2>  
<valuelist2>)
```

Function Type:

Neighborhood

Description:

Returns sum of pixels in focal window **<focus>** around each pixel of **<raster>**.

<option1> and **<option2>**, if present, may be either a **<use_option>** or an **<apply_option>**. A **<use_option>** is one of the following:

```
IGNORE_VALUE  
USE_VALUE  
USE_LOOKUP_TABLE
```

An **<apply_option>** is one of the following:

```
NO_APPLY_AT_VALUE  
APPLY_AT_VALUE  
APPLY_LOOKUP_TABLE
```

If **<option1>** and **<option2>** are both present, one of them must be a **<use_option>** and the other must be an **<apply_option>**.

<valuelist1> and **<valuelist2>**, if present, may be either a scalar or table expression, or they may be:

```
{ <value1>, <value2>, ..., <valueN> }
```

where **<value1>**, **<value2>**, etc., are each scalar expressions.

A **<use_option>** and its **<valuelist>** specifies which values are to be used in computing the focal function. Pixels in the neighborhood whose values are specified as being ignored will not be counted in the calculation of the focal function. **IGNORE_VALUE** specifies that the values in **<valuelist>** will not be used. **USE_VALUE** specifies that only the values in **<valuelist>** are used, all other values are ignored. **USE_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not a value is used: **TRUE** indicates to use the value, **FALSE** to ignore it. If all values in the neighborhood of a pixel are ignored, the output value is 0.

An **<apply_option>** and its **<valuelist>** specifies whether or not the focal function is calculated for a particular input pixel value. If the option specifies that the function is not to be applied for the value of a particular input pixel, the focal function is not calculated at that pixel, and the function returns the value of the input pixel. **NO_APPLY_AT_VALUE** specifies that the function will not be applied at values in **<valuelist>**. **APPLY_AT_VALUE** specifies that the function is applied only at the values in **<valuelist>**, and at no other values. **APPLY_LOOKUP_TABLE** specifies that its **<valuelist>** will be cast to **BINARY** and used as a binary lookup table to determine whether or not to apply the function at a value: **TRUE** indicates to apply at the value, **FALSE** indicates not to apply at the value.

Data Types:

<focus> may be any numeric type; it is converted to **BINARY**. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	FLOAT	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	not supported	
STRING	not supported	



<use_option> and **<apply_option>** may be used only with **INTEGER** input raster data.

Object Types:

<raster> is a **RASTER**; **<focus>** is a **MATRIX**. **<valuelist1>** and **<valuelist2>** are either **SCALAR**, **TABLE**, or a comma-separated list of **SCALAR**s enclosed in braces. Output is a **RASTER** with same number of layers as **<raster>**.

See Also:

[+ \(Addition\)](#)
[SUM](#)
[GLOBAL SUM](#)
[STACK SUM](#)





Global

[GLOBAL DIVERSITY](#) (*Global Diversity*)

[GLOBAL MAJORITY](#) (*Global Majority*)

[GLOBAL MAX](#) (*Global Maximum*)

[GLOBAL MEAN](#) (*Global Mean*)

[GLOBAL MEDIAN](#) (*Global Median*)

[GLOBAL MIN](#) (*Global Minimum*)

[GLOBAL MINORITY](#) (*Global Minority*)

[GLOBAL SD](#) (*Global Standard Deviation*)

[GLOBAL STANDARD DEVIATION](#) (*Global Standard Deviation*)

[GLOBAL SUM](#) (*Global Sum*)



For more information see [Standard Rules](#).



GLOBAL DIVERSITY (Global Diversity)

Syntax:

```
GLOBAL DIVERSITY (<arg1>)
or
GLOBAL DIVERSITY (<arg1>, <ignoreoption>)
or
GLOBAL DIVERSITY (<arg1>, <ignoreoption> <backgroundvalue>)
```

Function Type:

Global

Description:

Computes number of different values in each layer of **<arg1>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value. **<ignoreoption>** and **<backgroundvalue>** may be used only if **<arg1>** is a **RASTER** object.

If **<ignoreoption>** is not present and **<arg1>** is a **RASTER** object, the computation depends on whether **<arg1>** is a previously existing file, or a **RASTER** created within the model. If **<arg1>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<arg1>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<arg1>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option will be used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

The type of **<arg1>** determines the output type:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.

Object Types:

<arg1> may be any object type. If **<arg1>** is a **SCALAR**, **TABLE**, or **MATRIX**, the result is a **SCALAR**. If **<arg1>** is a **RASTER**, the result is a **TABLE** with the same number of rows as **<arg1>** has layers.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**. **<ignoreoption>** and **<backgroundvalue>** may be present only if **<arg1>** is a **RASTER**.

See Also:[DIVERSITY](#)[FOCAL DIVERSITY](#)[STACK DIVERSITY](#)[SET DEFAULT STATISTICS](#)

GLOBAL MAJORITY (Global Majority)

Syntax:

```
GLOBAL MAJORITY (<arg1>)
or
GLOBAL MAJORITY (<arg1>, <ignoreoption>)
or
GLOBAL MAJORITY (<arg1>, <ignoreoption> <backgroundvalue>)
```

Function Type:

Global

Description:

Computes the most commonly occurring value (mode) in each layer of **<arg1>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value. **<ignoreoption>** and **<backgroundvalue>** may be used only if **<arg1>** is a **RASTER** object.

If **<ignoreoption>** is not present and **<arg1>** is a **RASTER** object, the computation depends on whether **<arg1>** is a previously existing file, or a **RASTER** created within the model. If **<arg1>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<arg1>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<arg1>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option will be used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

The type of **<arg1>** determines the output type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.

Object Types:

<arg1> may be any object type. If **<arg1>** is a **SCALAR**, **TABLE**, or **MATRIX**, the result is a **SCALAR**. If **<arg1>** is a **RASTER**, the result is a **TABLE** with the same number of rows as **<arg1>** has layers.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**. **<ignoreoption>** and **<backgroundvalue>** may be present only if **<arg1>** is a **RASTER**.

See Also:[MAJORITY](#)[FOCAL MAJORITY](#)[STACK MAJORITY](#)[SET DEFAULT STATISTICS](#)

GLOBAL MAX (Global Maximum)

Syntax:

```
GLOBAL MAX (<arg1>)
or
GLOBAL MAX (<arg1>, <ignoreoption>)
or
GLOBAL MAX (<arg1>, <ignoreoption> <backgroundvalue>)
```

Function Type:

Global

Description:

Computes the maximum value of each layer of **<arg1>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value. **<ignoreoption>** and **<backgroundvalue>** may be used only if **<arg1>** is a **RASTER** object.

If **<ignoreoption>** is not present and **<arg1>** is a **RASTER** object, the computation depends on whether **<arg1>** is a previously existing file, or a **RASTER** created within the model. If **<arg1>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<arg1>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<arg1>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option will be used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

The type of **<arg1>** determines the output type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	FLOAT	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.

Object Types:

<arg1> may be any object type. If **<arg1>** is a **SCALAR**, **TABLE**, or **MATRIX**, the result is a **SCALAR**. If **<arg1>** is a **RASTER**, the result is a **TABLE** with the same number of rows as **<arg1>** has layers.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**. **<ignoreoption>** and **<backgroundvalue>** may be present only if **<arg1>** is a **RASTER**.

See Also:[MAX](#)[FOCAL MAX](#)[STACK MAX](#)[SET DEFAULT STATISTICS](#)

GLOBAL MEAN (Global Mean)

Syntax:

```
GLOBAL MEAN (<arg1>)
or
GLOBAL MEAN (<arg1>, <ignoreoption>)
or
GLOBAL MEAN (<arg1>, <ignoreoption> <backgroundvalue>)
```

Function Type:

Global

Description:

Computes mean of all elements in each layer of **<arg1>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value. **<ignoreoption>** and **<backgroundvalue>** may be used only if **<arg1>** is a **RASTER** object.

If **<ignoreoption>** is not present and **<arg1>** is a **RASTER** object, the computation depends on whether **<arg1>** is a previously existing file, or a **RASTER** created within the model. If **<arg1>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<arg1>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<arg1>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option will be used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

The type of **<arg1>** determines the output type:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.



Object Types:

<arg1> may be any object type. If **<arg1>** is a **SCALAR**, **TABLE**, or **MATRIX**, the result is a **SCALAR**. If **<arg1>** is a **RASTER**, the result is a **TABLE** with the same number of rows as **<arg1>** has layers.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**. **<ignoreoption>** and **<backgroundvalue>** may be present only if **<arg1>** is a **RASTER**.

See Also:[MEAN](#)[FOCAL MEAN](#)[STACK MEAN](#)[SET DEFAULT STATISTICS](#)

GLOBAL MEDIAN (Global Median)

Syntax:

```
GLOBAL MEDIAN (<arg1>)

or

GLOBAL MEDIAN (<arg1>, <ignoreoption>)

or

GLOBAL MEDIAN (<arg1>, <ignoreoption> <backgroundvalue>)
```

Function Type:

[Global](#)

Description:

Computes the median of each layer of **<arg1>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value. **<ignoreoption>** and **<backgroundvalue>** may be used only if **<arg1>** is a **RASTER** object.

If **<ignoreoption>** is not present and **<arg1>** is a **RASTER** object, the computation depends on whether **<arg1>** is a previously existing file, or a **RASTER** created within the model. If **<arg1>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<arg1>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<arg1>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option will be used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

The type of **<arg1>** determines the output type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.



Object Types:

<arg1> may be any object type. If **<arg1>** is a **SCALAR**, **TABLE**, or **MATRIX**, the result is a **SCALAR**. If **<arg1>** is a **RASTER**, the result is a **TABLE** with the same number of rows as **<arg1>** has layers.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**. **<ignoreoption>** and **<backgroundvalue>** may be present only if **<arg1>** is a **RASTER**.

See Also:[MEDIAN](#)[FOCAL MEDIAN](#)[STACK MEDIAN](#)[SET DEFAULT STATISTICS](#)

GLOBAL MIN (Global Minimum)

Syntax:

```
GLOBAL MIN (<arg1>)
or
GLOBAL MIN (<arg1>, <ignoreoption>)
or
GLOBAL MIN (<arg1>, <ignoreoption> <backgroundvalue>)
```

Function Type:

[Global](#)

Description:

Computes the minimum value of each layer of **<arg1>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value. **<ignoreoption>** and **<backgroundvalue>** may be used only if **<arg1>** is a **RASTER** object.

If **<ignoreoption>** is not present and **<arg1>** is a **RASTER** object, the computation depends on whether **<arg1>** is a previously existing file, or a **RASTER** created within the model. If **<arg1>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<arg1>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<arg1>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option will be used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

The type of **<arg1>** determines the output type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.

Object Types:

<arg1> may be any object type. If **<arg1>** is a **SCALAR**, **TABLE**, or **MATRIX**, the result is a **SCALAR**. If **<arg1>** is a **RASTER**, the result is a **TABLE** with the same number of rows as **<arg1>** has layers.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**. **<ignoreoption>** and **<backgroundvalue>** may be present only if **<arg1>** is a **RASTER**.

See Also:[MIN](#)[FOCAL MIN](#)[STACK MIN](#)[SET DEFAULT STATISTICS](#)

GLOBAL MINORITY (Global Minority)

Syntax:

```
GLOBAL MINORITY (<arg1>)  
  
or  
  
GLOBAL MINORITY (<arg1>, <ignoreoption>)  
  
or  
  
GLOBAL MINORITY (<arg1>, <ignoreoption> <backgroundvalue>)
```

Function Type:

Global

Description:

Computes the least commonly occurring value in each layer of **<arg1>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value. **<ignoreoption>** and **<backgroundvalue>** may be used only if **<arg1>** is a **RASTER** object.

If **<ignoreoption>** is not present and **<arg1>** is a **RASTER** object, the computation depends on whether **<arg1>** is a previously existing file, or a **RASTER** created within the model. If **<arg1>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<arg1>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<arg1>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option will be used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

The type of **<arg1>** determines the output type:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.

Object Types:

<arg1> may be any object type. If **<arg1>** is a **SCALAR**, **TABLE**, or **MATRIX**, the result is a **SCALAR**. If **<arg1>** is a **RASTER**, the result is a **TABLE** with the same number of rows as **<arg1>** has layers.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**. **<ignoreoption>** and **<backgroundvalue>** may be present only if **<arg1>** is a **RASTER**.

See Also:[MINORITY](#)[FOCAL MINORITY](#)[STACK MINORITY](#)[SET DEFAULT STATISTICS](#)

GLOBAL SD (Global Standard Deviation)

Syntax:

```
GLOBAL SD (<arg1>)
or
GLOBAL SD (<arg1>, <ignoreoption>)
or
GLOBAL SD (<arg1>, <ignoreoption> <backgroundvalue>)
```

Function Type:

[Global](#)

Description:

Computes the standard deviation of all elements in each layer of **<arg1>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value. **<ignoreoption>** and **<backgroundvalue>** may be used only if **<arg1>** is a **RASTER** object.

If **<ignoreoption>** is not present and **<arg1>** is a **RASTER** object, the computation depends on whether **<arg1>** is a previously existing file, or a **RASTER** created within the model. If **<arg1>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<arg1>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<arg1>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option will be used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

The type of **<arg1>** determines the output type:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	COLOR	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.

Object Types:

<arg1> may be any object type. If **<arg1>** is a **SCALAR**, **TABLE**, or **MATRIX**, the result is a **SCALAR**. If **<arg1>** is a **RASTER**, the result is a **TABLE** with the same number of rows as **<arg1>** has layers.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**. **<ignoreoption>** and **<backgroundvalue>** may be present only if **<arg1>** is a **RASTER**.

Notes:

Identical to:

GLOBAL STANDARD DEVIATION

See Also:

[GLOBAL STANDARD DEVIATION](#)

[SD](#)

[FOCAL SD](#)

[STACK SD](#)

[SET DEFAULT STATISTICS](#)



GLOBAL STANDARD DEVIATION (Global Standard Deviation)

Syntax:

```
GLOBAL STANDARD DEVIATION (<arg1>)

or

GLOBAL STANDARD DEVIATION (<arg1>, <ignoreoption>)

or

GLOBAL STANDARD DEVIATION (<arg1>, <ignoreoption>
<backgroundvalue>)
```

Function Type:

Global

Description:

Computes the standard deviation of all elements in each layer of **<arg1>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value. **<ignoreoption>** and **<backgroundvalue>** may be used only if **<arg1>** is a **RASTER** object.

If **<ignoreoption>** is not present and **<arg1>** is a **RASTER** object, the computation depends on whether **<arg1>** is a previously existing file, or a **RASTER** created within the model. If **<arg1>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<arg1>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<arg1>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option will be used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

The type of **<arg1>** determines the output type:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	COLOR	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.

Object Types:

<arg1> may be any object type. If **<arg1>** is a **SCALAR**, **TABLE**, or **MATRIX**, the result is a **SCALAR**. If **<arg1>** is a **RASTER**, the result is a **TABLE** with the same number of rows as **<arg1>** has layers.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**. **<ignoreoption>** and **<backgroundvalue>** may be present only if **<arg1>** is a **RASTER**.

Notes:

Identical to:

GLOBAL SD

See Also:

[GLOBAL SD](#)

[STANDARD DEVIATION](#)

[FOCAL STANDARD DEVIATION](#)

[STACK STANDARD DEVIATION](#)

[SET DEFAULT STATISTICS](#)



GLOBAL SUM (Global Sum)

Syntax:

```
GLOBAL SUM (<arg1>)
or
GLOBAL SUM (<arg1>, <ignoreoption>)
or
GLOBAL SUM (<arg1>, <ignoreoption> <backgroundvalue>)
```

Function Type:

[Global](#)

Description:

Computes the total of all elements in each layer of **<arg1>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value. **<ignoreoption>** and **<backgroundvalue>** may be used only if **<arg1>** is a **RASTER** object.

If **<ignoreoption>** is not present and **<arg1>** is a **RASTER** object, the computation depends on whether **<arg1>** is a previously existing file, or a **RASTER** created within the model. If **<arg1>** is a **RASTER** created within the model, the default statistics option is used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

If **<arg1>** is a previously existing raster file, and **<ignoreoption>** is not present, the result will normally be computed from the statistics stored with the file. If all layers of **<arg1>** contain statistics which were computed using all values, or if all layers have statistics which were computed ignoring the same background value, the statistics from the file will be used to compute the result of this function. However, if layers used a different background value to compute statistics, or some layers ignored a background value while others used all values, the default statistics option will be used, as set by the [Preference Editor](#) or the [SET DEFAULT STATISTICS](#) statement.

Data Types:

The type of **<arg1>** determines the output type:

Input	Output	Comments
BINARY	FLOAT	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** may be any numeric type.

Object Types:

<arg1> may be any object type. If **<arg1>** is a **SCALAR**, **TABLE**, or **MATRIX**, the result is a **SCALAR**. If **<arg1>** is a **RASTER**, the result is a **TABLE** with the same number of rows as **<arg1>** has layers.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**. **<ignoreoption>** and **<backgroundvalue>** may be present only if **<arg1>** is a **RASTER**.

See Also:[SUM](#)[FOCAL SUM](#)[STACK SUM](#)[SET DEFAULT STATISTICS](#)



Matrix

[MATDIV](#) (*Matrix Division*)

[MATINV](#) (*Matrix Inverse*)

[MATMUL](#) (*Matrix Multiplication*)

[MATRIXTABLE](#) (*Convert Matrix to Table*)

[MATTRANS](#) (*Matrix Transpose*)

[TABLETOMATRIX](#) (*Convert Table to Matrix*)



For more information see [Standard Rules](#).



MATDIV (Matrix Division)

Syntax: `MATDIV (<matrix1>, <matrix2>)`

Function Type: [Point](#)

Description: Divides **<matrix1>** by **<matrix2>** using standard matrix division.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: Both inputs must be **MATRIX**. The matrices must be square and the same size.

The output will be a **MATRIX** the same size as the input matrices.

See Also: [/ \(Division\)](#)
[MATMUL](#)
[MATINV](#)

MATINV (Matrix Inverse)**Syntax:** `MATINV (<matrix1>)`**Function Type:** [Point](#)**Description:** Returns the inverse of **<matrix1>**.**Data Types:**

Input	Output	Comments
BINARY	not supported	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: The input must be a square **MATRIX**. The output will be a **MATRIX** the same size as the input.**Notes:** Inverse matrices are generally used for solving systems of mathematical equations involving several variables. An inverse matrix can also be used in computing inverse principal components.**See Also:** [INV](#)
[MATDIV](#)
[MATMUL](#)

MATMUL (Matrix Multiplication)

Syntax: MATMUL (<matrix1>, <matrix2>)

Function Type: [Point](#)

Description: Multiplies <matrix1> by <matrix2> using standard matrix multiplication.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types:

Inputs must be either **MATRIX** or **TABLE**. The number of columns in <matrix1> must be the same as the number of rows in <matrix2>.

The output will be a **MATRIX** with number of rows equal to the number of rows in <matrix1>. The number of columns in the output will be the same as the number of columns in <matrix2>.

The matrix product a of two arrays b and c is:

$$a_{ij} = \sum_{k=1}^n b_{ik}c_{kj}$$

where i is the row number and j is the column number.

See Also: [*\(Multiplication\)](#)

[MATINV](#)

[MATDIV](#)

[MATRIXTOTABLE](#)



MATRIXTOTABLE (Convert Matrix to Table)

Syntax: `MATRIXTOTABLE (<matrix1>)`

Function Type: [Point](#)

Description: Converts the one column matrix **<matrix1>** to a table.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	STRING	

Object Types: The input must be a one column **MATRIX**. The output will be a **TABLE** with the same number of rows as the input **MATRIX**.

See Also: [TABLETOMATRIX](#)
[MATMUL](#)



MATTRANS (Matrix Transpose)

Syntax: MATTRANS (<matrix1>)

Function Type: [Point](#)

Description: Returns the transpose of <matrix1>.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	STRING	

Object Types: The input type must be **MATRIX** or **TABLE**. The number of rows in the output **MATRIX** will be the number of columns in <matrix1>. The number of columns in the output will be the number of rows in <matrix1>.

Notes: Use **MATTRANS** to switch the vertical and horizontal orientation of <matrix1>.

See Also: [MATMUL](#)
[EIGENMATRIX](#)
[LINEARCOMB](#)



TABLETOMATRIX (Convert Table to Matrix)

Syntax: `TABLETOMATRIX (<table1>)`

Function Type: [Point](#)

Description: Converts the table **<table1>** to a one column matrix.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	STRING	

Object Types: The input must be a **TABLE**. The output will be a one column **MATRIX** with the same number of rows as the input **TABLE**.

See Also: [MATRIXTOTABLE](#)
[MATMUL](#)





Other

[ABS](#) (*Absolute Value*)

[ANGLE](#) (*Angle*)

[BINARY](#) (*Convert to Binary*)

[CEIL](#) (*Ceiling*)

[COMPLEX](#) (*Convert to Complex*)

[CONJ](#) (*Complex Conjugate*)

[DELTA](#) (*Delta*)

[EVEN](#) (*Test if Even*)

[FLOAT](#) (*Convert to Float*)

[FLOOR](#) (*Floor*)

[GAMMA](#) (*Gamma*)

[IMAG](#) (*Imaginary Part*)

[INTEGER](#) (*Convert to Integer*)

[INV](#) (*Multiplicative Inverse*)

[ODD](#) (*Test if Odd*)

[REAL](#) (*Real Part*)

[ROUND](#) (*Round*)

[SIGN](#) (*Sign*)

[SINC](#) (*Sinc*)

[STEP](#) (*Step*)

[TRUNC](#) (*Truncate*)

[WHOLE](#) (*Test if Whole Number*)



For more information see [Standard Rules](#).



ABS (Absolute Value)

Syntax:

ABS (<arg1>)

Function Type:

Point

Description:

Computes the absolute value of <arg1>. The absolute value of <arg1> is always greater than or equal to zero.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	FLOAT	Computes magnitude, i.e., SQRT (re*re + im*im)
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

Example:

ABS (2) equals 2

ABS (-2) equals 2

ABS ((2, 3)) equals 3.605551275, which is equal to $\sqrt{2^2 + 3^2}$

ANGLE (Angle)

Syntax: `ANGLE (<arg1>)`

Function Type: [Point](#)

Description: Returns the angle for a complex number, i.e.:

ATAN (IMAG (<arg1>) / REAL (<arg1>))

Returns zero for other types.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [CONJ](#)
[ATAN](#)
[IMAG](#)
[REAL](#)



BINARY (Convert to Binary)

Syntax: `BINARY (<arg1>)`

Function Type: [Point](#)

Description: Returns true if non-zero, false if zero.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	BINARY	
COLOR	undefined	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [NOT](#)
[ISNONZERO](#)



CEIL (Ceiling)

Syntax:`CEIL (<arg1>)`**Function Type:**[Point](#)**Description:**Computes the least integer greater than or equal to **<arg1>**.**Data Types:**

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

See Also:[FLOOR](#)[ROUND](#)

COMPLEX (Convert to Complex)

Syntax:

COMPLEX (<arg1>)

or

COMPLEX (<arg1>, <arg2>)

Function Type:

Point

Description:

Converts to **COMPLEX** type. If <arg1> and <arg2> are present, uses <arg1> for the real part and <arg2> for the imaginary part. If only <arg1> is used and <arg1> is **BINARY**, **INTEGER**, or **FLOAT**, <arg1> is used for the real part and zero for the imaginary part.

Data Types:

Input	Output	Comments
BINARY	COMPLEX	
INTEGER	COMPLEX	
FLOAT	COMPLEX	
COMPLEX	COMPLEX	
COLOR	undefined	
STRING	not supported	

Object Types:

All object types, standard rules.

See Also:

[REAL](#)

[IMAG](#)

[ABS](#)

CONJ (Complex Conjugate)**Syntax:** `CONJ (<arg1>)`**Function Type:** [Point](#)**Description:** Returns the conjugate of a complex number, i.e., the real part minus the imaginary part. Returns **<arg1>** for other types.**Data Types:**

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.**Notes:** Equivalent to:
 $\text{REAL}(<\text{arg1}>) - \text{IMAG}(<\text{arg1}>)$ **See Also:** [REAL](#)
[IMAG](#)
[ABS](#)
[ANGLE](#)

DELTA (Delta)

Syntax:

DELTA (<arg1>)

Function Type:

Point

Description:

True if <arg1> is zero, false otherwise.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	BINARY	
COLOR	undefined	
STRING	not supported	

Object Types:

All object types, standard rules.

Equivalent to NOT <arg1>.

See Also:

NOT

ISNONZERO

ISALLTRUE

EVEN (Test if Even)

Syntax: `EVEN (<arg1>)`

Function Type: [Point](#)

Description: Returns true if **<arg1>** is an even number, false otherwise.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	FLOAT inputs converted to INTEGER
COMPLEX	not supported	
COLOR	undefined	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [ODD](#)



FLOAT (Convert to Float)

Syntax: `FLOAT (<arg1>)`

Function Type: [Point](#)

Description: Converts to **FLOAT** type.

Data Types:

Input	Output	Comments
BINARY	FLOAT	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	FLOAT	Returns magnitude: ABS (<arg1>)
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [ABS](#)
[REAL](#)



FLOOR (Floor)

Syntax: `FLOOR (<arg1>)`

Function Type: [Point](#)

Description: Computes the greatest integer less than or equal to **<arg1>**.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [INTEGER](#)
[TRUNC](#)
[CEIL](#)
[ROUND](#)



GAMMA (Gamma)

Syntax:

GAMMA (<arg1>)

Function Type:

[Point](#)

Description:

Computes the gamma function of **<arg1>**. **GAMMA** can be used in statistical analyses.

For an integer n:

$$\text{GAMMA}(n) = (n - 1) !$$

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

See Also:

[! \(Factorial\)](#)



IMAG (Imaginary Part)

Syntax: `IMAG (<arg1>)`

Function Type: [Point](#)

Description: Returns the imaginary part of a complex number. Returns zero for other types.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [REAL](#)
[CONJ](#)
[ABS](#)
[ANGLE](#)



INTEGER (Convert to Integer)

Syntax: `INTEGER (<arg1>)`

Function Type: [Point](#)

Description: Truncates **<arg1>**, returns **INTEGER** type.

Data Types:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	INTEGER	
COMPLEX	INTEGER	Truncated magnitude: INTEGER (ABS (<arg1>))
COLOR	undefined	
STRING	not supported	

Object Types: All object types, standard rules.



INV (Multiplicative Inverse)

Syntax: `INV (<arg1>)`

Function Type: [Point](#)

Description: Computes the multiplicative inverse of **<arg1>**, i.e., **1. / <arg1>**.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [/ \(Division\)](#)
[MATINV](#)



ODD (Test if Odd)

Syntax: ODD (<arg1>)

Function Type: [Point](#)

Description: Returns true if **<arg1>** is an odd number, false otherwise.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	FLOAT inputs converted to INTEGER
COMPLEX	not supported	
COLOR	undefined	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [EVEN](#)

REAL (Real Part)

Syntax:

```
REAL (<arg1>)
```

Function Type:

Point

Description:

Returns the real part of a complex number. Returns **<arg1>** for other types.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

See Also:

[FLOAT](#)

[IMAG](#)

[ABS](#)



ROUND (Round)

Syntax:

ROUND (<arg1>)

Function Type:

Point

Description:

Computes the nearest integer to <arg1>.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

Notes:

ROUND (-0.5) returns 0

ROUND (0.5) returns 1

See Also:

[INTEGER](#)

[TRUNC](#)

[FLOOR](#)

[CEIL](#)



SIGN (Sign)

Syntax:

`SIGN (<arg1>)`

Function Type:

[Point](#)

Description:

Determines the sign of **<arg1>**. Returns 1 if **<arg1>** is positive, 0 if 0, -1 if negative.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

Example:

SIGN (10) equals **1**

See Also:

[STEP](#)



SINC (Sinc)

Syntax:

SINC (<arg1>)

Function Type:

Point

Description:

Returns (**SIN** ($\pi * \text{<arg1>}$)) / ($\pi * \text{<arg1>}$).

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

See Also:

[SIN](#)



STEP (Step)

Syntax:

STEP (<arg1>)

Function Type:

[Point](#)

Description:

Returns true if <arg1> >= 0, false otherwise.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	not supported	
COLOR	undefined	
STRING	not supported	

Object Types:

All object types, standard rules.

Notes:

Equivalent to (<arg1> >= 0).

See Also:

[SIGN](#)



TRUNC (Truncate)

Syntax:

TRUNC (<arg1>)

Function Type:

[Point](#)

Description:

Truncates **<arg1>** to integer by removing the fractional part.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

Example:

TRUNC (8.9) equals 8.0
TRUNC (-8.9) equals -8.0

Notes:

TRUNC and **INTEGER** are similar, however they differ in the data type returned. If you input a **FLOAT** to **TRUNC**, a **FLOAT** will be returned.

See Also:

[INTEGER](#)

[FLOOR](#)

[ROUND](#)

WHOLE (Test if Whole Number)

Syntax: `WHOLE (<arg1>)`

Function Type: [Point](#)

Description: Returns true if **<arg1>** is a whole number (a non-negative integer). Returns false otherwise.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	BINARY	True if real is whole and imaginary is whole
COLOR	undefined	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [INTEGER](#)
[STEP](#)





Relational

EQ (*Equality*)

GE (*Greater Than or Equal*)

GT (*Greater Than*)

LE (*Less Than or Equal*)

LT (*Less Than*)

NE (*Inequality*)

== (*Equality*)

>= (*Greater Than or Equal*)

> (*Greater Than*)

<= (*Less Than or Equal*)

< (*Less Than*)

!= (*Inequality*)

=~ (*Case Insensitive String Equality*)

!~ (*Case Insensitive String Inequality*)

ISALLTRUE (*Test for All Non-zero*)

ISNONZERO (*Test for Non-zero*)



For more information see [Standard Rules](#).



EQ (Equality)

Syntaxes:

```
<arg1> EQ <arg2>  
<arg1> == <arg2>
```

Function Type:

Point

Description:

True if **<arg1>** and **<arg2>** are equal, false otherwise.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	BINARY	
COLOR	undefined	for scalars use ISALLTRUE (<arg1> EQ <arg2>)
STRING	BINARY	

Object Types:

All object types, standard rules.

Notes:

If inputs are **COLOR** scalars, the output data type is not fully supported, but may be used as input to the **ISALLTRUE** function, which will return **BINARY SCALAR**.

See Also:

[NE](#)

[=~](#) (Case Insensitive String Equality)

[MATCHES](#)

[ISALLTRUE](#)



GE (Greater Than or Equal)

Syntaxes:

```
<arg1> GE <arg2>  
<arg1> >= <arg2>
```

Function Type: [Point](#)

Description: True if **<arg1>** is greater than or equal to **<arg2>**, false otherwise.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	not supported	
COLOR	undefined	
STRING	BINARY	Uses lexical ordering

Object Types: All object types, standard rules.

See Also: [GT](#)



GT (Greater Than)

Syntaxes:

```
<arg1> GT <arg2>  
<arg1> > <arg2>
```

Function Type:

[Point](#)

Description:

True if **<arg1>** is greater than **<arg2>**, false otherwise.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	not supported	
COLOR	undefined	
STRING	BINARY	Uses lexical ordering

Object Types:

All object types, standard rules.

See Also:

[GE](#)



LE (Less Than or Equal)

Syntaxes:

```
<arg1> LE <arg2>
```

```
<arg1> <= <arg2>
```

Function Type:

[Point](#)

Description:

True if **<arg1>** is less than or equal to **<arg2>**, false otherwise.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	not supported	
COLOR	undefined	
STRING	BINARY	Uses lexical ordering

Object Types:

All object types, standard rules.

See Also:

[LT](#)



LT (Less Than)

Syntaxes:

```
<arg1> LT <arg2>  
<arg1> < <arg2>
```

Function Type:

[Point](#)

Description:

True if **<arg1>** is less than **<arg2>**, false otherwise.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	not supported	
COLOR	undefined	
STRING	BINARY	uses lexical ordering

Object Types:

All object types, standard rules.

See Also:

[LE](#)



NE (Inequality)**Syntaxes:**

```
<arg1> NE <arg2>
<arg1> != <arg2>
```

Function Type:[Point](#)**Description:**

True if **<arg1>** and **<arg2>** are not equal, false otherwise.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	BINARY	
COLOR	undefined	for scalars use ISNONZERO (<arg1> NE <arg2>)
STRING	BINARY	

Object Types:

All object types, standard rules.

Notes:

If inputs are **COLOR** scalars, the output data type is not fully supported, but may be used as input to the **ISNONZERO** function, which will return **BINARY SCALAR**.

See Also:

[!~](#) (Case Insensitive String Inequality)
[EQ](#)
[ISNONZERO](#)



=~ (Case Insensitive String Equality)

Syntax: `<string1> =~ <string2>`

Function Type: [Point](#)

Description: True if **<string1>** and **<string2>** are equal ignoring upper and lower case differences, false otherwise.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	not supported	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	BINARY	

Object Types: All object types, standard rules.

See Also: [!~](#) (Case Insensitive String Inequality)

[==](#) (Equality)

[MATCHES](#)

!~ (Case Insensitive String Inequality)

Syntax: `<string1> !~ <string2>`

Function Type: [Point](#)

Description: True if **<string1>** and **<string2>** are not equal ignoring upper and lower case differences, false otherwise.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	not supported	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	BINARY	

Object Types: All object types, standard rules.

See Also: [=~](#) (Case Insensitive String Equality)

[!=](#) (Inequality)



ISALLTRUE (Test for All Non-zero)

Syntax: `ISALLTRUE (<arg1>)`

Function Type: [Point](#)

Description: Returns **BINARY SCALAR** regardless of the input type. If any element of the input table or matrix is zero, returns false. Returns true if all elements are non-zero.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	BINARY	
COLOR	BINARY	
STRING	not supported	

Object Types: **SCALAR**, **TABLE**, or **MATRIX** on input, **SCALAR** output.

RASTER input is not supported.

Notes: **ISALLTRUE** will also accept as input the "undefined" data types returned by operations such as:

```
color1 == color2
```

where **color1** and **color2** are **COLOR SCALARs**. In this case,

```
ISALLTRUE (color1 == color2)
```

will return a **BINARY SCALAR** which is true if the two colors are identical, or false if they are not.

See Also: [ISNONZERO](#)

[==](#) (Equality)

[BINARY](#)

ISNONZERO (Test for Non-zero)

Syntax: `ISNONZERO (<arg1>)`

Function Type: [Point](#)

Description: Returns **BINARY SCALAR** regardless of input type. If any element of the input table or matrix is non-zero, returns true. Returns false if all elements are zero.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	BINARY	
FLOAT	BINARY	
COMPLEX	BINARY	
COLOR	BINARY	
STRING	not supported	

Object Types: **SCALAR**, **TABLE**, or **MATRIX** on input, **SCALAR** output.

RASTER input is not supported.

Notes: **ISNONZERO** will also accept as input the "undefined" data types returned by operations such as:

```
color1 != color2
```

where **color1** and **color2** are **COLOR SCALARs**. In this case,

```
ISNONZERO (color1 != color2)
```

will return a **BINARY SCALAR** which is true if the two colors are different, or false if they are not.

See Also: [ISALLTRUE](#)

[!=](#) (Inequality)

[BINARY](#)





Size

[CELLAREA](#) (*Area of Grid Cells*)

[CELLUNITS](#) (*Cell Size Units*)

[CELLX](#) (*X Cell Size*)

[CELLY](#) (*Y Cell Size*)

[LAYERHEIGHT](#) (*Height of Raster Layer*)

[LAYERWIDTH](#) (*Width of Raster Layer*)

[NUMCOLS](#) (*Number of Columns*)

[NUMLAYERS](#) (*Number of Layers*)

[NUMROWS](#) (*Number of Rows*)



For more information see [Standard Rules](#).



CELLAREA (Area of Grid Cells)

Syntax:)>

CELLAREA (<units

Function Type:

[Point](#)

Description:

Returns the area of one pixel in the units specified. The area is derived from the current Working Window cell size. <units> is a **STRING** which specifies an area unit from the file <\$IMAGE_HOME>/etc/units.dat, such as “**hectares**” or “**acres**.” Also, if <units> is either “**pixels**” or “**none**,” **1** is returned. If an area unit is specified and the Working Window is not georeferenced, **0** is returned. If <units> is not recognized as a area unit, “**pixels**,” or “**none**,” **0** is returned.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	not supported	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	FLOAT	

Object Types:

Input and output are **SCALAR**.

See Also:

[CELLX](#)

[CELLY](#)



CELLUNITS (Cell Size Units)

Syntax: CELLUNITS

Function Type: [Point](#)

Description: Returns a **STRING** specifying the distance units used in the current Working Window. If the current Working Window is not georeferenced, “**None**” is returned.

Data Types:

Input	Output	Comments
none	STRING	

Object Types: Output is **SCALAR**.

See Also: [CELLX](#)
[CELLY](#)



CELLX (X Cell Size)

Syntax:

CELLX (<units>)

Function Type:

[Point](#)

Description:

Returns the X cell size of one pixel in the units specified. The cell width is from the current Working Window. <units> is a **STRING** which specifies a distance unit from the file <\$IMAGINE_HOME>/etc/units.dat, such as “meters” or “feet.” Also, if <units> is either “pixels” or “none,” 1 is returned. If a distance unit is specified and the Working Window is not georeferenced, 0 is returned. If <units> is not recognized as a distance unit, “pixels,” or “none,” 0 is returned.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	not supported	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	FLOAT	

Object Types:

Input and output are **SCALAR**.

See Also:

[CELLAREA](#)

[CELLUNITS](#)

[CELLY](#)

CELLY (Y Cell Size)

Syntax:

CELLY (<units>)

Function Type:

Point

Description:

Returns the Y cell size of one pixel in the units specified. The cell height is from the current Working Window. <units> is a **STRING** which specifies a distance unit from the file <\$IMAGINE_HOME>/etc/units.dat, such as “meters” or “feet.” Also, if <units> is either “pixels” or “none,” **1** is returned. If a distance unit is specified and the Working Window is not georeferenced, **0** is returned. If <units> is not recognized as a distance unit, “pixels,” or “none,” **0** is returned.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	not supported	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	FLOAT	

Object Types:

Input and output are **SCALAR**.

See Also:

CELLAREA
CELLUNITS
CELLX



LAYERHEIGHT (Height of Raster Layer)

Syntax: `LAYERHEIGHT (<arg1>)`

Function Type: [Point](#)

Description: Returns the height in pixels of the first layer of the stack of raster file layers associated with **<arg1>**. If there is no raster file layer associated with **<arg1>**, **0** is returned.

Data Types:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	INTEGER	
COMPLEX	INTEGER	
COLOR	INTEGER	
STRING	INTEGER	

Object Types: Returns **0** if input is any type other than **RASTER**.

See Also: [NUMROWS](#)
[NUMLAYERS](#)
[LAYERWIDTH](#)

LAYERWIDTH (Width of Raster Layer)

Syntax: `LAYERWIDTH (<arg1>)`

Function Type: [Point](#)

Description: Returns the width in pixels of the first layer of the stack of raster file layers associated with **<arg1>**. If there is no raster file layer associated with **<arg1>**, **0** is returned.

Data Types:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	INTEGER	
COMPLEX	INTEGER	
COLOR	INTEGER	
STRING	INTEGER	

Object Types: Returns **0** if input is any type other than **RASTER**.

See Also: [NUMCOLS](#)
[NUMLAYERS](#)
[LAYERHEIGHT](#)



NUMCOLS (Number of Columns)

Syntax: NUMCOLS (<arg1>)

Function Type: [Point](#)

Description: Returns the number of columns in an object.

Data Types:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	INTEGER	
COMPLEX	INTEGER	
COLOR	INTEGER	
STRING	INTEGER	

Object Types: All input object types supported, see Notes for **RASTER** type. Output object type is **SCALAR**.

Notes: Always returns 1 for **SCALAR** or **TABLE**. Returns the number of columns for **MATRIX**. Returns the width of the Working Window for **RASTER** input, which may not be the same as width of the associated file layer. Earlier versions of this function returned the tile size for **RASTER** input rather than the width of the Working Window.

See Also: [NUMROWS](#)
[NUMLAYERS](#)
[LAYERWIDTH](#)



NUMLAYERS (Number of Layers)**Syntax:** `NUMLAYERS (<arg1>)`**Function Type:** [Point](#)**Description:** Returns number of layers in object.**Data Types:**

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	INTEGER	
COMPLEX	INTEGER	
COLOR	INTEGER	Return value is 3
STRING	INTEGER	

Object Types: All input object types supported. Output object type is **SCALAR**.**Notes:** Returns 1 for **SCALAR**, **TABLE**, or **MATRIX** of any type other than **COLOR**. Returns 3 for **COLOR** objects. Returns the number of layers for **RASTER** object.**See Also:** [NUMCOLS](#)
[NUMROWS](#)

NUMROWS (Number of Rows)

Syntax: NUMROWS (<arg1>)

Function Type: [Point](#)

Description: Returns the number of rows in an object.

Data Types:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	INTEGER	
COMPLEX	INTEGER	
COLOR	INTEGER	
STRING	INTEGER	

Object Types: All input object types supported, see Notes for **RASTER** type. Output object type is **SCALAR**.

Notes: Always returns 1 for **SCALAR**. Returns the number of rows for **TABLE** and **MATRIX**. Returns the height of the Working Window for **RASTER** input, which may not be the same as height of the associated file layer. Earlier versions of this function returned the tile size for **RASTER** input rather than the height of the Working Window

See Also: [NUMCOLS](#)
[NUMLAYERS](#)
[LAYERHEIGHT](#)

Stack

[STACK DIVERSITY](#) (*Stack Diversity*)

[STACK MAJORITY](#) (*Stack Majority*)

[STACK MAX](#) (*Stack Maximum*)

[STACK MEAN](#) (*Stack Mean*)

[STACK MEDIAN](#) (*Stack Median*)

[STACK MIN](#) (*Stack Minimum*)

[STACK MINORITY](#) (*Stack Minority*)

[STACK SD](#) (*Stack Standard Deviation*)

[STACK STANDARD DEVIATION](#) (*Stack Standard Deviation*)

[STACK SUM](#) (*Stack Sum*)



For more information see [Standard Rules](#).



STACK DIVERSITY (Stack Diversity)

Syntax: `STACK DIVERSITY (<arg1>)`

Function Type: [Point](#)

Description: Returns a single layer each of whose pixels contain the number of different values for that pixel among the layers of the input.

Data Types:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: Input must be RASTER with at least two layers. Result is single layer raster.

See Also: [DIVERSITY](#)
[FOCAL DIVERSITY](#)
[GLOBAL DIVERSITY](#)

STACK MAJORITY (Stack Majority)

Syntax: `STACK MAJORITY (<arg1>)`

Function Type: [Point](#)

Description: Returns a single layer each of whose pixels contain the most commonly occurring value for that pixel among the layers of the input.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: Input must be RASTER with at least two layers. Result is single layer raster.

See Also: [MAJORITY](#)
[FOCAL MAJORITY](#)
[GLOBAL MAJORITY](#)



STACK MAX (Stack Maximum)

Syntax: `STACK MAX (<arg1>)`

Function Type: [Point](#)

Description: Returns a single layer each of whose pixels contain the maximum value for that pixel among the layers of the input.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: Input must be RASTER with at least two layers. Result is single layer raster.

See Also: [MAX](#)
[FOCAL MAX](#)
[GLOBAL MAX](#)



STACK MEAN (Stack Mean)

Syntax: `STACK MEAN (<arg1>)`

Function Type: [Point](#)

Description: Returns a single layer each of whose pixels contain the mean value for that pixel among the layers of the input.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	converted to FLOAT
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	not supported	
STRING	not supported	

Object Types: Input must be RASTER with at least two layers. Result is single layer raster.

See Also: [MEAN](#)
[FOCAL MEAN](#)
[GLOBAL MEAN](#)



STACK MEDIAN (Stack Median)

Syntax: `STACK MEDIAN (<arg1>)`

Function Type: [Point](#)

Description: Returns a single layer each of whose pixels contain the median value for that pixel among the layers of the input.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: Input must be RASTER with at least two layers. Result is single layer raster.

Notes: The median is the number in the middle of a set of values. If there is an even number of input layers, then **MEDIAN** will return the next higher value in the set of values.

See Also: [MEDIAN](#)
[FOCAL MEDIAN](#)
[GLOBAL MEDIAN](#)

STACK MIN (Stack Minimum)

Syntax: `STACK MIN (<arg1>)`

Function Type: [Point](#)

Description: Returns a single layer each of whose pixels contain the minimum value for that pixel among the layers of the input.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: Input must be RASTER with at least two layers. Result is single layer raster.

See Also: [MIN](#)
[FOCAL MIN](#)
[GLOBAL MIN](#)



STACK MINORITY (Stack Minority)

Syntax: `STACK MINORITY (<arg1>)`

Function Type: [Point](#)

Description: Returns a single layer each of whose pixels contain the least commonly occurring value for that pixel among the layers of the input.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: Input must be RASTER with at least two layers. Result is single layer raster.

See Also: [MINORITY](#)
[FOCAL MINORITY](#)
[GLOBAL MINORITY](#)



STACK SD (Stack Standard Deviation)**Syntax:** `STACK SD (<arg1>)`**Function Type:** [Point](#)**Description:** Returns a single layer each of whose pixels contain the standard deviation of the values for that pixel from all the layers of the input.**Data Types:**

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	converted to FLOAT
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	not supported	
STRING	not supported	

Object Types: Input must be RASTER with at least two layers. Result is single layer raster.**Notes:** Identical to:`STACK STANDARD DEVIATION (<arg1>)`
See Also: [SD](#)
[FOCAL SD](#)
[GLOBAL SD](#)


STACK STANDARD DEVIATION

Syntax: `STACK STANDARD DEVIATION (<arg1>)`

Function Type: [Point](#)

Description: Returns a single layer each of whose pixels contain the standard deviation of the values for that pixel from all the layers of the input. The standard deviation is a measure of how widely values are dispersed from the average value (mean).

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	converted to FLOAT
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	not supported	
STRING	not supported	

Object Types: Input must be RASTER with at least two layers. Result is single layer raster.

Notes: Identical to:

`STACK SD (<arg1>)`

Standard deviation uses the following formula:

$$\sqrt{\frac{n \sum x^2 - (\sum x)^2}{n(n-1)}}$$

See Also: [SD](#)
[FOCAL STANDARD DEVIATION](#)
[GLOBAL STANDARD DEVIATION](#)



STACK SUM (Stack Sum)

Syntax: `STACK SUM (<arg1>)`

Function Type: [Point](#)

Description: Returns a single layer each of whose pixels contain the sum of the values for that pixel over all the layers of the input.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	not supported	
STRING	not supported	

Object Types: Input must be RASTER with at least two layers. Result is single layer raster.

See Also: [+ \(Addition\)](#)
[FOCAL SUM](#)
[GLOBAL SUM](#)





Statistical (Local)

DENSITY (*Local Density*)

DIVERSITY (*Local Diversity*)

MAJORITY (*Local Majority*)

MAX (*Local Maximum*)

MEAN (*Local Mean*)

MEDIAN (*Local Median*)

MIN (*Local Minimum*)

MINORITY (*Local Minority*)

RANK (*Local Rank*)

SD (*Local Standard Deviation*)

STANDARD DEVIATION (*Local Standard Deviation*)

SUM (*Local Sum*)



For more information see [Standard Rules](#).



DENSITY (Local Density)

Syntax: `DENSITY (<arg1>, <arg2>, <arg3>, ...)`

Function Type: [Point](#)

Description: Returns the number of occurrences of value of **<arg1>** among input values. This function requires at least two inputs.

Data Types:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [FOCAL DENSITY](#)

DIVERSITY (Local Diversity)

Syntax: `DIVERSITY (<arg1>, <arg2>, <arg3>, ...)`

Function Type: [Point](#)

Description: Returns the number of different values among inputs. This function requires at least two inputs.

Data Types:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [FOCAL DIVERSITY](#)
[GLOBAL DIVERSITY](#)
[STACK DIVERSITY](#)



MAJORITY (Local Majority)

Syntax: MAJORITY (<arg1>, <arg2>, <arg3>, ...)

Function Type: [Point](#)

Description: Returns the most commonly occurring value among the given input values. This function requires at least two inputs.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: All object types, standard rules.

Example: MAJORITY (10, 7, 7, 29.9, 2) equals 7

See Also: [FOCAL MAJORITY](#)
[GLOBAL MAJORITY](#)
[STACK MAJORITY](#)

MAX (Local Maximum)

Syntax: MAX (<arg1>, <arg2>, <arg3>, ...)

Function Type: [Point](#)

Description: Returns the maximum value in the list of input arguments. This function requires at least two inputs.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

Example: MAX (10, 7, 9, 27, 2) equals 27

See Also: [FOCAL MAX](#)
[GLOBAL MAX](#)
[STACK MAX](#)



MEAN (Local Mean)

Syntax:

MEAN (<arg1>, <arg2>, <arg3>, ...)

Function Type:

[Point](#)

Description:

Returns the mean of input values. This function requires at least two inputs.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	converted to FLOAT
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

See Also:

[FOCAL MEAN](#)

[GLOBAL MEAN](#)

[STACK MEAN](#)



MEDIAN (Local Median)

Syntax: `MEDIAN (<arg1>, <arg2>, <arg3>, ...)`

Function Type: [Point](#)

Description: Returns the median of the given input values. This function requires at least two inputs.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

Example: MEDIAN (1, 2, 3, 4, 5) equals 3
 MEDIAN (1, 2, 3, 4, 5, 6) equals 4

Notes: The median is the number in the middle of a set of numbers. If there is an even set of numbers, then **MEDIAN** will return the next higher number in the set of numbers.

See Also: [FOCAL MEDIAN](#)
[GLOBAL MEDIAN](#)
[STACK MEDIAN](#)



MIN (Local Minimum)

Syntax:

MIN (<arg1>, <arg2>, <arg3>, ...)

Function Type:

[Point](#)

Description:

Returns the minimum value of the given input arguments. This function requires at least two inputs.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	COLOR	
STRING	not supported	

Object Types:

All object types, standard rules.

Example:

MIN (10, 7, 9, 27, 2) equals 2

See Also:

[FOCAL MIN](#)

[GLOBAL MIN](#)

[STACK MIN](#)



MINORITY (Local Minority)

Syntax: `MINORITY (<arg1>, <arg2>, <arg3>, ...)`

Function Type: [Point](#)

Description: Returns the least commonly occurring value among the given input values. This function requires at least two inputs.

Data Types:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: All object types, standard rules.

Example: `MINORITY (1, 1, 3, 4, 4, 5, 5)` equals 3

See Also: [FOCAL MINORITY](#)
[GLOBAL MINORITY](#)
[STACK MINORITY](#)



RANK (Local Rank)

Syntax:

RANK (<arg1>, <arg2>, <arg3>, ...)

Function Type:

[Point](#)

Description:

Returns the number of inputs whose value is less than the value of **<arg1>**. This function requires at least two inputs.

Data Types:

Input	Output	Comments
BINARY	INTEGER	
INTEGER	INTEGER	
FLOAT	INTEGER	
COMPLEX	not supported	
COLOR	undefined	
STRING	not supported	

Object Types:

All object types, standard rules.

Example:

RANK (5, 3, 6, 2, 1, 7) equals 3

See Also:

[FOCAL RANK](#)



SD (Local Standard Deviation)

Syntax: `SD (<arg1>, <arg2>, <arg3>, ...)`

Function Type: [Point](#)

Description: Returns the standard deviation value of input arguments. This function requires at least two inputs.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	converted to FLOAT
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

Notes: Identical to:

`STANDARD DEVIATION (<arg1>, <arg2>, <arg3>, ...)`

See Also: [STANDARD DEVIATION](#)

[FOCAL SD](#)

[GLOBAL SD](#)

[STACK SD](#)



STANDARD DEVIATION (Local Standard Deviation)

Syntax: STANDARD DEVIATION (<arg1>, <arg2>, <arg3>, ...)

Function Type: [Point](#)

Description: Returns the standard deviation value of input arguments. The standard deviation is a measure of how widely values are dispersed from the average value (mean). This function requires at least two inputs.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	converted to FLOAT
FLOAT	FLOAT	
COMPLEX	FLOAT	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

Example: STANDARD DEVIATION (1395, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) equals **34.44544801405389**

Notes: Identical to:
SD (<arg1>, <arg2>, <arg3>, ...)
Standard deviation uses the following formula:

$$\sqrt{\frac{n \sum x^2 - (\sum x)^2}{n(n-1)}}$$

See Also: [SD](#)
[FOCAL STANDARD DEVIATION](#)
[GLOBAL STANDARD DEVIATION](#)
[STACK STANDARD DEVIATION](#)



SUM (Local Sum)

Syntax: `SUM (<arg1>, <arg2>, <arg3>, ...)`

Function Type: [Point](#)

Description: Returns the sum of all input arguments. This function requires at least two inputs.

Data Types:

Input	Output	Comments
BINARY	BINARY	Equivalent to logical OR of inputs
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

Example: SUM (3, 2) equals 5

See Also: [+ \(Addition\)](#)
[FOCAL SUM](#)
[GLOBAL SUM](#)
[STACK SUM](#)





String

// (Concatenation)

CAT (Concatenate Strings)

LENGTH (Length of String)

LOWERCASE (Lowercase Conversion)

MATCHES (String Wildcard Match)

UPPERCASE (Uppercase Conversion)



For more information see [Standard Rules](#).



CAT (Concatenate Strings)

Syntaxes: CAT (<string1>, <string2>)
 <string1> // <string2>

Function Type: [Point](#)

Description: Concatenate **<string2>** to the end of **<string1>**.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	not supported	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	STRING	

Object Types: All object types, standard rules.

Notes: Equivalent to:
 <string1> // <string2>

See Also: [LENGTH](#)

LENGTH (Length of String)

Syntax: `LENGTH (<string>)`

Function Type: [Point](#)

Description: Find the number of characters in a string.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	not supported	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	INTEGER	

Object Types: All object types, standard rules.

See Also: [CAT](#)



LOWERCASE (Lowercase Conversion)

Syntax: `LOWERCASE (<string>)`

Function Type: [Point](#)

Description: Convert the characters in a string to lowercase.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	not supported	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	STRING	

Object Types: All object types, standard rules.

See Also: [UPPERCASE](#)



MATCHES (String Wildcard Match)

Syntax: `<arg1> MATCHES <arg2>`

Function Type: [Point](#)

Description: True if **<arg2>** matches the wildcard test string in **<arg1>**.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	not supported	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	BINARY	

Object Types: All object types, standard rules.

Notes: A "*" in **<arg1>** may match any number of characters (including zero) in **<arg2>**. A "?" in **<arg1>** will match a single character in **<arg2>**.

See Also: [EQ](#)
[=~ \(Case Insensitive String Equality\)](#)



UPPERCASE (Uppercase Conversion)

Syntax: `UPPERCASE (<string>)`

Function Type: [Point](#)

Description: Convert the characters in a string to uppercase.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	not supported	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	STRING	

Object Types: All object types, standard rules.

See Also: [LOWERCASE](#)



Surface

[ASPECT](#) (*Aspect*)

[DEGREE SLOPE](#) (*Degree Slope*)

[PERCENT SLOPE](#) (*Percent Slope*)

[RELIEF](#)



For more information see [Standard Rules](#).



ASPECT (Aspect)

Syntax: ASPECT (<raster>)

Function Type: [Neighborhood](#)

Description: Computes aspect in degrees based on a 3 x 3 neighborhood around each pixel. <raster> is assumed to contain elevation values.

Output value 0 is due north with degrees increasing clockwise. Output value 90 is due east, 180 due south, and 270 due west. Output value 361 is flat.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: <raster> is a **RASTER**. Output is a **RASTER** with same number of layers as <raster>.

Notes: See [ASPECT](#) for computation of slope.

See Also: [PERCENT SLOPE](#)
[DEGREE SLOPE](#)
[RELIEF](#)



DEGREE SLOPE (Degree Slope)

Syntax:

```
DEGREE SLOPE (<raster>, <units>)
```

or

```
DEGREE SLOPE (<raster>, <xsize>, <ysize>)
```

Function Type:

[Neighborhood](#)

Description:

Computes the slope in degrees based on a 3 x 3 neighborhood around each pixel. **<raster>** is assumed to contain elevation values.

Either **<units>** or **<xsize>** and **<ysize>** determine the relationship between the units of elevation in the input **<raster>** and the ground unit pixel size in the Working Window.

<units>, if present, must be a **STRING** constant containing the name of the units of elevation. **<units>** should be "meters," "feet," "km," "yards," etc. A complete list of supported units is contained in the file **<\$IMAGINE_HOME>/etc/units.dat**. If **<units>** is used, the Working Window must be georeferenced, and the units of elevation must be compatible with the pixel size units, i.e., they must both be distance units or both be angle units.

<xsize> and **<ysize>**, if present, are numeric scalars which specify the X and Y pixel size, respectively, in the same units used for elevation in **<raster>**. The Working Window does not have to be georeferenced when **<xsize>** and **<ysize>** are used.

Data Types:

<units> is a **STRING** constant. **<xsize>** and **<ysize>** are numeric scalars. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types:

<raster> is a **RASTER**; **<units>** is a **STRING** constant. **<xsize>** and **<ysize>** are **SCALAR**.

Output is a **RASTER** with same number of layers as **<raster>**.

Notes:

See [Slope - Degrees](#) for computation of slope.

See Also:

[PERCENT SLOPE](#)
[RELIEF](#)



PERCENT SLOPE (Percent Slope)

Syntax:

```
PERCENT SLOPE (<raster>, <units>)
```

or

```
PERCENT SLOPE (<raster>, <xsize>, <ysize>)
```

Function Type:

[Neighborhood](#)

Description:

Computes the slope as a percentage based on a 3 x 3 neighborhood around each pixel. **<raster>** is assumed to contain elevation values.

Either **<units>** or **<xsize>** and **<ysize>** determine the relationship between the units of elevation in the input **<raster>** and the ground unit pixel size in the Working Window.

<units>, if present, must be a **STRING** constant containing the name of the units of elevation. **<units>** should be "meters," "feet," "km," "yards," etc. A complete list of supported units is contained in the file **<\$IMAGINE_HOME>/etc/units.dat**. If **<units>** is used, the Working Window must be georeferenced, and the units of elevation must be compatible with the pixel size units, i.e., they must both be distance units, or both be angle units.

<xsize> and **<ysize>**, if present, are numeric scalars which specify the x and y pixel size, respectively, in the same units used for elevation in **<raster>**. The Working Window does not have to be georeferenced when **<xsize>** and **<ysize>** are used.

Data Types:

<units> is a **STRING** constant. **<xsize>** and **<ysize>** are numeric scalars. The type of **<raster>** determines the output type.

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types:

<raster> is a **RASTER**; **<units>** is a **STRING** constant. **<xsize>** and **<ysize>** are **SCALAR**.

Output is a **RASTER** with same number of layers as **<raster>**.

Notes:

See [Slope - Percent](#) for computation of slope.

See Also:

[DEGREE SLOPE](#)
[RELIEF](#)



RELIEF (Shaded Relief)

Syntax:

```
RELIEF (<raster>, <azimuth>, <elevation>, <ambient>, <units>)
```

or

```
RELIEF(<raster>, <azimuth>, <elevation>, <ambient>, <xsize>, <ysize>)
```

Function Type:

[Neighborhood](#)

Description:

Computes shaded relief based on a 3 x 3 neighborhood around each pixel. **<raster>** is assumed to contain elevation values.

Either **<units>** or **<xsize>** and **<ysize>** determine the relationship between the units of elevation in the input **<raster>** and the ground unit pixel size in the Working Window.

<units>, if present, must be a **STRING** constant containing the name of the units of elevation. **<units>** should be "meters," "feet," "km," "yards," etc. A complete list of supported units is contained in the file **<\$IMAGINE_HOME>/etc/units.dat**. If **<units>** is used, the Working Window must be georeferenced, and the units of elevation must be compatible with the pixel size units, i.e., they must both be distance units, or both be angle units.

<xsize> and **<ysize>**, if present, are numeric scalars which specify the X and Y pixel size, respectively, in the same units used for elevation in **<raster>**. The Working Window does not have to be georeferenced when **<xsize>** and **<ysize>** are used.

Data Types:

<units> is a **STRING** constant. **<azimuth>**, **<elevation>**, **<ambient>**, **<xsize>**, and **<ysize>** are numeric scalars. The type of **<raster>** determines the output type:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types:

<raster> is a **RASTER**; **<units>** is a **STRING** constant. **<azimuth>**, **<elevation>**, **<ambient>**, **<xsize>**, and **<ysize>** are **SCALAR**.

Output is a **RASTER** with same number of layers as **<raster>**.

See Also:

[ASPECT](#)

[PERCENT SLOPE](#)

[DEGREE SLOPE](#)





Trigonometric

[ACOS](#) (*Arccosine*)

[ACOSH](#) (*Hyperbolic Arccosine*)

[ASIN](#) (*Arcsine*)

[ASINH](#) (*Hyperbolic Arcsine*)

[ATAN](#) (*Arctangent*)

[COS](#) (*Cosine*)

[COSH](#) (*Hyperbolic Cosine*)

[SIN](#) (*Sine*)

[SINH](#) (*Hyperbolic Sine*)

[TAN](#) (*Tangent*)

[TANH](#) (*Hyperbolic Tangent*)



For more information see [Standard Rules](#).



ACOS (Arccosine)

Syntax: `ACOS (<arg1>)`

Function Type: [Point](#)

Description: Computes the arccosine of **<arg1>**. The arccosine is the angle whose cosine is **<arg1>**. The return angle is given in radians. To convert radians to degrees, use the following formula:

$$\text{radians} \times 180/\pi = \text{degrees}$$

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

Example: `ACOS (-0.5)` equals 2.094 (or $2\pi/3$ radians)

`ACOS (-0.5) x 180/ π` equals 120 degrees

See Also: [ASIN](#)
[COS](#)
[ACOSH](#)

ACOSH (Hyperbolic Arccosine)**Syntax:** `ACOSH (<arg1>)`**Function Type:** [Point](#)**Description:** Computes the hyperbolic arccosine of **<arg1>**.**Data Types:**

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.**Example:** `ACOSH (1)` equals 0
See Also: [COSH](#)
[ACOS](#)
[ASINH](#)


ASIN (Arcsine)

Syntax: `ASIN (<arg1>)`

Function Type: [Point](#)

Description: Computes the arcsine of **<arg1>**. The arcsine is the angle whose sine is **<arg1>**. The returned value is in radians. To convert radians to degrees, use the following formula:

$$\text{radians} \times 180/\pi = \text{degrees}$$

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [SIN](#)
[ASINH](#)



ASINH (Hyperbolic Arcsine)

Syntax: `ASINH (<arg1>)`

Function Type: [Point](#)

Description: Computes the hyperbolic arcsine of **<arg1>**.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

See Also: [SINH](#)
[ASIN](#)



ATAN (Arctangent)

Syntax: ATAN (<arg1>)

Function Type: [Point](#)

Description: Computes the arctangent of <arg1>. The arctangent is the angle whose tangent is <arg1>. The result is in radians. To convert radians to degrees, use the following formula:

$$\text{radians} \times 180/\pi = \text{degrees}$$

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

Example: ATAN (1) equals 0.785 ($\pi/4$ radians)

ATAN (1) \times 180/ π equals 45 degrees

See Also: [TAN](#)



COS (Cosine)**Syntax:** `COS (<arg1>)`**Function Type:** [Point](#)**Description:** Computes the cosine of **<arg1>**. Enter **<arg1>** in radians. To convert degrees to radians, use the following formula:

$$\text{degrees} \times \pi/180 = \text{radians}$$

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.**Example:** `COS (1.047197551196598)` equals 0.5`COS (60 x π /180)` equals 0.5, the cosine of 60 degrees**See Also:** [SIN](#)
[ACOS](#)
[COSH](#)

COSH (Hyperbolic Cosine)

Syntax: COSH (<arg1>)

Function Type: [Point](#)

Description: Computes the hyperbolic cosine of <arg1>.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

Example: COSH (4) equals 27.308233

Notes: The formula for hyperbolic cosine is: $\cosh(x) = \frac{e^x + e^{-x}}{2}$

See Also: [COS](#)
[ACOSH](#)
[SINH](#)
[EXP](#)

SIN (Sine)**Syntax:** `SIN (<arg1>)`**Function Type:** [Point](#)**Description:** Computes the sine of **<arg1>**. Enter **<arg1>** in radians. To convert degrees to radians, use the following formula:

$$\text{degrees} \times \pi/180 = \text{radians}$$

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.**Example:** `SIN (0.5236)` equals 0.5**Notes:** The sine of π is 0.**See Also:** [SINH](#)
[ASIN](#)

SINH (Hyperbolic Sine)

Syntax: `SINH (<arg1>)`

Function Type: [Point](#)

Description: Computes the hyperbolic sine of **<arg1>**. You can use the hyperbolic sine function to approximate a cumulative probability distribution.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

Example: `SINH (1)` equals 1.175201194

`SINH (-1)` equals -1.175201194

Notes: The formula for the hyperbolic sine is: $\sinh(x) = \frac{e^x - e^{-x}}{2}$

See Also: [SIN](#)
[ASINH](#)
[COSH](#)
[EXP](#)



TAN (Tangent)**Syntax:** TAN (<arg1>)**Function Type:** [Point](#)**Description:** Computes the tangent of **<arg1>**. Enter **<arg1>** in radians. To convert degrees to radians, use the following formula:

$$\text{degrees} \times \pi/180 = \text{radians}$$

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.**Example:** TAN (0.7853981634) equals 1.TAN (45 x $\pi/180$) equals 1**See Also:** [ATAN](#)[TANH](#)[SINH](#)[COSH](#)[EXP](#)

TANH (Hyperbolic Tangent)

Syntax: `TANH (<arg1>)`

Function Type: [Point](#)

Description: Computes the hyperbolic tangent of **<arg1>**.

Data Types:

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	COMPLEX	
COLOR	COLOR	
STRING	not supported	

Object Types: All object types, standard rules.

Example: `TANH (-2)` equals -0.96402758

`TANH (0)` equals 0

Notes: The formula for hyperbolic tangent is: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

See Also: [TAN](#)



Zonal

[SUMMARY](#) (*Summary*)

[ZONAL DIVERSITY](#) (*Zonal Diversity*)

[ZONAL MAJORITY](#) (*Zonal Majority*)

[ZONAL MAJORITY COUNT](#) (*Zonal Majority Count*)

[ZONAL MAJORITY FRACTION](#) (*Zonal Majority Fraction*)

[ZONAL MAX](#) (*Zonal Maximum from Summary*)

[ZONAL MAX](#) (*Zonal Maximum from Two Rasters*)

[ZONAL MEAN](#) (*Zonal Mean from Summary*)

[ZONAL MEAN](#) (*Zonal Mean from Two Rasters*)

[ZONAL MEDIAN](#) (*Zonal Median*)

[ZONAL MIN](#) (*Zonal Minimum from Summary*)

[ZONAL MIN](#) (*Zonal Minimum from Two Rasters*)

[ZONAL RANGE](#) (*Zonal Range from Summary*)

[ZONAL RANGE](#) (*Zonal Range from Two Rasters*)

[ZONAL SD](#) (*Zonal Standard Deviation from Summary*)

[ZONAL SD](#) (*Zonal Standard Deviation from Two Rasters*)

[ZONAL STANDARD DEVIATION](#) (*Zonal Standard Deviation from Summary*)

[ZONAL STANDARD DEVIATION](#) (*Zonal Standard Deviation from Two Rasters*)



For more information see [Standard Rules](#).



SUMMARY (Cross Tabulation)

Syntax: SUMMARY (<zone_raster>, <class_raster>)

Function Type: Zonal

Description: Returns a **MATRIX** containing a cross tabulation of the two input rasters. In the returned matrix, position **[i, j]** (i.e., row **i**, column **j**) contains the number of pixels which have value **i** in <zone_raster> and value **j** in <class_raster>. The values in <zone_raster> are referred to as “zones,” and the values in <class_raster> as “classes.”

Data Types: <zone_raster> and <class_raster> must contain only unsigned integer values.

Input	Output	Comments
BINARY	INTEGER	converted to INTEGER
INTEGER	INTEGER	
FLOAT	not supported	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

Object Types: <zone_raster> and <class_raster> are **RASTER**. Output is a **MATRIX** with (**GLOBAL MAX (<zone_raster>) + 1**) rows and (**GLOBAL MAX (<class_raster>) + 1**) columns; the number of zones is the number of rows and the number of classes is the number of columns.



ZONAL DIVERSITY (Zonal Diversity)

Syntax:

```
ZONAL DIVERSITY (<matrix>)
or
ZONAL DIVERSITY (<matrix>, <ignoreoption>)
or
ZONAL DIVERSITY (<matrix>, <ignoreoption> <backgroundvalue>)
```

Function Type:

[Point](#)

Description:

The input **<matrix>** should be the output of the SUMMARY function. This function computes the number of different values in each zone. The rows of **<matrix>** represent the zones, and the columns represent the classes input to the **SUMMARY** function. This function returns a **TABLE** containing the number of different classes contained in each zone.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input classes in the computation, **IGNORE** indicates to ignore a background class. **<backgroundvalue>** specifies the background class to be ignored. If **<backgroundvalue>** is not present, zero is used as the background class.

Data Types:

<matrix> and **<backgroundvalue>** may be any numeric type, and are converted to **INTEGER**. The output is also **INTEGER**:

Input	Output	Comments
BINARY	INTEGER	inputs converted to INTEGER
INTEGER	INTEGER	
FLOAT	INTEGER	inputs converted to INTEGER
COMPLEX	INTEGER	inputs converted to INTEGER
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

The result is **TABLE**. The number of rows in the output **TABLE** is the same as the number of rows in **<matrix>**, which is the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

[SUMMARY](#)
[DIVERSITY](#)
[FOCAL DIVERSITY](#)
[GLOBAL DIVERSITY](#)
[STACK DIVERSITY](#)



ZONAL MAJORITY (Zonal Majority)

Syntax:

```
ZONAL MAJORITY (<matrix>)
or
ZONAL MAJORITY (<matrix>, <ignoreoption>)
or
ZONAL MAJORITY (<matrix>, <ignoreoption> <backgroundvalue>)
```

Function Type:

[Point](#)

Description:

The input **<matrix>** should be the output of the **SUMMARY** function. This function computes the most commonly occurring value in each zone. The rows of **<matrix>** represent the zones, and the columns represent the classes input to the **SUMMARY** function. This function returns a **TABLE** containing the most commonly occurring class value contained in each zone.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input classes in the computation, **IGNORE** indicates to ignore a background class. **<backgroundvalue>** specifies the background class to be ignored. If **<backgroundvalue>** is not present, zero is used as the background class.

Data Types:

<matrix> and **<backgroundvalue>** may be any numeric type, and are converted to **INTEGER**. The output is also **INTEGER**:

Input	Output	Comments
BINARY	INTEGER	inputs converted to INTEGER
INTEGER	INTEGER	
FLOAT	INTEGER	inputs converted to INTEGER
COMPLEX	INTEGER	inputs converted to INTEGER
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

The result is **TABLE**. The number of rows in the output **TABLE** is the same as the number of rows in **<matrix>**, which is the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

[SUMMARY](#)
[MAJORITY](#)
[FOCAL MAJORITY](#)
[GLOBAL MAJORITY](#)
[STACK MAJORITY](#)



ZONAL MAJORITY COUNT (Zonal Majority Count)

Syntax:

```
ZONAL MAJORITY COUNT (<matrix>)
```

or

```
ZONAL MAJORITY COUNT (<matrix>, <ignoreoption>)
```

or

```
ZONAL MAJORITY COUNT (<matrix>, <ignoreoption> <backgroundvalue>)
```

Function Type:

[Point](#)

Description:

The input **<matrix>** should be the output of the **SUMMARY** function. This function computes the number of pixels in the most commonly occurring value in each zone. The rows of **<matrix>** represent the zones, and the columns represent the classes input to the **SUMMARY** function. This function returns a **TABLE** containing the number of pixels in the most commonly occurring class in each zone.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input classes in the computation, **IGNORE** indicates to ignore a background class. **<backgroundvalue>** specifies the background class to be ignored. If **<backgroundvalue>** is not present, zero is used as the background class.

Data Types:

<matrix> and **<backgroundvalue>** may be any numeric type, and are converted to **INTEGER**. The output is also **INTEGER**:

Input	Output	Comments
BINARY	INTEGER	inputs converted to INTEGER
INTEGER	INTEGER	
FLOAT	INTEGER	inputs converted to INTEGER
COMPLEX	INTEGER	inputs converted to INTEGER
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

The result is **TABLE**. The number of rows in the output **TABLE** is the same as the number of rows in **<matrix>**, which is the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

[SUMMARY](#)

[ZONAL MAJORITY](#)

[ZONAL MAJORITY FRACTION](#)



ZONAL MAJORITY FRACTION (Zonal Majority Fraction)

Syntax:

```
ZONAL MAJORITY FRACTION (<matrix>)  
  
or  
  
ZONAL MAJORITY FRACTION (<matrix>, <ignoreoption>)  
  
or  
  
ZONAL MAJORITY FRACTION (<matrix>, <ignoreoption>  
<backgroundvalue>)
```

Function Type:

Point

Description:

The input **<matrix>** should be the output of the **SUMMARY** function. This function computes the fraction of the total zone which overlaps the majority class in each zone. The rows of **<matrix>** represent the zones, and the columns represent the classes input to the **SUMMARY** function. This function returns a **TABLE** containing the fraction of the total zone which overlaps the majority class in each zone.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input classes in the computation, **IGNORE** indicates to ignore a background class. **<backgroundvalue>** specifies the background class to be ignored. If **<backgroundvalue>** is not present, zero is used as the background class.

Data Types:

<matrix> and **<backgroundvalue>** may be any numeric type, and are converted to **INTEGER**. The output is **FLOAT**; output values range from 0.0 to 1.0.

Input	Output	Comments
BINARY	FLOAT	inputs converted to INTEGER
INTEGER	FLOAT	
FLOAT	FLOAT	inputs converted to INTEGER
COMPLEX	FLOAT	inputs converted to INTEGER
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

The result is **TABLE**. The number of rows in the output **TABLE** is the same as the number of rows in **<matrix>**, which is the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

[SUMMARY](#)
[ZONAL MAJORITY](#)
[ZONAL MAJORITY COUNT](#)

ZONAL MAX (Zonal Maximum from Summary)

Syntax:

```
ZONAL MAX (<matrix>)
```

or

```
ZONAL MAX (<matrix>, <ignoreoption>)
```

or

```
ZONAL MAX (<matrix>, <ignoreoption> <backgroundvalue>)
```

Function Type:

Point

Description:

The input **<matrix>** should be the output of the **SUMMARY** function. This function computes the maximum class value in each zone. The rows of **<matrix>** represent the zones, and the columns represent the classes input to the **SUMMARY** function. This function returns a **TABLE** containing the maximum class value in each zone.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input classes in the computation, **IGNORE** indicates to ignore a background class. **<backgroundvalue>** specifies the background class to be ignored. If **<backgroundvalue>** is not present, zero is used as the background class.

Data Types:

<matrix> and **<backgroundvalue>** may be any numeric type, and are converted to **INTEGER**. The output is also **INTEGER**:

Input	Output	Comments
BINARY	INTEGER	inputs converted to INTEGER
INTEGER	INTEGER	
FLOAT	INTEGER	inputs converted to INTEGER
COMPLEX	INTEGER	inputs converted to INTEGER
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

The result is **TABLE**. The number of rows in the output **TABLE** is the same as the number of rows in **<matrix>**, which is the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

[SUMMARY](#)

[MAX](#)

[FOCAL MAX](#)

[GLOBAL MAX](#)

[STACK MAX](#)

[ZONAL MAX](#) (from rasters)



ZONAL MAX (Zonal Maximum from Two Rasters)

Syntax:

```
ZONAL MAX (<zone_raster>, <value_raster>)

or

ZONAL MAX (<zone_raster>, <value_raster>, <ignoreoption>)

or

ZONAL MAX (<zone_raster>, <value_raster>, <ignoreoption>
<backgroundvalue>)
```

Function Type:

Zonal

Description:

This function finds the maximum value in **<value_raster>** which overlays each zone of **<zone_raster>**. This function returns a **TABLE** containing one row per zone. Row **i** of the returned table contains the maximum value from **<value_raster>** of all pixels which have value **i** in **<zone_raster>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value.

Data Types:

<zone_raster> may be any numeric type, and is converted to **INTEGER**. **<backgroundvalue>** may be any numeric type. The output type is the same as the type of **<value_raster>**:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

<zone_raster> and **<value_raster>** are **RASTER**. The result is a **TABLE**. The number of rows in the output **TABLE** is (**GLOBAL MAX (<zone_raster>) + 1**), i.e., the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

[SUMMARY](#) [MAX](#) [FOCAL MAX](#) [GLOBAL MAX](#)
[STACK MAX](#) [ZONAL MAX](#) (from summary)



ZONAL MEAN (Zonal Mean from Summary)

Syntax:

```
ZONAL MEAN (<matrix>)

or

ZONAL MEAN (<matrix>, <ignoreoption>)

or

ZONAL MEAN (<matrix>, <ignoreoption> <backgroundvalue>)
```

Function Type:

Point

Description:

The input **<matrix>** should be the output of the **SUMMARY** function. This function computes the mean class value in each zone. The rows of **<matrix>** represent the zones, and the columns represent the classes input to the **SUMMARY** function. This function returns a **TABLE** containing the mean class value in each zone.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input classes in the computation, **IGNORE** indicates to ignore a background class. **<backgroundvalue>** specifies the background class to be ignored. If **<backgroundvalue>** is not present, zero is used as the background class.

Data Types:

<matrix> and **<backgroundvalue>** may be any numeric type, and are converted to **INTEGER**. The output is **FLOAT**:

Input	Output	Comments
BINARY	FLOAT	inputs converted to INTEGER
INTEGER	FLOAT	
FLOAT	FLOAT	inputs converted to INTEGER
COMPLEX	FLOAT	inputs converted to INTEGER
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

The result is **TABLE**. The number of rows in the output **TABLE** is the same as the number of rows in **<matrix>**, which is the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

[SUMMARY](#)
[MEAN](#)
[FOCAL MEAN](#)
[GLOBAL MEAN](#)
[STACK MEAN](#)
[ZONAL MEAN \(from rasters\)](#)



ZONAL MEAN (Zonal Mean from Two Rasters)

Syntax:

```
ZONAL MEAN (<zone_raster>, <value_raster>)  
  
or  
  
ZONAL MEAN (<zone_raster>, <value_raster>, <ignoreoption>)  
  
or  
  
ZONAL MEAN (<zone_raster>, <value_raster>, <ignoreoption>  
<backgroundvalue>)
```

Function Type:

[Zonal](#)

Description:

This function finds the mean of all values in **<value_raster>** which overlay each zone of **<zone_raster>**. This function returns a **TABLE** containing one row per zone. Row **i** of the returned table contains the mean value from **<value_raster>** of all pixels which have value **i** in **<zone_raster>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value.

Data Types:

<zone_raster> may be any numeric type, and is converted to **INTEGER**. **<backgroundvalue>** may be any numeric type. **<value_raster>** may be **INTEGER** or **FLOAT**. The output type is **FLOAT**.

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

<zone_raster> and **<value_raster>** are **RASTER**. The result is a **TABLE**. The number of rows in the output **TABLE** is (**GLOBAL MAX (<zone_raster>) + 1**), i.e., the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

[SUMMARY](#) [MEAN](#) [FOCAL MEAN](#) [GLOBAL MEAN](#)
[STACK MEAN](#) [ZONAL MEAN](#) (from summary)



ZONAL MEDIAN (Zonal Median)

Syntax:

```
ZONAL MEDIAN (<matrix>)

or

ZONAL MEDIAN (<matrix>, <ignoreoption>)

or

ZONAL MEDIAN (<matrix>, <ignoreoption> <backgroundvalue>)
```

Function Type:

[Point](#)

Description:

The input **<matrix>** should be the output of the **SUMMARY** function. This function computes the statistical median class value in each zone. The rows of **<matrix>** represent the zones, and the columns represent the classes input to the **SUMMARY** function. This function returns a **TABLE** containing the statistical median class value in each zone.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input classes in the computation, **IGNORE** indicates to ignore a background class. **<backgroundvalue>** specifies the background class to be ignored. If **<backgroundvalue>** is not present, zero is used as the background class.

Data Types:

<matrix> and **<backgroundvalue>** may be any numeric type, and are converted to **INTEGER**. The output is also **INTEGER**:

Input	Output	Comments
BINARY	INTEGER	inputs converted to INTEGER
INTEGER	INTEGER	
FLOAT	INTEGER	inputs converted to INTEGER
COMPLEX	INTEGER	inputs converted to INTEGER
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

The result is **TABLE**. The number of rows in the output **TABLE** is the same as the number of rows in **<matrix>**, which is the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

[SUMMARY](#)
[MEDIAN](#)
[FOCAL MEDIAN](#)
[STACK MEDIAN](#)
[GLOBAL MEDIAN](#)



ZONAL MIN (Zonal Minimum from Summary)

Syntax:

```
ZONAL MIN (<matrix>)

or

ZONAL MIN (<matrix>, <ignoreoption>)

or

ZONAL MIN (<matrix>, <ignoreoption> <backgroundvalue>)
```

Function Type:

Point

Description:

The input **<matrix>** should be the output of the **SUMMARY** function. This function computes the minimum class value in each zone. The rows of **<matrix>** represent the zones, and the columns represent the classes input to the **SUMMARY** function. This function returns a **TABLE** containing the minimum class value in each zone.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input classes in the computation, **IGNORE** indicates to ignore a background class. **<backgroundvalue>** specifies the background class to be ignored. If **<backgroundvalue>** is not present, zero is used as the background class.

Data Types:

<matrix> and **<backgroundvalue>** may be any numeric type, and are converted to **INTEGER**. The output is also **INTEGER**:

Input	Output	Comments
BINARY	INTEGER	inputs converted to INTEGER
INTEGER	INTEGER	
FLOAT	INTEGER	inputs converted to INTEGER
COMPLEX	INTEGER	inputs converted to INTEGER
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

The result is **TABLE**. The number of rows in the output **TABLE** is the same as the number of rows in **<matrix>**, which is the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

[SUMMARY](#)

[MIN](#)

[FOCAL MIN](#)

[GLOBAL MIN](#)

[STACK MIN](#)

[ZONAL MIN](#) (from rasters)



ZONAL MIN (Zonal Minimum from Two Rasters)

Syntax:

```
ZONAL MIN (<zonal_raster>, <value_raster>)

or

ZONAL MIN (<zonal_raster>, <value_raster>, <ignoreoption>)

or

ZONAL MIN (<zonal_raster>, <value_raster>, <ignoreoption>
<backgroundvalue>)
```

Function Type:

Zonal

Description:

This function finds the minimum value in **<value_raster>** which overlays each zone of **<zonal_raster>**. This function returns a **TABLE** containing one row per zone. Row **i** of the returned table contains the minimum value from **<value_raster>** of all pixels which have value **i** in **<zonal_raster>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value.

Data Types:

<zonal_raster> may be any numeric type, and is converted to **INTEGER**. **<backgroundvalue>** may be any numeric type. The output type is the same as the type of **<value_raster>**:

Input	Output	Comments
BINARY	BINARY	
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

<zonal_raster> and **<value_raster>** are **RASTER**. The result is a **TABLE**. The number of rows in the output **TABLE** is (**GLOBAL MAX (<zonal_raster>) + 1**), i.e., the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

SUMMARY

MIN

FOCAL MIN

GLOBAL MIN

STACK MIN

ZONAL MIN (from summary)



ZONAL RANGE (Zonal Range from Summary)

Syntax:

```
ZONAL RANGE (<matrix>)
or
ZONAL RANGE (<matrix>, <ignoreoption>)
or
ZONAL RANGE (<matrix>, <ignoreoption> <backgroundvalue>)
```

Function Type:

Point

Description:

The input **<matrix>** should be the output of the **SUMMARY** function. This function computes the range between the minimum and maximum class values in each zone. The rows of **<matrix>** represent the zones, and the columns represent the classes input to the **SUMMARY** function. This function returns a **TABLE** containing the range between the minimum and maximum class values in each zone. Range is computed as:

MIN - MAX + 1

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input classes in the computation, **IGNORE** indicates to ignore a background class. **<backgroundvalue>** specifies the background class to be ignored. If **<backgroundvalue>** is not present, zero is used as the background class.

Data Types:

<matrix> and **<backgroundvalue>** may be any numeric type, and are converted to **INTEGER**. The output is also **INTEGER**:

Input	Output	Comments
BINARY	INTEGER	inputs converted to INTEGER
INTEGER	INTEGER	
FLOAT	INTEGER	inputs converted to INTEGER
COMPLEX	INTEGER	inputs converted to INTEGER
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

The result is **TABLE**. The number of rows in the output **TABLE** is the same as the number of rows in **<matrix>**, which is the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

[SUMMARY](#)

[ZONAL RANGE](#) (from rasters)



ZONAL RANGE (Zonal Range from Two Rasters)

Syntax:

```
ZONAL RANGE (<zone_raster>, <value_raster>)

or

ZONAL RANGE (<zone_raster>, <value_raster>, <ignoreoption>)

or

ZONAL RANGE (<zone_raster>, <value_raster>, <ignoreoption>
<backgroundvalue>)
```

Function Type:

Zonal

Description:

This function finds the range of values in **<value_raster>** which overlay each zone of **<zone_raster>**. This function returns a **TABLE** containing one row per zone. Row **i** of the returned table contains the range of values from **<value_raster>** of all pixels which have value **i** in **<zone_raster>**. If **<value_raster>** is **INTEGER** or **BINARY**, the range is computed as:

MAX - MIN + 1

If **<value_raster>** is **FLOAT**, the range is computed as:

MAX - MIN

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value.

Data Types:

<zone_raster> may be any numeric type, and is converted to **INTEGER**. **<backgroundvalue>** may be any numeric type. The output type depends on the type of **<value_raster>**:

Input	Output	Comments
BINARY	INTEGER	converted to INTEGER
INTEGER	INTEGER	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

<zone_raster> and **<value_raster>** are **RASTER**. The result is a **TABLE**. The number of rows in the output **TABLE** is **(GLOBAL MAX (<zone_raster>) + 1)**, i.e., the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

See Also:

SUMMARY

ZONAL RANGE (from summary)



ZONAL SD (Zonal Standard Deviation from Summary)

Syntax:

```
ZONAL SD (<matrix>)
or
ZONAL SD (<matrix>, <ignoreoption>)
or
ZONAL SD (<matrix>, <ignoreoption> <backgroundvalue>)
```

Function Type:

Point

Description:

The input **<matrix>** should be the output of the **SUMMARY** function. This function computes the standard deviation of the class values in each zone. The rows of **<matrix>** represent the zones, and the columns represent the classes input to the **SUMMARY** function. This function returns a **TABLE** containing the standard deviation of the class values in each zone.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input classes in the computation, **IGNORE** indicates to ignore a background class. **<backgroundvalue>** specifies the background class to be ignored. If **<backgroundvalue>** is not present, zero is used as the background class.

Data Types:

<matrix> and **<backgroundvalue>** may be any numeric type, and are converted to **INTEGER**. The output is **FLOAT**:

Input	Output	Comments
BINARY	FLOAT	inputs converted to INTEGER
INTEGER	FLOAT	
FLOAT	FLOAT	inputs converted to INTEGER
COMPLEX	FLOAT	inputs converted to INTEGER
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

The result is **TABLE**. The number of rows in the output **TABLE** is the same as the number of rows in **<matrix>**, which is the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

Notes:

Equivalent to:

```
ZONAL STANDARD DEVIATION (<matrix>)
ZONAL STANDARD DEVIATION (<matrix>, <ignoreoption>)
ZONAL STANDARD DEVIATION (<matrix>, <ignoreoption>
<backgroundvalue>)
```

See Also:

[SUMMARY](#)

[ZONAL STANDARD DEVIATION](#) (from summary)

[ZONAL SD](#)

[SD](#)

[FOCAL SD](#)

[GLOBAL SD](#)

[STACK SD](#)



ZONAL SD (Zonal Standard Deviation from Two Rasters)

Syntax:

```
ZONAL SD (<zone_raster>, <value_raster>)  
  
or  
  
ZONAL SD (<zone_raster>, <value_raster>, <ignoreoption>)  
  
or  
  
ZONAL SD (<zone_raster>, <value_raster>, <ignoreoption>  
<backgroundvalue>)
```

Function Type:

Zonal

Description:

This function finds the standard deviation of all values in **<value_raster>** which overlay each zone of **<zone_raster>**. This function returns a **TABLE** containing one row per zone. Row **i** of the returned table contains the standard deviation from **<value_raster>** of all pixels which have value **i** in **<zone_raster>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value.

Data Types:

<zone_raster> may be any numeric type, and is converted to **INTEGER**. **<backgroundvalue>** may be any numeric type. **<value_raster>** may be **INTEGER** or **FLOAT**. The output type is **FLOAT**.

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

<zone_raster> and **<value_raster>** are **RASTER**. The result is a **TABLE**. The number of rows in the output **TABLE** is (**GLOBAL MAX (<zone_raster>) + 1**), i.e., the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.



See Also:

[SUMMARY](#)

[ZONAL STANDARD DEVIATION](#) (from summary)

[SD](#)

[FOCAL SD](#)

[GLOBAL SD](#)

[STACK SD](#)



ZONAL STANDARD DEVIATION (from Summary)

Syntax:

```
ZONAL STANDARD DEVIATION (<matrix>)
or
ZONAL STANDARD DEVIATION (<matrix>, <ignoreoption>)
or
ZONAL STANDARD DEVIATION (<matrix>, <ignoreoption>
<backgroundvalue>)
```

Function Type:

Point

Description:

The input **<matrix>** should be the output of the **SUMMARY** function. This function computes the standard deviation of the class values in each zone. The rows of **<matrix>** represent the zones, and the columns represent the classes input to the **SUMMARY** function. This function returns a **TABLE** containing the standard deviation of the class values in each zone.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input classes in the computation, **IGNORE** indicates to ignore a background class. **<backgroundvalue>** specifies the background class to be ignored. If **<backgroundvalue>** is not present, zero is used as the background class.

Data Types:

<matrix> and **<backgroundvalue>** may be any numeric type, and are converted to **INTEGER**. The output is **FLOAT**:

Input	Output	Comments
BINARY	FLOAT	inputs converted to INTEGER
INTEGER	FLOAT	
FLOAT	FLOAT	inputs converted to INTEGER
COMPLEX	FLOAT	inputs converted to INTEGER
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

The result is **TABLE**. The number of rows in the output **TABLE** is the same as the number of rows in **<matrix>**, which is the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.

Notes:

Equivalent to:

```
ZONAL SD (<matrix>)
ZONAL SD (<matrix>, <ignoreoption>)
ZONAL SD (<matrix>, <ignoreoption> <backgroundvalue>)
```



See Also:

[SUMMARY](#)

[ZONAL SD](#) (from rasters)

[ZONAL STANDARD DEVIATION](#) (from rasters)

[SD](#)

[FOCAL SD](#)

[GLOBAL SD](#)

[STACK SD](#)



ZONAL STANDARD DEVIATION (from Two Rasters)

Syntax:

```
ZONAL STANDARD DEVIATION (<zone_raster>, <value_raster>)
```

or

```
ZONAL STANDARD DEVIATION (<zone_raster>, <value_raster>,  
<ignoreoption>)
```

or

```
ZONAL STANDARD DEVIATION (<zone_raster>, <value_raster>,  
<ignoreoption> <backgroundvalue>)
```

Function Type:

[Zonal](#)

Description:

This function finds the standard deviation of all values in **<value_raster>** which overlay each zone of **<zone_raster>**. This function returns a **TABLE** containing one row per zone. Row **i** of the returned table contains the standard deviation from **<value_raster>** of all pixels which have value **i** in **<zone_raster>**.

<ignoreoption> is either **USEALL** or **IGNORE**. **USEALL** indicates to use all input values in the computation, **IGNORE** indicates to ignore a background value. **<backgroundvalue>** specifies the background value to be ignored. If **<backgroundvalue>** is not present, zero is used as the background value.

Data Types:

<zone_raster> may be any numeric type, and is converted to **INTEGER**. **<backgroundvalue>** may be any numeric type. **<value_raster>** may be **INTEGER** or **FLOAT**. The output type is **FLOAT**.

Input	Output	Comments
BINARY	not supported	
INTEGER	FLOAT	
FLOAT	FLOAT	
COMPLEX	not supported	
COLOR	not supported	
STRING	not supported	

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**.

Object Types:

<zone_raster> and **<value_raster>** are **RASTER**. The result is a **TABLE**. The number of rows in the output **TABLE** is (**GLOBAL MAX (<zone_raster>) + 1**), i.e., the number of zones.

<ignoreoption> is one of the keywords **USEALL** or **IGNORE**. **<backgroundvalue>** must be **SCALAR**.



See Also:

[SUMMARY](#)

[ZONAL SD](#) (from summary)

[ZONAL STANDARD DEVIATION](#) (from summary)

[SD](#)

[FOCAL STANDARD DEVIATION](#)

[GLOBAL STANDARD DEVIATION](#)

[STACK STANDARD DEVIATION](#)





Section III

Indexes

Index of Symbols

Symbol	Usage
!	Factorial
!=	Inequality
!~	Case Insensitive String Inequality
&	Bitwise And
&&	Logical And
*	Multiplication
**	Raise to Power
+	Addition
-	Subtraction Negation
/	Division
//	Concatenation
.	Map Raster Through Descriptor Column
::	Read Descriptor Column or Color Table See TABLE Declarations, Descriptors and Color Tables
<	Less Than
<=	Less Than or Equal
=	See Assignment Statements
==	Equality
=~	Case Insensitive String Equality
>	Greater Than
>=	Greater Than or Equal
^	Bitwise Exclusive Or
	Bitwise Or
	Logical Or
~	Bitwise Not

Indexes

Index of Keywords

The following **keywords** are interpreted to have special meaning by the Modeler, and should not be used as variable names:

Keyword	Usage
ABS	Absolute Value
ACOS	Arccosine
ACOSH	Hyperbolic Arccosine
AND	Logical And
ANGLE	Angle
AOI	See SET AOI statement See Raster Declarations, Area Of Interest Specification See Vector Declarations, Area Of Interest Specification
AS	See VIEW statement
ASIN	Arcsine
ASINH	Hyperbolic Arcsine
ASPECT	Aspect
ATAN	Arctangent
ATHEMATIC	See RASTER Declarations, Layer Type Parameters
BILINEAR	See RASTER Declarations, Interpolation Parameters See SET DEFAULT INTERPOLATION statement
BIN	See RASTER Declarations, Bin Function Specification
BINARY	Data Type Specifier Convert to BINARY
BINS	See RASTER Declarations, Bin Function Specification
BIT	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
BOUNDARY	Boundary
C128	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
C64	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
CAT	Concatenate Strings
CATEGORICAL	See RASTER Declarations, Layer Type Parameters
CEIL	Ceiling
CELLAREA	Area of Grid Cells
CELLSIZE	See SET CELLSIZE statement



Keyword	Usage
CELLUNITS	Cell Size Units
CELLX	X Cell Size
CELLY	Y Cell Size
CIRC	Test if Inside Unit Circle
CLUMP	Clump (Contiguity Analysis)
COLOR	Data Type Specifier Create Color Scalar
COLORTABLE	See TABLE Declarations, Descriptors and Color Tables See: (Read Descriptor Column or Color Table)
COMPLEX	Data Type Specifier Convert to COMPLEX
COMPLEX_DOUBLE	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
COMPLEX_SINGLE	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
COMPONENTS	See PRINCIPAL COMPONENTS
CONDITIONAL	Conditional
CONJ	Complex Conjugate
CONTINUOUS	See RASTER Declarations, Layer Type Parameters
CONVOLUTION	See RASTER Declarations, Interpolation Parameters See SET DEFAULT INTERPOLATION statement
CONVOLVE	Convolution
CORRELATION	Correlation from Covariance Matrix or Raster
COS	Cosine
COSH	Hyperbolic Cosine
COVARIANCE	Covariance Matrix
CUBIC	See RASTER Declarations, Interpolation Parameters See SET DEFAULT INTERPOLATION statement
DEBUG	For Internal Use Only
DEFAULT	Binary Constant , Equals 1 or TRUE
DEGREE	See DEGREE SLOPE
DELETE	Reserved for Future Use
DELETE_IF_EXISTING	See RASTER Declarations, Existence Parameters
DELROWS	Delete Rows from Sieved Descriptor Column
DELTA	Delta



Indexes

Keyword	Usage
DENSITY	Local Density See FOCAL DENSITY
DESCRIPTOR	See TABLE Declarations, Descriptors and Color Tables
DEVIATION	See STANDARD DEVIATION See FOCAL STANDARD DEVIATION See GLOBAL STANDARD DEVIATION See STACK STANDARD DEVIATION See ZONAL STANDARD DEVIATION
DIRECT	See RASTER Declarations, Bin Function Specification See DIRECT LOOKUP
DIST	Distance
DIVERSITY	Local Diversity See FOCAL DIVERSITY See GLOBAL DIVERSITY See STACK DIVERSITY See ZONAL DIVERSITY
DOUBLE	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
EDGE	See Raster Declarations, Edge Extension Specification
EIGENMATRIX	Compute Matrix of Eigenvectors
EIGENVALUES	Compute Table of Eigenvalues
EITHER	See EITHER...IF...OR....OTHERWISE
ELSE	See Flow Control, Conditional Branching
EQ	Equality
EVEN	Test if Even
EXP	Exponential
F32	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
F64	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
FALSE	Binary Constant , Equals 0
FILE	See RASTER Declarations, Using Files See RASTER Declarations, Window Specification See SET WINDOW statement
FILL	See Raster Declarations, Edge Extension Specification
FLOAT	Data Type Specifier Convert to FLOAT
FLOAT_DOUBLE	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement

Keyword	Usage
FLOAT_SINGLE	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
FLOOR	Floor
FOCAL	Used in Neighborhood Functions FOCAL MAX, FOCAL SUM, etc.
FOR	Reserved for Future Use
FROM	See RASTER Declarations, Bin Function Specification
GAMMA	Gamma
GE	Greater Than or Equal
GLOBAL	Used in Neighborhood Functions GLOBAL MAX, GLOBAL SUM, etc.
GT	Greater Than
HISTMATCH	Histogram Matching
HISTOEQ	Histogram Equalization
HISTOGRAM	Histogram
HUE	Get Hue from RGB
IF	See EITHER...IF...OR....OTHERWISE See Flow Control, Conditional Branching
IGNORE	See SET DEFAULT STATISTICS statement See RASTER Declarations, Statistics Parameters See GLOBAL functions See COVARIANCE , HISTOEQ , HISTOGRAM , PRINCIPAL COMPONENTS , RASTERMATCH , and STRETCH
IHSTOBLU	Get Blue from Intensity, Hue and Saturation
IHSTOGRN	Get Green from Intensity, Hue and Saturation
IHSTORED	Get Red from Intensity, Hue and Saturation
IHSTORGB	Get Red, Green and Blue from Intensity, Hue and Saturation
IMAG	Imaginary Part
INDEX	Index (Find Matching Item on List)
INPUT	See RASTER Declarations, Access Parameters
INTEGER	Data Type Specifier Convert to INTEGER
INTENS	Get Intensity from RGB
INTERPOLATION	See RASTER Declarations, Interpolation Parameters See SET DEFAULT INTERPOLATION statement
INTERSECTION	See SET WINDOW statement
INV	Multiplicative Inverse



Indexes

Keyword	Usage
ISALLTRUE	Test for All Non-zero
ISNONZERO	Test for Non-zero
LAYERHEIGHT	Height of Raster Layer
LAYERWIDTH	Width of Raster Layer
LE	Less Than or Equal
LENGTH	Length of String
LINEAR	See RASTER Declarations, Bin Function Specification
LINEARCOMB	Linear Combination
LOG	Natural Logarithm See RASTER Declarations, Bin Function Specification
LOG10	Common Logarithm
LOOKUP	Map Input Values Through Lookup Table See DIRECT LOOKUP
LOWERCASE	Lowercase Conversion
LT	Less Than
MAJORITY	Local Majority See FOCAL MAJORITY See GLOBAL MAJORITY See STACK MAJORITY See ZONAL MAJORITY See ZONAL MAJORITY COUNT See ZONAL MAJORITY FRACTION
MAP	See RASTER Declarations, Window Specification See SET WINDOW statement
MAPX	Create Raster Containing X Map Coordinates
MAPY	Create Raster Containing Y Map Coordinates
MATCHES	String Wildcard Match
MATDIV	Matrix Division
MATINV	Matrix Inverse
MATMUL	Matrix Multiplication
MATRIX	Object Type Specifier Create Matrix from List of Scalars See MATRIX SERIES
MATRIXTOTABLE	Convert 1 Column Matrix to Table
MATTRANS	Matrix Transpose

Keyword	Usage
MAX	Local Maximum See FOCAL MAX See GLOBAL MAX See STACK MAX See ZONAL MAX
MEAN	Local Mean See FOCAL MEAN See GLOBAL MEAN See STACK MEAN See ZONAL MEAN
MEDIAN	Local Median See FOCAL MEDIAN See GLOBAL MEDIAN See STACK MEDIAN See ZONAL MEDIAN
MIN	Local Minimum See FOCAL MIN See GLOBAL MIN See STACK MIN See ZONAL MIN
MINORITY	Local Minority See FOCAL MINORITY See GLOBAL MINORITY See STACK MINORITY
MOD	Modulus
NE	Inequality
NEAREST	See RASTER Declarations, Interpolation Parameters See SET DEFAULT INTERPOLATION statement
NEIGHBOR	See RASTER Declarations, Interpolation Parameters See SET DEFAULT INTERPOLATION statement
NEW	See RASTER Declarations, Existence Parameters
NONE	See SET AOI statement See Raster Declarations, Area Of Interest Specification
NOT	Logical NOT
NUMCOLS	Number of Columns
NUMLAYERS	Number of Layers
NUMROWS	Number of Rows
ODD	Test if Odd
OLD	See RASTER Declarations, Existence Parameters
OR	Logical OR See EITHER...IF...OR....OTHERWISE
ORIGIN	See SET DEFAULT ORIGIN statement



Indexes

Keyword	Usage
OTHERWISE	See EITHER...IF...OR....OTHERWISE
OUTPUT	See RASTER Declarations, Access Parameters
PERCENT	See PERCENT SLOPE
PI	Float Constant
PICK	Pick (Get nth Item on List)
PIXEL	See RASTER Declarations, Window Specification See SET WINDOW statement
PIXELX	Create Raster Containing Column Number
PIXELY	Create Raster Containing Row Number
POWER	Raise to Power
PRINCIPAL	See PRINCIPAL COMPONENTS
PRINTTREE	For Internal Use Only
QUIT	Quit Statement
RANDOM	Generate random data.
RANK	Local Rank See FOCAL RANK
RASTER	Object Type Specifier
RASTERMATCH	Raster Matching
READ	READ statement
REAL	Real Part
RECT	Rectangle
REFLECT	See Raster Declarations, Edge Extension Specification
RELIEF	Shaded Relief
RESET	For Internal Use Only
RGBTOIHS	Get Intensity, Hue and Saturation from Red, Green and Blue
ROUND	Round
S16	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
S32	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
S8	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
SATUR	Get Saturation from RGB
SCALAR	Object Type Specifier

Keyword	Usage
SD	Local Standard Deviation See FOCAL SD See GLOBAL SD See STACK SD See ZONAL SD
SEARCH	Search (Proximity Analysis)
SEED	See SET RANDOM SEED
SERIES	See MATRIX SERIES See TABLE SERIES
SET	See Setting Windows See Other SET statements
SHOW	SHOW statement
SIEVETABLE	Get Sieve Lookup Table
SIGN	Sign
SIGNED	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
SIGNED_16_BIT	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
SIGNED_32_BIT	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
SIGNED_8_BIT	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
SIN	Sine
SINC	Sinc
SINGLE	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
SINH	Hyperbolic Sine
SLOPE	See DEGREE SLOPE See PERCENT SLOPE
SQRT	Square Root
STACK	Convert FLOAT TABLE to COLOR SCALAR
STACKLAYERS	Stack Raster Layers
STANDARD	See STANDARD DEVIATION See FOCAL STANDARD DEVIATION See GLOBAL STANDARD DEVIATION See STACK STANDARD DEVIATION See ZONAL STANDARD DEVIATION
STATISTICS	See SET DEFAULT STATISTICS statement
STEP	Step



Indexes

Keyword	Usage
STRETCH	Stretch raster data
STRING	Data Type Specifier
SUM	Local Sum See FOCAL SUM See GLOBAL SUM See STACK SUM
SUMMARY	Summary - Cross Tabulation
TABLE	Object Type Specifier Create Table from List of Scalars See TABLE SERIES
TABLETOMATRIX	Convert Table to 1 Column Matrix
TAN	Tangent
TANH	Hyperbolic Tangent
THEMATIC	See RASTER Declarations, Layer Type Parameters
TILESIZE	See SET TILESIZE statement
TO	See RASTER Declarations, Bin Function Specification
TRI	Triangle
TRUE	Binary Constant , Equals 1
TRUNC	Truncate
U1	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
U16	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
U2	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
U32	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
U4	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
U8	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
UNION	See SET WINDOW statement
UNLESS	See Flow Control, Conditional Branching
UNSIGNED	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
UNSIGNED_1_BIT	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement

Keyword	Usage
UNSIGNED_16_BIT	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
UNSIGNED_2_BIT	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
UNSIGNED_32_BIT	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
UNSIGNED_4_BIT	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
UNSIGNED_8_BIT	See RASTER Declarations, Data Type Parameters See SET DEFAULT <datatype> statement
UNSTACK	Convert COLOR SCALAR to FLOAT TABLE
UNTIL	See Flow Control, Looping
UPPERCASE	Uppercase Conversion
USEALL	See SET DEFAULT STATISTICS statement See RASTER Declarations, Statistics Parameters See GLOBAL functions See COVARIANCE , HISTOEQ , HISTOGRAM , PRINCIPAL COMPONENTS , RASTERMATCH , and STRETCH
USEFILE	Reserved for future use
USING	See VIEW statement
VIEW	Obsolete. No longer supported
WHILE	See Flow Control, Looping
WHOLE	Test if Whole Number
WINDOW	See SET WINDOW statement See RASTER Declarations, Window Specification See VECTOR Declarations, Window Specification
WRITE	WRITE statement
ZONAL RANGE	Zonal Range from Summary or from two Rasters



