



MACRO Reference Guide

Version 9.1

December 2020

SYNOPSYS®

Copyright Notice and Proprietary Information

Copyright © 2020 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

“This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____.”

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys' company and certain product names are trademarks of Synopsys, as set forth at: <https://www.synopsys.com/company/legal/trademarks-brands.html>. All other product or company names may be trademarks of their respective owners.

Contents

Chapter 1	Introduction.....	1
	Getting to Know the User Interface for the MACRO Feature.....	2
	LightTools MACRO Program File Names.....	2
	Running a LightTools MACRO Program.....	2
	MACRO Program Initialization File.....	3
	MACRO Program Limitations.....	3
	MACRO Program Execution.....	3
	MACRO Program Display Parameters.....	4
Chapter 2	The LightTools MACRO BASIC Interpreter	5
	Name Conventions.....	5
	Program Limitations.....	5
	Expressions and Operators.....	6
	Commands and Functions.....	7
	Interactive Input/Output Commands.....	7
	File Input/Output Commands.....	7
	Program Control Commands.....	7
	Miscellaneous Commands.....	8
	Mathematical Functions.....	8
	String Functions.....	9
	File Input/Output Functions.....	10
	Miscellaneous Functions.....	10
Chapter 3	LightTools Extensions to the BASIC Interpreter	11
	Data Access Functions.....	11
	Macro Data Access String Help.....	12
	Functions to Create a List - LTDBLIST\$, LTSELECTLIST\$.....	13
	Functions to Access Objects in a List.....	13
	Function to Verify Compatibility of an Object Key and Filter - LTDBTYPE.....	14
	Functions to Access Data Values - LTDBGET.....	14
	Functions to Set Data Values - LTDBSET.....	15
	Function to Delete a List - LTLISTDELETE.....	15
	Functions to Analyze Object Keys.....	15
	Data Access Filters.....	16
	General Filters.....	17
	Filter Index.....	19
	Data Access Functions to Analyze Object Keys.....	20
	Function to List Valid Data Elements for an Object Key - LTDBKEYDUMP.....	20
	Function to Translate an Object Key - LTDBKEYSTR\$.....	20
	LightTools MACRO Commands.....	21
	Command to Execute LightTools Commands Within a LightTools MACRO Program - LTCMD.....	21
	Command to Set LightTools MACRO Program Control Parameters - LTBSET.....	22
	LightTools MACRO Functions.....	23
	Functions to Return a Formatted String - LTSTR\$, LTCOORD2\$, and LTCOORD3\$.....	23
	Function to Return the Status of a LightTools Command - LTGETSTAT.....	23
	Function to Evaluate an Expression - LTEVAL.....	24
	Functions to Set and Acquire Values of LightTools Variables - LTSETVAR and LTGETVAR.....	24
	Function to Return the Type of a LightTools Variable - LTCHECKVAR.....	25
	Function to Return LightTools Version Information - LTVERSION\$.....	25

LightTools and LightTools MACRO Variables	26
Data Access Strings	28
Data Access String Syntax.....	28
Data Access Filter and Index	28
Data Element	28
Data Access String Uses	28
Data Access Strings versus Data Access Lists	29
Window Commands.....	31
 Chapter 4 Command/Function Reference	33
Conventions	34
ABS.....	36
ACOS.....	37
ASC.....	38
ASIN.....	39
ATAN2	40
ATN	41
CALL	42
CEIL.....	43
CHR\$	44
CLOSE #.....	45
COS.....	46
DATA	47
DATES\$.....	48
DEG	49
DIM.....	50
DO.....	52
DO WHILE	53
DOC	54
ECHO.....	55
END	56
END FUNCTION	57
END IF.....	58
END SUB.....	59
EOF	60
ERL	61
ERR.....	62
ERROR	63
EXIT DO.....	64
EXIT FOR.....	65
EXP	66
EXP10	67
FLOOR	68
FOR.....	69
FUNCTION	71
GOSUB/GO SUB	72
GOTO/GO TO	74
IF THEN	76
INPUT.....	78
INPUT #.....	79
INSTR.....	80
INT	82

LEFT\$	83
LEN	84
LET	85
LINE INPUT	86
LINE INPUT #	87
LOG	88
LOG10	89
LOOP	90
LOOP UNTIL	92
LTBSET	93
LTBSET	95
LTCHECKVAR	97
LTCMD	98
LTCMD	99
LTCOORD2\$	100
LTCOORD3\$	101
LTDBGET	102
LTDBGET\$	103
LTDBGETI	104
LTDBGETIS	105
LTDBGETIJ	106
LTDBGETIJ\$	109
LTDBGETSURFDATA	110
LTDBGETSURFVEC	111
LTDBKEYDUMP	112
LTDBKEYSTR\$	113
LTDBLIST\$	114
LTDBQUICKRAYAIM	115
LTDBQUICKRAYQUERY	116
LTDBSET	117
LTDBSETI	118
LTDBSETSURFVEC	119
LTDBTYPE	120
LTEVAL	121
LTGETLASTMSG\$	122
LTGETPHOTOPICFUNCTION	123
LTGETSCOTOPICFUNCTION	124
LTGETSTAT	125
LTGETVAR	126
LTGETVAR\$	127
TLISTATPOS\$	128
TLISTBYNAME\$	129
TLISTDELETE	130
TLISTGETPOS	131
TLISTLAST\$	132
TLISTNEXT\$	133
TLISTSETPOS	134
TLISTSIZE	135
LTSELECTLIST\$	136
LTSETVAR	137
LTSTR\$	138
LTSURFSPLINEPATCH	139

LTSURFSPLINESWEEP	140
LTVERSION\$	141
MAX	142
MERGE.....	143
MID\$.....	145
MIN.....	147
NEXT	148
ON ERROR GOSUB/ON ERROR GO SUB	149
ON GOSUB/ON GO SUB.....	151
ON GOTO/ON GO TO.....	153
OPEN FOR APPEND AS #.....	155
OPEN FOR INPUT AS #	157
OPEN FOR OUTPUT AS #	158
POW	159
PRINT	160
PRINT USING.....	161
PRINT #	163
PRINT # USING.....	165
RAD	167
RANDOMIZE.....	168
READ.....	169
REM	170
RESTORE.....	172
RETURN.....	173
RIGHT\$	174
RND	175
SGN.....	176
SIN	177
SPACE\$	178
SQR.....	179
STR\$	180
STRING\$	181
SUB.....	182
SWAP	183
SYSTEM.....	184
TAN	185
TIMES\$.....	186
VAL	187
WEND.....	188
WHILE.....	189
WIDTH #	190
WRITE	192
WRITE #.....	193
 Chapter 5 MACRO by Example	 195
STEP 1: Create a Mirror Using LTCMD.....	195
STEP 2: Modify the Mirror.....	196
STEP 3: Modify the Mirror Using an Input Variable	198
STEP 4: Modify the Mirror Using Data Access Filters.....	198
STEP 5: Trace Rays and Output Ray Data	200
STEP 6: Modify the Shape of the Mirror.....	202
STEP 7: Run an Illumination Simulation	204

STEP 8: Move the Source and Rerun the Illumination Simulation	205
STEP 9: View the Illumination Simulation Data and Export to a File	207
Appendix A Specific Data Access Filters.....	209
Appendix B General Data Access Filters	221

Contents

Chapter 1 Introduction

The LightTools MACRO feature is integrated with the LightTools Core Module and is compatible with all other LightTools modules. It allows the user to write programs which are executed from within LightTools. It combines the functionality of a BASIC interpreter with the LightTools command interpreter. Provision is made for executing all LightTools commands as well as accessing and modifying all LightTools data from within the programs.

LightTools MACRO integrates a BASIC interpreter which provides commands for performing such programming tasks as interactive input/output, file input/output, and program control. It also provides functions for performing mathematical and data string operations. A number of mathematical and logical operators are also provided for the construction of executable expressions. The LightTools MACRO BASIC Interpreter is described in Chapter 2, *The LightTools MACRO BASIC Interpreter*.

LightTools MACRO includes extensions to the BASIC interpreter for access to LightTools data and the construction and input of LightTools commands. Access to LightTools data is performed via the LightTools MACRO data access functions. These functions allow the user to create and manipulate lists of LightTools data objects. For example, a macro could create a list of all solid objects, find the last object created, and change its surfaces from refracting to reflecting. Macros can also include LightTools commands. This allows any user input to LightTools via the mouse and keyboard to also be performed via a macro.

The combination of a BASIC interpreter and access to LightTools data allows users to write macros that extend the existing analysis capability in LightTools as well as facilitate model building. For example, a user can write macros that perform the following types of functions:

- Creation of objects that involves repetitive tasks, such as building faceted reflectors or any polyline surface type, helical filaments, etc.
- Compute parameters that are not directly output by LightTools, such as the paraxial focal length of a lens, the major and minor ellipse axes of a conic surface, etc.
- Perform post processing of existing analyses, such as computing the difference between two illumination runs.
- Automation of the design process. For example, the results of one simulation can be used to determine the changes needed for the next simulation.
- Documentation of designs. Users can tailor LightTools output of important parameters and format as needed.

Getting to Know the User Interface for the MACRO Feature

This section describes the LightTools user interface for the LightTools MACRO feature. Included are data input, database and graphic update commands, and any user related interaction processes.

LightTools MACRO Program File Names

LightTools MACRO program file names utilize the LightTools standard naming convention. The extension .ltb is used to identify a LightTools MACRO program file as follows:

filename.n.ltb

where

n is the version number.

Running a LightTools MACRO Program

To run a LightTools MACRO program, the name of the file can be entered at any command prompt. The LightTools MACRO filename cannot be a LightTools command, alias, or script since these take precedence over LightTools MACRO files in order of execution.

LightTools searches for the LightTools MACRO program file name using the same search path used for script files. This search path is specified using the environment variable ORA_SCRIPT_PATH.

A LightTools MACRO program is executed using the Run command or the **Run Macro** menu option. The **Run Macro** option is found under the **Tools** menu.

The Run command or menu option displays the Open dialog box, allowing you to choose a program file from those available in a directory, as shown in Figure 1.

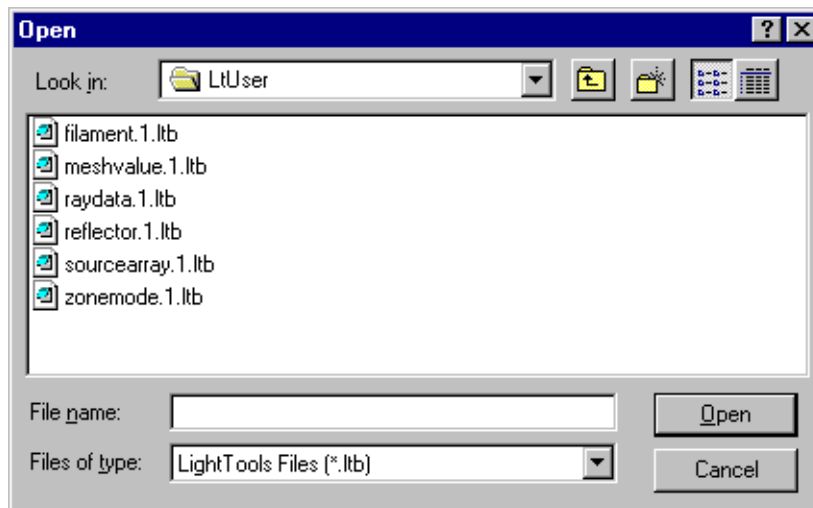


Figure 1. The Open File dialog box for selecting macro program files

MACRO Program Initialization File

A special LightTools MACRO program file named `profile.n.ltb`, if it exists, is read prior to reading the designated LightTools MACRO program file. Typical usage for this special LightTools MACRO program includes collecting a variety of common utility subprograms which may be shared by several LightTools MACRO programs.

MACRO Program Limitations

No parameters can be passed to the LightTools MACRO program when it is invoked. Interactive input or input from a data file is provided for by using the `INPUT` and `LINE INPUT` commands within the LightTools MACRO program (See Chapter 4, *Command/Function Reference*).

A LightTools MACRO program cannot execute another LightTools MACRO program.

MACRO Program Execution

During execution of a MACRO, all other user input to LightTools is denied. The `INPUT` and `LINE INPUT` commands can be used to display a MACRO Input dialog box, shown in Figure 2. The user may enter one or more comma delimited values or a string value in the one line data entry portion of the dialog box.

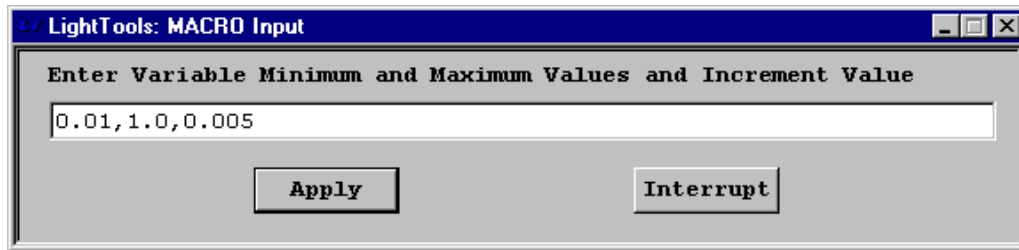


Figure 2. The MACRO Input dialog box

The user may interrupt the LightTools MACRO program execution by selecting Interrupt in the Interrupt Operation dialog box, shown in Figure 3.



Figure 3. The Interrupt Operation dialog box

When a LightTools MACRO program is invoked, only its name is written to the recover file. Therefore, to replay a session which invokes a LightTools MACRO program, the LightTools MACRO program must accompany the recover file.

Errors encountered while processing the LightTools MACRO program are reported to the user via an error message written to the Console window.

MACRO Program Display Parameters

LightTools MACRO programs can use the LTBSET command to control certain display parameters. The LTBSET command used with the UPDATE keyword controls the LightTools database and graphic update mode. The graphics in the 2D Design, 3D Design, and Imaging Path views can be automatically updated whenever modifications are made to the LightTools database. Database and graphic updating slows program execution; therefore, it is recommended to turn the update mode off until needed. A value of 1 turns the automatic update mode on. A value of 0 turns the automatic update mode off. The update mode status is retained upon completion of a LightTools macro program. An example of this command is shown below.

Turn the automatic update mode on:

```
LTBSET UPDATE 1
```

The LTBSET command used with the DIALOG keyword controls the display of LightTools dialog boxes. A value of 1 turns the dialog box display mode on. A value of 0 turns the dialog box display mode off. An example of this command is shown below.

Turn the dialog box display mode off:

```
LTBSET DIALOG 0
```

If the LightTools MACRO program has suppressed LightTools dialog box display during execution with the LTBSET DIALOG command, it is automatically restored when the LightTools MACRO program is complete. The LTBSET DIALOG command does not apply to the LightTools MACRO Input dialog boxes or the Interrupt Operation dialog box.

For more details on these and other parameters that can be controlled with the LTBSET command, see Chapter 4, *Command/Function Reference*.

Chapter 2 The LightTools MACRO BASIC Interpreter

This chapter describes the subset of the ANSI Full BASIC Programming Language Specification that is implemented as part of the LightTools MACRO feature.

Name Conventions

Function and command names are not case sensitive. Variable names are case sensitive. Variable names must begin with an alpha character, and may contain zero or more alphanumeric characters. All variables are considered numerical variables unless the last character of the variable name is the dollar sign character (\$), in which case the variable is a string variable.

Program Limitations

The maximum number of parameters that may be passed to a function or subprogram is 100.

The maximum nesting depth is 64 levels. This is a cumulative total and one level is required for each of the following: loops (do, for, while), if statements, and subprogram and function calls.

The maximum length of a line in a LightTools MACRO program file is 255 characters. LightTools MACRO does not support the use of the ampersand character (&) for continuation of a line as in some BASIC interpreters.

Expressions and Operators

Expressions are constructed with one or more operators and one or more operands. Parentheses may be used to control or override the default precedence for evaluating an expression with multiple operators. Table 1 lists all operators and indicates their precedence level. A level of 1 indicates a higher precedence than a level of 2, etc. All operators except the unary minus require two operands.

Operator	Precedence level	Description
-	1	Unary minus
^	2	Exponentiation
*	3	Multiplication
/	3	Division
MOD	4	Modulus
+	5	Addition
-	5	Subtraction
=	6	Equality and assignment
<>	7	Inequality
<	7	Less than
>	7	Greater than
<=	8	Less than or equal to
>=	8	Greater than or equal to
AND	9	Conjunction
OR	9	Disjunction
NOT	9	Negation
XOR	9	Exclusive or

Table 1. LightTools MACRO operators and their precedence level

Commands and Functions

The following subsections list the subset of ANSI Full BASIC commands and functions that are provided by LightTools MACRO. For full details on usage and syntax, see Chapter 4, *Command/Function Reference*.

Interactive Input/Output Commands

A LightTools MACRO program may query the user for input while executing. This is accomplished by displaying a dialog box, in which the user may enter data. Both INPUT and LINE INPUT may be used for this purpose. Textual and numerical output may be generated and displayed in the LightTools Console window using the PRINT command.

INPUT	Requests data from the user via a LightTools dialog box.
LINE INPUT	Requests data from the user via a LightTools dialog box.
PRINT [USING]	Displays output in the LightTools Console window and writes output to the LightTools log file.

File Input/Output Commands

Sequential file input/output is supported with LightTools MACRO. Files are designated with device numbers. A valid device number specification is a number preceded by the pound character (#). Valid device numbers are 0 through 15. Therefore a maximum of 16 files may be opened at any one time.

CLOSE #	Closes a file.
DOC	Writes the list of LightTools MACRO commands and functions to a file.
INPUT #	Reads data from a file and assigns data to a list of variables.
LINE INPUT #	Reads an entire line from a file and assigns it to a string variable.
OPEN FOR INPUT OUTPUT APPEND AS #	Opens a file for reading (INPUT), writing (OUTPUT), or writing (APPEND) as an appendage.
PRINT # [USING]	Prints output to a file
WIDTH #	Designates the width of a file.
WRITE #	Writes data to a file.

Program Control Commands

CALL	Invokes a subprogram.
DATA	Stores numerical and string constants, accessed by the READ command.
DO [WHILE]	Initiates a DO loop.
ECHO	Enables or disables the display of executed program statements in the LightTools Console window.
END [FUNCTION IF SUB]	Terminates a program, subprogram, function, or an IF sequence.

CALL	Invokes a subprogram.
EXIT DO FOR	Exits a DO or FOR loop.
FOR	Initiates a FOR loop.
FUNCTION	Defines a function.
GOSUB	Directs the program to a subprogram
GOTO	Directs the program to a line.
IF THEN [ELSE ELSEIF]	Initiates and directs an IF sequence.
LET	Assigns a value to a variable.
LOOP [UNTIL]	Terminates a DO loop.
MERGE	Appends a program file to the program currently being executed.
NEXT	Terminates a FOR loop.
ON ERROR GOSUB	Directs a program to a subprogram when an error occurs.
ON GOSUB GOTO	Directs a program to a subprogram or line based on the value of a variable.
READ	Reads values from DATA statements.
RESTORE	Resets the line and position counters for DATA and READ statements.
RETURN	Terminates a subprogram called by GOSUB and directs the program back to the branch point.
SUB	Defines a subprogram.
SYSTEM	Allows execution of a system command during macro program execution.
WEND	Terminates a WHILE loop.
WHILE	Initiates a WHILE loop.

Miscellaneous Commands

DIM	Dimensions variables as arrays or matrices.
ERROR	Simulates an error (See ERR and ERL functions).
RANDOMIZE	Reseeds the random number generator (See RND function).
REM	Designates a line as a comment.
SWAP	Exchanges the values of two variables.

Mathematical Functions

ABS	Returns the absolute value of a number.
ACOS	Returns the arccosine value of a number.
ASIN	Returns the arcsine value of a number.
ATAN2	Returns the arctangent value of the ratio of two numbers.
ATN	Returns the arctangent value of a number.
CEIL	Returns the smallest whole number not less than a number.

COS	Returns the cosine value of a number.
DEG	Returns the number of degrees in a number of radians.
EXP	Returns the exponential value of a number.
EXP10	Returns the value of 10 raised to a power designated by a number.
FLOOR	Returns the largest whole number not greater than a number.
INT	Returns the largest whole number less than or equal to a number.
LOG	Returns the natural logarithm value of a number.
LOG10	Returns the common logarithm value of a number.
MAX	Returns the higher value of two numbers.
MIN	Returns the lower value of two numbers.
POW	Returns the value of a number raised to a power designated by a second number.
RAD	Returns the number of radians in a number of degrees.
RND	Returns a random number.
SGN	Returns the mathematical sign of a number.
SIN	Returns the sine value of a number.
SQR	Returns the square root value of a number.
TAN	Returns the tangent value of a number.

String Functions

ASC	Returns the ASCII code number for the first character of a string.
CHR\$	Returns a one character string corresponding to an ASCII code number.
DATE\$	Returns the current date.
INSTR	Returns the position at which a substring begins within a string.
LEFT\$	Returns a substring with a designated length composed of the characters beginning at the left side of a string.
LEN	Returns the length of a string.
MID\$	Returns a substring of a string beginning at a starting position in the string, and continuing for a designated length.
RIGHT\$	Returns a substring with a designated length composed of the characters at the right side of a string.
SPACE\$	Returns a string of blank spaces with a designated length.
STR\$	Returns a string which is the decimal representation of a number.
STRING\$	Returns a string with a designated length which is composed of characters which are the first character of a string.
TIME\$	Returns the current time.
VAL	Returns the numerical value of the leading numerical characters of a string.

File Input/Output Functions

EOF Returns the positional status of an open file with respect to the end of the file.

Miscellaneous Functions

ERL Returns the line number for the most recent error.

ERR Returns the error number of the most recent error.

Chapter 3 LightTools Extensions to the BASIC Interpreter

This section lists extensions to the BASIC interpreter described in Chapter 2. These extensions allow access to data within the LightTools database, provide input and output functions needed to exchange data with LightTools, and enable you to construct and execute LightTools commands. This chapter summarizes these functions and commands and describes how they are used. To see a complete description of each individual function and command, including syntax and examples, see Chapter 4.

Data Access Functions

Access to LightTools data is performed via the LightTools MACRO data access functions. These functions allow you to create a list of LightTools objects, loop through the list, and retrieve and modify data values associated with those objects. You define the contents of a list using filters that allow you to access LightTools objects by type. A wide range of filters is available, enabling you to produce a long, general list or a short, specific list of objects. A hierarchy of filters exists, in which the higher level filters are used to create more general lists and the lower level filters more specific lists. For example, the higher level filter SOLIDS would be used to create a list of all solid objects in your LightTools model, such as lens primitives, cube primitives, sphere primitives, etc. Whereas the lower level filter ASPHERE_SURFACE would be used to create a list of only aspheric surfaces on one or more lens primitives in the model.

The list of types used in the various data access functions are:

listkey\$	A specific list of objects created by the LTDBLIST\$ function (e.g., list of all lens primitives).
filter\$	A filter used to qualify the objects that are contained in the list (e.g., LENS_PRIMITIVE).
objectkey\$	One of the objects in the list (e.g., the first lens in the list). An object key can be a string defined by a previous list access function (see below), or a filter with an index qualifier (e.g., LENS_PRIMITIVE[1]).
data_element\$	A property of an object that is being accessed (e.g., the diameter of the first lens). Data elements associated with each filter are listed in Appendix C.
data_value	A value of data being accessed or changed (e.g, the diameter value itself).
data_array_2D	A 2-dimensional array containing the YZ coordinates of points.
data_array_3D	A 3-dimensional array containing the XYZ coordinates of points.
index	Value of index number used with a data element to access data in one and two dimensional arrays.

Filters and the hierarchy of filters are described in more detail in *Data Access Filters* on page 16. To illustrate the data access functions, the following filters are used:

COMPONENTS	High level filter containing all entities, ray traceable objects, solid objects, cemented interfaces, polylines, annotation text, groups, and dummy surfaces.
LENS_PRIMITIVE	Subclass of SOLID filter containing only lens primitives.

To illustrate the data access functions, the following data element are used:

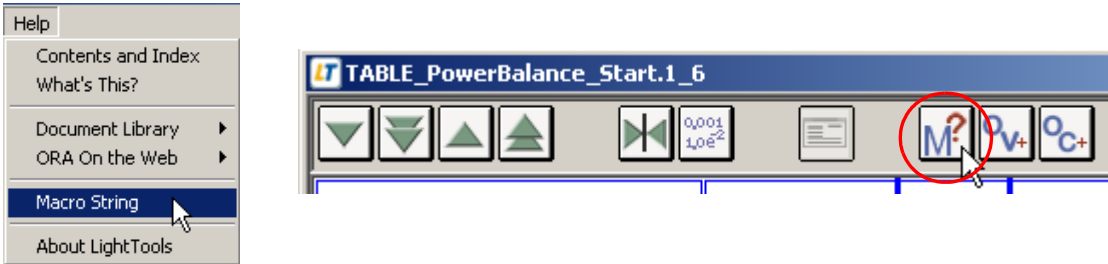
DIAMETER	The physical diameter of a lens.
----------	----------------------------------

Like all BASIC functions, the data access functions return a value. In some cases, the value is a status (i.e., an integer value indicating whether or not the function was performed successfully), while other functions return a list key or object key. Therefore, the functions must be assigned to a variable unless they are used as arguments or in expressions.

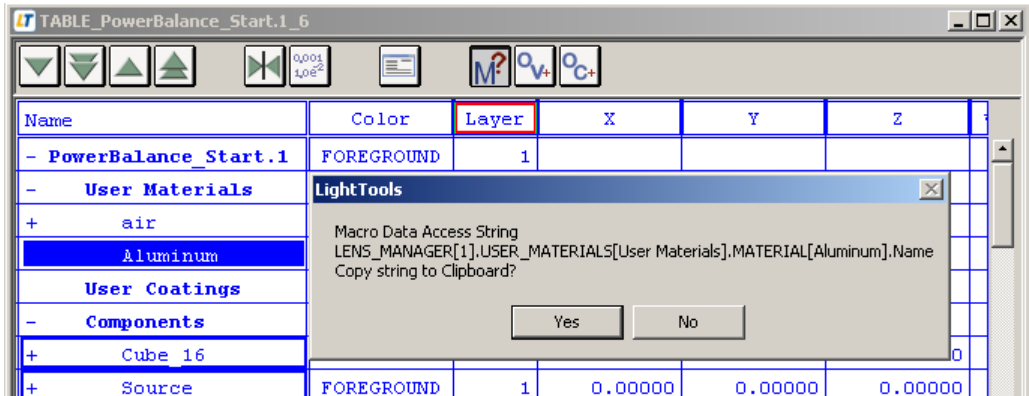
For details on all of the data access functions and their syntax and usage, see Chapter 4.

Macro Data Access String Help

You can view and copy the data access string for MACRO data items in the LightTools Table view. This macro string Help is available from the Table view toolbar and the Help menu, shown below.



First, select the cell of interest in the Table view and then click either the menu or toolbar button to display the data access string dialog box, shown below. To copy the string, click Copy to Clipboard. Click Dismiss to close the window.



Functions to Create a List - LTDBLIST\$, LTSELECTLIST\$

The LTDBLIST\$ function is used to create a list of objects that correspond to an object key and filter. The list is compiled from all objects in the LightTools database. Its syntax is:

```
LTDBLIST$(objectkey$, filter$)
```

For example, to create a list of all lens primitives in your model:

```
LensList$ = LTDBLIST$("COMPONENTS[1]", "LENS_PRIMITIVE")
```

In this example, the object key is the filter COMPONENTS with the index [1], the filter to define which specific components to include in the list is LENS_PRIMITIVE (this way cubes, spheres, and other non-lens primitives, are not included in the list), and LensList\$ is a list key that points to the list that is created.

The LTSELECTLIST\$ function is used to create a list of objects that correspond to a filter. The list is compiled from all objects in the currently active view window. Its syntax is:

```
LTSELECTLIST$(filter$)
```

For example, to create a list of all entities in your model:

```
EntityList$ = LTSELECTLIST$("ENTITY")
```

In this example, the filter is ENTITY, and EntityList\$ is a list key that points to the list that is created.

The lists created with the LTDBLIST\$ and LTSELECTLIST\$ functions are indexed starting at a position of 1 and continuing sequentially to the position of the last object key in the list. Therefore, the first object key is at position 1, the second object key is at position 2, and so on.

Functions to Access Objects in a List

Once a list has been created, object keys can be selected for specific objects in the list. This is accomplished via the list access functions:

LTLISTATPOS\$(listkey\$, n)	Returns the object key at the nth position in the list.
LTLISTNEXT\$(listkey\$)	Returns the object key at the current position in the list and sets the list to the next position.
LTLISTLAST\$(listkey\$)	Returns the object key at the last position in the list.

Following the LTDBLIST\$ example above:

```
LensList$ = LTDBLIST$("COMPONENTS[1]", "LENS_PRIMITIVE")
LensPrim1Key$ = LTLISTNEXT$(LensList$)
```

When a list is created, and its current position has not been modified, the LTLISTNEXT\$ function returns the object key at position 1 in the list. Therefore, LensPrim1Key\$ now corresponds to the first lens primitive in the list LensList\$. Subsequent LTLISTNEXT\$ functions return the object keys at positions 2, 3, 4, and so on. When the object key at the last position in the list has been returned by the LTLISTNEXT\$ function, subsequent LTLISTNEXT\$ functions return a null string as an object key.

Other data access functions that help users find a specific object in a list include:

LTLISTBYNAMES	Returns the object key that corresponds to an object's name.
LTLISTSETPOS	Sets the position in the list for the next LTLISTNEXT\$ function.
LTLISTGETPOS	Returns the current position in the list.
LTLISTSIZE	Returns the total number of objects in the list.

For more details on the use of these functions, please refer to Chapter 4.

Function to Verify Compatibility of an Object Key and Filter - LTDBTYPE

Once a list of object keys has been created, a test can be performed to determine if a specific data access filter is valid for the object corresponding to a specific object key. The test is done using the LTDBTYPE function. The LTDBTYPE function returns a value of 1 if the filter is valid, and a value of 0 if it is invalid. This function can be used as a logical expression in MACRO program statements. The returned value of 1 is equivalent to true, and the returned value of 0 is equivalent to false.

Following the examples above:

```
LensList$ = LTDBLIST$ ("COMPONENTS[1]", "LENS_PRIMITIVE")
LensPrim1Key$ = LTLISTNEXT$(LensList$)
FilterStatus = LTDBTYPE(LensPrim1Key$, "CIRC_LENS_PRIMITIVE")
```

In this case, the value of FilterStatus is set to 1 if the first object key in LensList\$ corresponds to a circular lens primitive. Otherwise, it is set to 0.

Functions to Access Data Values - LTDBGET

Once an object key has been defined that points to an object in the list, data associated with that object can be accessed via the LTDBGET functions:

LTDBGET(objectkey\$, data_element\$)	Returns the value of a data element for the object pointed to by object key.
--------------------------------------	--

Following the examples above:

```
LensList$ = LTDBLIST$ ("COMPONENTS[1]", "LENS_PRIMITIVE")
LensPrim1Key$ = LTLISTNEXT$(LensList$)
lp1_diameter = LTDBGET(LensPrim1Key$, "DIAMETER")
```

The variable lp1_diameter is now set equal to the diameter of the first lens primitive.

The LTDBGET functions vary depending on the type of data being accessed. For example, LTDBGET\$ would be used if the data value being accessed is a string, while LTDBGET would be used if the data value were numeric. Variations also exist for accessing string and numeric data in one and two dimensional arrays. For descriptions of all LTDBGET functions, please refer to Chapter 4.

Functions to Set Data Values - LTDBSET

Once an object key has been defined that points to an object in the list, data associated with that object can be modified via the LTDBSET functions:

LTDBSET(objectkey\$, data_element\$, data_value) Sets the value of a data element to a data value for the object pointed to by the object key.

Following the examples above:

```
LensList$ = LTDBLIST$("COMPONENTS[1]", "LENS_PRIMITIVE")
LensPrimlKey$ = LTLISTNEXT$(LensList$)
DiameterStatus = LTDBSET(LensPrimlKey$, "DIAMETER", 1.5)
```

The diameter of the first lens primitive is now set equal to 1.5. The assignment of data values made by the LTDBSET function returns an integer status value which is indicative of the validity of the assignment. A user-defined status variable, DiameterStatus, is set equal to the returned status value. The value of DiameterStatus would be 0 if the assignment is valid, or greater than zero if the assignment produced an error (e.g., the user tried to change the diameter of the lens to a negative number).

The LTDBSET functions vary depending on the dimensions of the data being modified. Variations also exist for modifying data in one and two dimensional arrays. For descriptions of all LTDBSET functions, please refer to Chapter 4.

Function to Delete a List - LTLISTDELETE

To make the most efficient use of system resources, it is good programming practice to delete lists when they are no longer being used. This is accomplished via the LTLISTDELETE function:

LTLISTDELETE(listkey\$) Deletes the list pointed to by the list key.

Following the examples above:

```
LensList$ = LTDBLIST$("COMPONENTS[1]", "LENS_PRIMITIVE")
LensPrimlKey$ = LTLISTNEXT$(LensList$)
DiameterStatus = LTDBSET(LensPrimlKey$, "DIAMETER", 1.5)
DeleteStatus = LTLISTDELETE(LensList$)
```

The value of DeleteStatus would be set to 0 if the deletion is successful, or greater than zero if the deletion were not successful (e.g., the list did not exist).

Functions to Analyze Object Keys

There are data access functions which are used to analyze object keys which are used to access data values in the LightTools database. The functions are listed below.

LTDBKEYDUMP Lists the valid data elements for an object key.

LTDBKEYSTR\$ Translates an object key into its equivalent data access string.

These functions are described in more detail in *Data Access Filters* on page 16.

Data Access Filters

Objects in LightTools follow a hierarchical structure. At the highest level of the hierarchy is the Lens_Manager database which contains all objects in LightTools. The Lens_Manager database consists of several databases at the next level of the hierarchy: The User_Materials database (glass and other materials data), the Spectral_Regions database (wavelength data), the NS_Rays database (non-sequential ray data), the Path_Manager (data associated with Imaging Path definitions), the Illum_Manager database (data associated with Illumination analysis such as source and receiver definitions), and the Components database (data associated with lenses and other solid objects, as well as text and polylines). Each of these databases consists of further subsets of objects. For example, the Components database contains all solid objects, such as lenses, cubes, prisms, etc. Each lens contains surfaces and each surface can contain zones which define optical properties for that surface. Each level of the LightTools hierarchy has a data access filter associated with it that is used in conjunction with the LightTools MACRO data access functions.

The LightTools hierarchy and data access filters are best illustrated in a Table view, as shown in Figure 4 below. Here we have the Components Table view for a model consisting of 2 lenses. The filterName column has been added to the Table view (right click on column and select filterName).

Name	filterName	Reflectance
- Components	COMPONENTS	
- Lens_1	SOLID	
- Boolean Operations		
- LP_5	CIRC_LENS_PRIMITIVE	
- LensFrontSurface	SPHERICAL_LENS_SURFACE	
- BareSurface	ZONE	
Reflect_Transmit	RT_AMPLITUDE_ZONE	0.00000
Specular	SPECULAR_DIRECTION_ZONE	
- zone_1	ZONE	
Reflect_Transmit	RT_AMPLITUDE_ZONE	0.00000
Specular	SPECULAR_DIRECTION_ZONE	
CircularZone	CIRC_ZONE_EXTENT	
+ LensRearSurface	SPHERICAL_LENS_SURFACE	
+ EdgeSurface	CYLINDER_SURFACE	
- Lens_2	SOLID	
- Boolean Operations		
- LP_6	CIRC_LENS_PRIMITIVE	
+ LensFrontSurface	SPHERICAL_LENS_SURFACE	
+ LensRearSurface	SPHERICAL_LENS_SURFACE	
+ EdgeSurface	CYLINDER_SURFACE	
+ Flat	PLANAR_SURFACE	
+ Flat	PLANAR_SURFACE	

Figure 4. The Components Table view showing LightTools hierarchy and data access filters

The LightTools hierarchy is shown via levels of indentation in the Names column and the filter associated with that level is listed in the filterName column. For example, if the user wanted to access all objects associated with the one of the 2 lenses, the filter “SOLID” would be used. If the user were interested in accessing the reflectance on a specific surface, the filter “RT_AMPLITUDE_ZONE” would be used.

A complete list of filters and their association with the levels of the LightTools hierarchy can be found in Appendix A, *Specific Data Access Filters*.

General Filters

In addition to the specific filters that appear in the filterName column, there exist general filters which contain a subset of specific filters. For example, the general filter PRIMITIVE can be used to create a list of all primitives in the model. This would be equivalent to creating several individual lists using specific filters such as CIRC_LENS_PRIMITIVE, RECT_LENS_PRIMITIVE, CUBE_PRIMITIVE, etc. All of these filters are at the same level of the hierarchy, but the general filter PRIMITIVE includes all types of primitives, while the specific filter CIRC_LENS_PRIMITIVE contains only circular lenses. General filters are useful for access to groups of similar object types at the same level in the data hierarchy. For a complete list of general filters and the specific filters they include, please refer to Appendix B, *General Data Access Filters*.

The following are examples of macros which employ general or specific filters to accomplish the same goal.

MACRO EXAMPLE 1

This macro creates a list of all lens primitives. It then sorts the circular lenses and rectangular lenses. It changes the diameters of the circular lenses and changes the width and height of the rectangular lenses.

```
REM   Create a list of all lens primitives and acquire the
REM   object key at the current position in the list.

LensPrimitiveList$ = LTDBLIST$("COMPONENTS[1]","LENS_PRIMITIVE")
LensPrimitive$ = LTLISTNEXT$(LensPrimitiveList$)

REM   Check to determine if the list of object keys is exhausted.
REM   If it is not exhausted, check to determine if the lens
REM   is circular or rectangular.
    WHILE LensPrimitive$ <> ""
        IF LTDBTYPE(LensPrimitive$,"CIRC_LENS_PRIMITIVE") THEN

REM   Change the diameter if the lens is circular or change
REM   the width and height if the lens is rectangular.
            DiameterStatus = LTDBSET(LensPrimitive$,"DIAMETER",25)

        ELSE
            WidthStatus = LTDBSET(LensPrimitive$,"WIDTH",25)
            HeightStatus = LTDBSET(LensPrimitive$,"HEIGHT",25)
        END IF
        LensPrimitive$ = LTLISTNEXT$(LensPrimitiveList$)
    WEND
```

MACRO EXAMPLE 2

This macro creates a list of all circular lens primitives and a list of all rectangular lens primitives. It then changes the diameters of the circular lenses and changes the width and height of the rectangular lenses.

```

REM   Create a list of all circular lens primitives and acquire
REM   the object key at the current position in the list.

CircLensList$ = LTDBLIST$("COMPONENTS[1]","CIRC_LENS_PRIMITIVE")
CircLensPrimitive$ = LTLISTNEXT$(CircLensList$)

REM   Check to determine if the list of object keys is exhausted.
REM   If it is not exhausted, change the diameter of the circular
REM   lens.

    WHILE CircLensPrimitive$ <> ""
        DiameterStatus = LTDBSET(CircLensPrimitive$,"DIAMETER",25)
        CircLensPrimitive$ = LTLISTNEXT$(CircLensList$)
    WEND

REM   Create a list of all rectangular lens primitives and acquire
REM   the object key at the current position in the list.

RectLensList$ = LTDBLIST$("COMPONENTS[1]","RECT_LENS_PRIMITIVE")
RectLensPrimitive$ = LTLISTNEXT$(RectLensList$)

REM   Check to determine if the list of object keys is exhausted.
REM   If it is not exhausted, change the width and height of the
REM   rectangular lens.

    WHILE RectLensPrimitive$ <> ""
        WidthStatus = LTDBSET(RectLensPrimitive$, "WIDTH",25)
        HeightStatus = LTDBSET(RectLensPrimitive$,"HEIGHT",25)
        RectLensPrimitive$ = LTLISTNEXT$(RectLensList$)
    WEND

```

Filter Index

When filters are used as object keys in the data access functions, they must include an index. The index can be a number (e.g., SOLID[2]), the name of an object (e.g., SOLID[Lens_07]), or a LightTools user-defined variable defined via the Define command and preceded with the @ character (e.g., SOLID[@firstlensnum]). A special variable, *last*, can also be used to point to the last object created (e.g., SOLID[@last]). Filter indices are also used to construct data access strings, which are discussed in detail in *Data Access Strings* on page 28.

Examples of using filter indices with data access functions:

```
REM    Create a list of surfaces for the 2nd solid in the model.

SurfList$ = LTDBLIST("SOLID[2]", "SURFACE")

REM    Change the diameter of the lens primitive named "LP_5"
REM    in the model.

DiamStatus = LTDBSET("CIRC_LENS_PRIMITIVE[LP_5]", "DIAMETER", 5.0)

REM    Change the Y coordinate of the last rectangular
REM    lens created.

YcoordStatus = LTDBSET("RECT_LENS_PRIMITIVE[@last]", "Y", -2.0)
```

Data Access Functions to Analyze Object Keys

The data access functions described below are used to determine the data elements that can be used with a particular object key to access data values in the LightTools database and to translate an object key into a data access string.

Function to List Valid Data Elements for an Object Key - LTDBKEYDUMP

The LTDBKEYDUMP function prints a list containing all valid data elements for a given object key. The list can be printed to a file or the LightTools Console window. The file name must be enclosed within double quotation marks. If the file name is omitted, the list is printed to the LightTools Console window. An example is given below.

Determine the valid data elements for a lens primitive:

```
LensList$ = LTDBLIST$("COMPONENTS[1]", "LENS_PRIMITIVE")
LensPrimlKey$ = LTLISTNEXT$(LensList$)
Data_elementStatus = LTDBKEYDUMP(LensPrimlKey$, "PrimData.txt")
```

The valid data elements for the first lens primitive are now in the file PrimData.txt. If the file was created without error, the value of Data_elementStatus would be set to zero. If there was an error, its value would be greater than zero.

Function to Translate an Object Key - LTDBKEYSTR\$

The LTDBKEYSTR\$ function translates a given object key and returns the equivalent data access string minus a data element. An example is given below.

Determine the equivalent data access string for a lens primitive object key:

```
LensList$ = LTDBLIST$("COMPONENTS[1]", "LENS_PRIMITIVE")
LensPrimlKey$ = LTLISTNEXT$(LensList$)
LensPrimlString$ = LTDBKEYSTR$(LensPrimlKey$)
```

The variable LensPrimlString\$ can now be used with valid data elements to construct a data access string to access the data values for the first lens primitive.

If the LTDBKEYSTR\$ function is used with a literal object key, it returns the name of the object as listed in a Table view. An example is given below.

Determine the name of the first lens primitive:

```
PrimlName$ = LTDBKEYSTR$("COMPONENTS[1].LENS_PRIMITIVE[1]")
```

The variable PrimlName\$ is now the name of the first lens primitive.

Data access strings are explained in more detail in *Data Access Strings* on page 28.

LightTools MACRO Commands

Commands specific to LightTools are included with the LightTools MACRO feature.

Command to Execute LightTools Commands Within a LightTools MACRO Program - LTCMD

The LTCMD command is used to execute any of the LightTools commands from within a LightTools MACRO. The LightTools command is sent to the LightTools command interpreter for processing. The LightTools command must be a string enclosed within double quotation marks (see Chapter 4). A complete listing of all LightTools commands, as well as their syntax and usage, can be found in the online document, *LightTools Command Reference Guide*. Examples of the use of the LTCMD command are shown below.

Open a new 3D Design view window:

```
LTCMD "3DDesignView"
```

Select all solids:

```
REM    Macro to select all solid objects using the MORE command.

REM    Create a list of all solid objects and get the first solid
REM    in the list.

SolidList$ = LTDBLIST$("COMPONENTS[1]","SOLID")
SolidKey$ = LTLISTNEXT$(SolidList$)

REM    Loop through the list of solids, get each solid's name,
REM    and perform the MORE command on that solid.

WHILE SolidKey$ <> ""
    SolidName$ = LTDBGET$(SolidKey$, "Name")
    LTCMD "More " + LTSTR$(SolidName$)
    SolidKey$ = LTLISTNEXT$(SolidList$)
WEND

REM    Delete the list of solid objects.

DeleteStatus = LTLISTDELETE(SolidList$)
```



Note: There is also an LTCMD function available in LightTools MACRO. It has a functionality similar to the LTCMD command and returns an integer status value indicative of the success of the execution of the LightTools command (see Chapter 4).

Command to Set LightTools MACRO Program Control Parameters - LTBSET

The LTBSET command is used to set values for LightTools MACRO program control parameters. The parameters control display of dialog boxes, error echoes, user interrupts, number of errors allowed, and LightTools database and graphic updates (see Chapter 4). Examples of the use of the LTBSET command are shown below.

Enable display of LightTools dialog boxes during macro execution:

```
LTBSET DIALOG 1
```

Terminate macro execution after ten errors:

```
LTBSET MAXERRORS 10
```



Note: There is also an LTBSET function available in LightTools MACRO. It has a functionality similar to the LTBSET command and returns an integer status value indicative of the success of the setting of the LightTools MACRO program control parameter (see Chapter 4).

LightTools MACRO Functions

Functions specific to LightTools are included with the LightTools MACRO feature.

Functions to Return a Formatted String - LTSTR\$, LTCOORD2\$, and LTCOORD3\$

The LTSTR\$ function is used to return a string enclosed within double quotation marks. This string can be used as input for the LTCMD command or other MACRO commands or functions. Examples are given below.

Select the first solid in a system:

```
SolidName1$ = LTDBGET$ ("SOLID[1]", "Name")
LTCMD "Select " + LTSTR$ (SolidName1$)
```

Set the value of a string variable to the name of the first solid:

```
SolidName1$ = LTDBGET$ ("SOLID[1]", "Name")
VariableStatus = LTSETVAR ("FIRST_SOLID", LTSTR$ (SolidName1$))
```

The LTCOORD2\$ and LTCOORD3\$ functions are used to return a string enclosed within double quotation marks. This string can be used as two or three dimensional input coordinates for the LTCMD command when creating LightTools objects. Examples are given below.

Create a sphere with a radius of 5 in a 2D Design view:

```
LTCMD "MCtrSphere " + LTCOORD2$ (0,0) + " " + LTCOORD2$ (0,5)
```

Create a sphere in a 3D Design view:

```
LTCMD "MCtrSphere " + LTCOORD3$ (0,0,0) + " " + LTCOORD3$ (0,5,0)
```

Function to Return the Status of a LightTools Command - LTGETSTAT

The LTGETSTAT function returns an integer value which is indicative of the status of the previously executed LightTools command. If the LightTools command was executed without error, the returned value is 0. If the LightTools command produced an error, the returned value is greater than zero. Examples are given below.

Create a sphere in a 3D Design view:

```
LTCMD "MCtrSphere " + LTCOORD3$ (0,0,0) + " " + LTCOORD3$ (0,5,0)
CommandStatus = LTGETSTAT ()
```

In this case, the value of CommandStatus would be 0 since the command MCtrSphere was executed without error.

```
LTCMD "MCtrSphere " + LTCOORD3$ (0,0,0) + "" + LTCOORD3$ (0,5,0)
CommandStatus = LTGETSTAT ()
```

In this case, the value of CommandStatus would be greater than zero since the command MCtrSphere would produce a syntax error due to the lack of a blank space between the LTCOORD3\$ functions.

Function to Evaluate an Expression - LTEVAL

The LTEVAL function is used to evaluate an expression which employs operands and operators. The expression is passed to the LightTools expression evaluator and a numeric value is returned. Examples are given below.

Determine the total number of SubObjects for two solids in a LightTools system:

```
Total=LTEVAL("{SOLID[1].numSubObjects}+{SOLID[2].numSubObjects}")
```

Determine the difference in position in the X direction for two solids in a LightTools system:

```
Total=LTEVAL("{SOLID[1].X}-{SOLID[2].X}")
```

Functions to Set and Acquire Values of LightTools Variables - LTSETVAR and LTGETVAR

Included in the LightTools MACRO feature are functions that allow the user to read LightTools variables and to assign values to LightTools variables within a LightTools MACRO program. Values can only be assigned to user-defined LightTools variables. Attempting to assign values to predefined LightTools variables results in an error. LightTools MACRO variables are not accessible with these functions. Examples are given below.

Set and acquire the value of a user-defined LightTools numeric variable:

```
Status = LTSETVAR("RANGE",25)
RANGE = LTGETVAR("RANGE")
```

Set and acquire the value of a user-defined LightTools string variable:

```
Shape1$ = LTDBGET$("CIRC_LENS_PRIMITIVE[1]","Element Shape")
Status = LTSETVAR("LENS_FORM",LTSTR$(Shape1$))
LENS_FORM$ = LTGETVAR$("LENS_FORM")
```

The LTSETVAR function is used to set values for both numeric and string user-defined LightTools variables. The LTGETVAR function is used to acquire values for numeric user-defined LightTools variables. The LTGETVAR\$ function is used to acquire values for string user-defined LightTools variables.

The LTGETVAR function can also be used to acquire the values of numeric predefined LightTools variables. These are explained in more detail in the section entitled LightTools and LightTools MACRO Variables which follows in this chapter.

Function to Return the Type of a LightTools Variable - LTCHECKVAR

The LTCHECKVAR function allows the user to determine the type of value that a LightTools variable has been assigned. The three types of values are integer numbers, real numbers, and character strings. The LTCHECKVAR function returns a value of 1 for integer variables, 2 for real variables, and 3 for string variables. If the variable is undefined, the LTCHECKVAR function returns a value of 0. Examples are given below.

Determine the type of a user-defined LightTools variable:

```
LTCMD "Define XYZ=123"
VariableType = LTCHECKVAR("XYZ")
```

The value of VariableType is set to 1 since the variable XYZ has an integer value.

Determine the type of a predefined LightTools variable:

```
VariableType = LTCHECKVAR("LINEAR_MEASURE_DISTANCE")
```

The value of VariableType is set to 2 since the variable LINEAR_MEASURE_DISTANCE has a value which is a real number (see *LightTools and LightTools MACRO Variables* on page 26).

Function to Return LightTools Version Information - LTVERSION\$

The LTVERSION\$ function returns a string which is composed of information regarding the version of LightTools software that is being used. There are four options for the numeric parameter which is used with the LTVERSION\$ function. The default value of 0 returns the version number. The value of 1 returns the version build date. The value of 2 returns the version name and number. The value of 3 returns the version name and number and build date. Examples are given below.

Determine the version name and number of the software:

```
VersionName$ = LTVERSION$(2)
```

The string variable VersionName\$ is set to the name and number of the software version.

Determine the date that the version of software was built:

```
VersionDate$ = LTVERSION$(1)
```

The string variable VersionDate\$ is set to the date that the software version was built.

LightTools and LightTools MACRO Variables

There are two types of LightTools variables: predefined and user-defined.

The Measure commands in LightTools sets predefined variables with the resulting values of the measurement process. These variables are created for linear and angular measurements using the LinearMeasure and AngularMeasure commands, respectively. Thus, any LightTools MACRO program can access these values using the LTGETVAR function.

The predefined variables are created in LightTools after the first Measure command is executed. After that time the variables listed below exist. All of the variables are created, but the values are determined by the specific Measure command issued. These are system variables and users may not redefine these variables using the Define command. This results in an error. This is also true if an attempt is made to assign values to the variables with the LTSETVAR function within a LightTools MACRO program.

Linear Measurement Variables:

```

LINEAR_MEASURE_DISTANCE
LINEAR_MEASURE_DELTA_X
LINEAR_MEASURE_DELTA_Y
LINEAR_MEASURE_DELTA_Z
LINEAR_MEASURE_ALPHA
LINEAR_MEASURE_BETA
LINEAR_MEASURE_GAMMA

```

Angular Measurement Variables:

```

ANGULAR_MEASURE_ANGLE
ANGULAR_MEASURE_CROSS_X
ANGULAR_MEASURE_CROSS_Y
ANGULAR_MEASURE_CROSS_Z

```

The Define command in LightTools is used to set values for user-defined variables. The variables can be set equal to a value or a data access string constructed with data access filters and data elements. The variable name must be followed by an equal sign and the value or data access string with no intervening spaces. String values must be enclosed within double quotation marks. Examples of the use of the Define command are given below.

Set the variable TOTAL to the value 4:

```
Define TOTAL=4
```

Set the variable NAME to the value Lens:

```
Define NAME="Lens"
```

Set the variable ZVAL to the Z position of the last segment of an NS ray:

```
Define ZVAL={NS_RAY[1].NS_SEGMENT[@last].global_z}
```

Set the variable AVE to the average illuminance value on receiver 10:

```
Define AVE={RECEIVER[receiver_10].MESH[illuminanceMesh_19].average}
```

Set the variable CELL to the intensity value at cell (2,3) in the mesh on the last receiver:

```
Define CELL={RECEIVER[@last].MESH[0].cellvalue[2][3]}
```

Once user-defined variables have been defined, they can be listed in the LightTools Console window by using the Define command with a question mark. An example is given below.

The command sequence:

```
Define TOTAL=4  
Define NAME="Lens"  
Define ?
```

produces this message in the Console window:

```
Info: SYMBOL TOTAL = 4  
Info: SYMBOL NAME = "Lens"
```

LightTools MACRO variables are defined using the LET command in a LightTools MACRO program (see Chapter 4). Examples of LightTools MACRO variable definitions are given below.

```
LET X = 2  
or X = 2
```

Data Access Strings

Data access strings are constructed using data access filters and data elements. They can be used to set or read data values of data elements in the LightTools database.

Data Access String Syntax

Data access strings must be enclosed within braces { }.

Data access strings specify data values using data access filters with indices and a data element with indices, if necessary.

The data access filters and the data element must be delimited by periods.

For example:

```
{COMPONENTS[1].SOLID[Lens_1].SURFACE[LensFrontSurface].Curvature}
```

Data Access Filter and Index

The data access filters follow the hierarchy described in *Data Access Filters* on page 16. The index is specified in the same manner as described in that section. The filter and index portion of the data access string may be repeated as many times as needed.

Data Element

Some data elements allow access to one or two dimensional data value arrays and thus require subscripts to point to the specific array element. Subscripts are designated using square bracket notation following the data element name. One dimensional array elements require one open bracket, an integer subscript, and one close bracket. Two dimensional array data requires the same syntax for each individual subscript. User-defined variables may be used as subscripts by preceding the variable name with the @ character.

Data Access String Uses

Data access strings can be used to set data values for data elements on a command line. The data values are set by following the last brace of the data access string with an equal sign and a data value. No spaces are permitted between the last brace of the data access string, the equal sign, and the data value. The data value can be a number, a string enclosed within double quotation marks, or a data access string. Data access strings can only be used in this manner at a LightTools command prompt or as a LightTools command processed by the LightTools MACRO command, LTCMD. For example:

Change the curvature of the front surface of Lens_1:

```
{SOLID[Lens_1].SURFACE[LensFrontSurface].Curvature}=.1
```

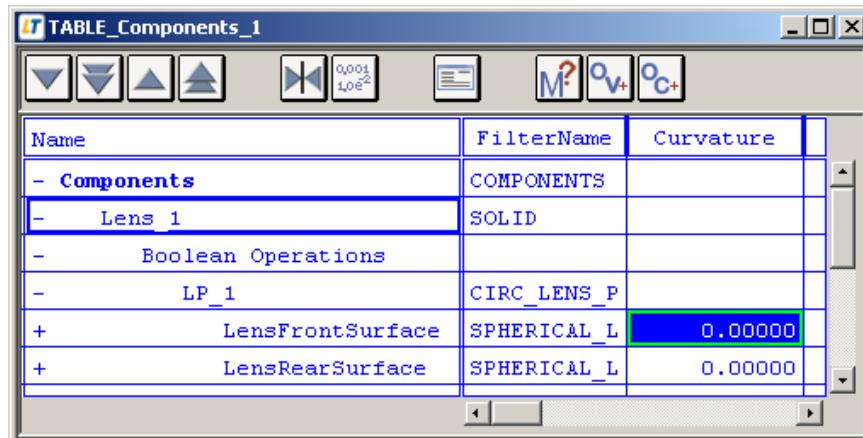
Make the Z position of Lens_1 the same as the Z position of Lens_2:

```
{SOLID[Lens_1].PRIMITIVE[1].Z}={SOLID[Lens_2].PRIMITIVE[1].Z}
```

Change the maximum hits parameter of the last surface of the last solid to 40:

```
LTCMD "{SOLID[@last].SURFACE[@last].Max_hits}=40"
```

Data access strings can be used to set data values for data elements in a Table view. This is done by entering the data access string at the “Enter value for cell.” prompt in the Table view window. The cell value is set to the data value specified by the data access string. For example, in the Table view shown in Figure 5 below, the curvature of the front surface of Lens_1 is set to the curvature of its rear surface.



Name	FilterName	Curvature
- Components	COMPONENTS	
- Lens_1	SOLID	
- Boolean Operations		
- LP_1	CIRC_LENS_P	
+ LensFrontSurface	SPHERICAL_L	0.00000
+ LensRearSurface	SPHERICAL_L	0.00000

Figure 5. Using data access strings to set data values for data elements in a table view

Data access strings can be used to assign values to user-defined LightTools variables with the LightTools Define command. Data access strings can only be used in this manner at a LightTools command prompt or as a LightTools command processed by the LightTools MACRO command, LTCMD.

At a command prompt, set the variable ZVAL to the Z position of the last segment of an NS ray:

```
Define ZVAL={NS_RAY[1].NS_SEGMENT[@last].Global Z}
```

At a command prompt, set the variable AVE to the average illuminance value on receiver 10:

```
Define AVE={RECEIVER[receiver_10].MESH[illuminanceMesh_19].Average}
```

In a macro, set the variable CELL to the intensity value at cell (2,3) in the mesh on the last receiver:

```
LTCMD "Define CELL={RECEIVER[@last].MESH[1].CellValue[2][3]}"
```

Data access strings can be read in a dialog box field by entering them in the field. Data access strings can be used as operands in an arithmetic expression. Data access strings cannot contain any arithmetic expressions. This means that arithmetic expressions cannot be used as a filter or array index. User-defined variables are limited to use as index values and shall not be used to specify any other portion of the data access string.

Data Access Strings versus Data Access Lists

Both data access strings and data access lists can be used to access data values for data elements in the LightTools database. However, when the database is accessed on a repeated basis, the search time for data values becomes an issue. When data access strings are used, the database must be completely searched for the data value specified by the data access string each time the data access string is encountered. The search is performed according to the filter sequence specified by the data access string. If a data access list has been created which contains object keys corresponding to the data value of interest, only the data access list must be searched to access the data value. Therefore, the choice of

whether to use data access strings or data access lists is dependent on how often data values are to be accessed. The trade-off is the database search time versus the list creation and search time and the programming effort involved in each case.

The macro example shown below illustrates this point. In this case, the Maximum Number of Hits value is to be changed on all surfaces. A list of all the surfaces is created, and then the MAX HITS value is changed for each surface in the list. For a system with several surfaces, this approach is much more efficient than using data access strings to make the changes. However, if the system contained 1 or 2 surfaces, the use of data access strings would prove to be more efficient.

```
REM      Macro to change the Maximum Number of Hits value on all
REM      surfaces from the default of 1000 to 2000.

REM      Create a list of all surfaces and get the first surface
REM      in the list.

SurfaceList$ = LTDBLIST$("COMPONENTS[1]","SURFACE")
SurfaceKey$ = LTLISTNEXT$(SurfaceList$)

      REM      Loop through the list of surfaces and change the
      REM      MAX HITS value to 2000.

      WHILE SurfaceKey$ <> ""
        Status = LTDBSET(SurfaceKey$,"MAX HITS",2000)
        SurfaceKey$ = LTLISTNEXT$(SurfaceList$)
      WEND

REM      Delete the surface list.

DeleteStatus = LTLISTDELETE(SurfaceList$)
```

Window Commands

There are a set of commands within LightTools that are equivalent to operations performed by clicking the mouse on windows, dialog boxes, or buttons. Examples of several of these commands are shown below. The untitled qualifier used in several of the commands below refers to the name of an untitled LightTools system file. In an actual session, the specified title of the LightTools system file would be used as the qualifier.

<code>\VConsole</code>	Set focus to the Console window. Case sensitive.
<code>\V3D</code>	Set focus to the 3D Design view window. Case Sensitive.
<code>\V2D_untitled, \VPath_untitled,</code>	Set focus to specific 2D window (such as charts and tables).
<code>\VNSRayFootprint_Rays_1,</code>	Note that you must use specific title for the window. The
<code>\VTABLE_untitled,</code>	title can be retrieved using the following command:
<code>\VTABLE_Components_1</code>	<code>ViewTitle = lt.ViewGet("View[@Last]", "Title")</code>

Whenever any of these operations is performed via the mouse, the equivalent command is written to the LightTools Recover File (LightTools.n.ltr, where n is the version number). Therefore, it is possible to determine the equivalent window command for a mouse action by examining the contents of this file. These window commands can then be executed within a LightTools macro program by using the LTCMD command with any of the window commands as an argument. Therefore, the user has access to windows, dialog boxes, and data fields while executing a LightTools macro program. This allows the user interactive control over LightTools in the manner of a graphical user interface from within a LightTools macro program.

Use the `\O \Q` construct to identify changes for a specific object. `\O` introduces the construct, followed by the object access string and one or more lines indicating the data to change and its value. `\Q` terminates the construct. For example,

```
\O LENS_MANAGER[1]
units = nanometers
\Q
```


Chapter 4 Command/Function Reference

This chapter contains an alphabetical listing of all commands and functions that are available with LightTools MACRO. It includes the subset of ANSI Full BASIC commands and functions that are provided by LightTools MACRO, as well as the complete set of *LightTools* specific commands and functions that are provided as an extension to the ANSI Full BASIC subset. For each entry there is a detailed description explaining the functionality and syntax of the command or function, as well as examples of its use. Included with each entry are explanations of the following items:

DESCRIPTION

TYPE (Command or Function)

SYNTAX

ARGUMENTS

RETURN VALUES (Functions only)

NOTES (Prerequisites, Value Ranges)

EXAMPLES

Conventions

The following conventions are used in this Command/Function Reference:

Generic arguments are designated by characters with a style that is both boldface and italicized.

EXAMPLES: ***X*** , ***device-number***

Functions that return string values only and variables that are assigned string values only are designated by a trailing dollar sign (\$).

EXAMPLES: STRING\$, ***S\$***

Constant string arguments must be enclosed within double quotation marks. The exceptions to this convention are the ***label\$*** arguments used in the CALL, SUB, FUNCTION, GOTO, GOSUB, and ON ERROR GOSUB commands, and the ***remark\$*** argument used in the REM command.

EXAMPLES: "ABCDE" , "123ABC"

Functions that return numeric values only and variables that are assigned numeric values only do not utilize a trailing character.

EXAMPLES: ABS , ***X***

Some variables can be assigned numeric or string values. These variables do not utilize a trailing character.

EXAMPLES: ***Ak*** , ***parameterk***

Command/function arguments that are enclosed within parentheses are to be entered in this manner in an actual program statement.

EXAMPLES: (***X***) , (***S\$***)

Command/function components that are separated by a vertical bar (|) are options from which a single component is selected.

EXAMPLES: ELSE | ELSEIF , ***label\$*** | ***line***

Command/function components that are enclosed within brackets ([]) are optional. They are not required in a program statement.

EXAMPLES: [STEP ***increment***] , [, ***V2*** ...]

Any delimiting punctuation symbols (e.g., ; : +) that are enclosed within brackets are also optional.

EXAMPLES: [, ***V2*** ...] , [, | ; | + ***A2*** ...]

The consecutive period symbol (...) used after arguments in the command/function syntax specifications implies that the preceding arguments can be continuously entered in a similar fashion up to a limit determined by the general program limits.

EXAMPLES: [, ***C2*** ...] , [, ***parameter2*** ...]

The consecutive period symbol (...) used after the exponential number e is indicative of an incompletely displayed number. That is, the digits in the number continue on to the limit of precision for a LightTools MACRO numeric value. That limit of precision is double precision with at most 17 significant digits.

EXAMPLES: e = 2.7182818...

The examples given for the commands/functions contain listings of program input and output. The input section of an example is formatted as plain text. The output section of an example is formatted as boldface text.

```

EXAMPLES:  X = 25.52
           Y = ABS(X)
           PRINT X;Y

           25.5200000    25.5200000

```

In some examples of the commands/functions, there are annotations for program output that are enclosed within braces ({ }). These annotations are not indicative of the actual program output.

```

EXAMPLES:  {This is the output in a LightTools dialog
           box. Enter the values 1,2 for X and Y in the
           LightTools dialog box and click Apply.}

```

In some examples of the commands/functions, there are listings of the contents of input and output files. These are provided as a means of clarification for the example. They are not indicative of the actual program input and output. They are indicative of the contents of the actual files that are used for input and output in the examples.

```

EXAMPLES:  Contents of myfile.txt:
           1
           2
           3

```

Subprograms can be defined by using a label only or by using the SUB command with a label. Subprograms defined by a label only are called by the GOSUB, and ON ERROR GOSUB commands which direct the program to a label. These subprograms must end with a RETURN command. Subprograms defined by the SUB command and a label are called by the CALL command. These subprograms accept input and/or output parameters passed by the CALL command. These subprograms must end with an END SUB command. Subprograms can also be defined by using a line number only. These subprograms are called by the GOSUB and ON GOSUB commands which direct the program to a line number. These subprograms must end with a RETURN command.

```

EXAMPLES:  GOSUB MARINE
           MARINE:
           Z = X/Y
           RETURN

           CALL MARINE(A,B,C)
           SUB MARINE(X,Y,Z)
           Z = X/Y
           END SUB

           1      I = 1
           2      ON I GOSUB 4
           3      END
           4      Z = X/Y
           5      RETURN

```

ABS

DESCRIPTION:

Returns the absolute value of a number which is the magnitude of the number regardless of sign.

TYPE:

Function (Mathematical)

SYNTAX:

$\text{ABS}(X)$

ARGUMENTS:

The X argument is the number for which the absolute value is returned.

RETURN VALUES:

The absolute value of the X argument.

NOTES:

The absolute value of a positive number is the positive number. The absolute value of a negative number is the negative value of the negative number. The absolute value of zero is zero. The absolute value of a number is expressed mathematically as $\text{ABS}(X) = |X|$.

The X argument must be a real number. It can be a constant, variable, function, or expression. The range of values is $-\infty < X < \infty$.

The range of values for $\text{ABS}(X)$ is $0 \leq \text{ABS}(X) < \infty$. The returned value of ABS has the same precision as its argument.

EXAMPLES:

1.	$X = 25.52$ $Y = \text{ABS}(X)$ $\text{PRINT } X, Y$	
	25.5200000	25.5200000
2.	$X = -25.52$ $Y = \text{ABS}(X)$ $\text{PRINT } X, Y$	
	-25.5200000	25.5200000
3.	$X = 0.0$ $Y = \text{ABS}(X)$ $\text{PRINT } X, Y$	
	0	0

ACOS

DESCRIPTION:

Returns the arccosine of a number in radians. The arccosine is useful for calculating the angle of a line whose length and horizontal component are known.

TYPE:

Function (Mathematical)

SYNTAX:

`ACOS(X)`

ARGUMENTS:

The *X* argument is the number for which the arccosine is returned.

RETURN VALUES:

The arccosine of the *X* argument.

NOTES:

The arccosine is the angle in radians whose cosine is equal to the argument of this function. The *X* argument must be a real number. It can be a constant, variable, function, or expression. The range of values is $-1 \leq X \leq 1$.

The range of values for `ACOS(X)` is $0 \leq \text{ACOS}(X) \leq \pi$. The returned value of `ACOS` has the same precision as its argument. The `ACOS` function is the inverse of the `COS` function. For example, $X = \text{ACOS}(\text{COS}(X))$.

EXAMPLES:

```
1.  X = 0.5
    Y = ACOS(X)
    PRINT X,Y

    0.5000000      1.0471976

2.  X = -0.5
    Y = ACOS(X)
    PRINT X,Y

    -0.5000000     2.0943951
```

ASC

DESCRIPTION:

Returns the ASCII code number for the first character in a string.

TYPE:

Function (String)

SYNTAX:

ASC(*S*)

ARGUMENTS:

The *S* argument is the string for which the ASCII value is returned.

RETURN VALUES:

The ASCII code number for the first character in the *S* argument.

NOTES:

The *S* argument must be a defined string. It can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). The first character of the *S* argument (other than the quotation marks) must be an ASCII character. Each character is assigned an ASCII code number which is an integer from 0 to 127. The ASC function is undefined for the null string ("").

EXAMPLES:

- ```
S = "ABCDE"
X = ASC(S)
PRINT S;X
```

**ABCDE    65**
- ```
X = ASC("ABCDE")
PRINT X
```

65

ASIN

DESCRIPTION:

Returns the arcsine of a number in radians. The arcsine is useful for calculating the angle of a line whose length and vertical component are known.

TYPE:

Function (Mathematical)

SYNTAX:

ASIN(*X*)

ARGUMENTS:

The *X* argument is the number for which the arcsine is returned.

RETURN VALUES:

The arcsine of the *X* argument.

NOTES:

The arcsine is the angle in radians whose sine is equal to the argument of this function. The *X* argument must be a real number. It can be a constant, variable, function, or expression. The range of values is $-1 \leq X \leq 1$.

The range of values for ASIN(*X*) is $-\pi/2 \leq \text{ASIN}(X) \leq \pi/2$. The returned value of ASIN has the same precision as its argument. The ASIN function is the inverse of the SIN function. For example, $X = \text{ASIN}(\text{SIN}(X))$.

EXAMPLES:

```
1.  X = 0.5
    Y = ASIN(X)
    PRINT X,Y

    0.5000000      0.5235988

2.  X = -0.5
    Y = ASIN(X)
    PRINT X,Y

    -0.5000000    -0.5235988
```

ATAN2

DESCRIPTION:

Returns the arctangent of the ratio of two numbers in radians. This form of the arctangent function is useful for calculating the angle of a line whose vertical and horizontal components (i.e., slope) are known.

TYPE:

Function (Mathematical)

SYNTAX:

ATAN2(*Y*, *X*)

ARGUMENTS:

The *Y* and *X* arguments are the numbers for which the arctangent is returned.

RETURN VALUES:

The arctangent of the ratio of the *Y* argument to the *X* argument (i.e., Y/X).

NOTES:

The arctangent is the angle in radians whose tangent is equal to the ratio of the arguments of this function. The *Y* and *X* arguments must be real numbers. They can be constants, variables, functions, or expressions. The ranges of values are $-\infty < Y < \infty$ and $-\infty < X < \infty$. The range of values for their ratio is $-\infty < Y/X < \infty$.

The range of values for ATAN2(*Y*,*X*) is $-\pi < \text{ATAN2}(Y,X) \leq \pi$. The returned value of ATAN2 has the same precision as its arguments. If the *X* and *Y* arguments are 0, the ATAN2 function returns a value of 0.

EXAMPLES:

```
1.  X = 5
    Y = 2
    Z = ATAN2 (Y, X)
    PRINT X, Y, Z

    5      2      0.3805064
```


ATN

DESCRIPTION:

Returns the arctangent of a number in radians. This form of the arctangent function is useful for calculating the angle of a line whose slope (i.e., ratio of the vertical component to the horizontal component) is known.

TYPE:

Function (Mathematical)

SYNTAX:

$\text{ATN}(X)$

ARGUMENTS:

The X argument is the number for which the arctangent is returned.

RETURN VALUES:

The arctangent of the X argument.

NOTES:

The arctangent is the angle in radians whose tangent value is equal to the argument of this function. The X argument must be a real number. It can be a constant, variable, function, or expression. The range of values is $-\infty < X < \infty$.

The range of values for $\text{ATN}(X)$ is $-\pi/2 < \text{ATN}(X) < \pi/2$. The returned value of ATN has the same precision as its argument. The ATN function is the inverse of the TAN function. For example, $X = \text{ATN}(\text{TAN}(X))$.

EXAMPLES:

```
1.  X = 0.4
    Y = ATN(X)
    PRINT X, Y

    0.4000000      0.3805064

2.  X = -0.4
    Y = ATN(X)
    PRINT X, Y

    -0.4000000     -0.3805064
```

CALL

DESCRIPTION:

Calls a subprogram and passes its associated parameters.

TYPE:

Command (Program Control)

SYNTAX:

CALL *label\$* [(*parameter1* [, *parameter2* ...])]

ARGUMENTS:

The *label\$* argument is the name of the subprogram called.

The *parameterk* arguments are the parameters passed to the subprogram.

RETURN VALUES:

None.

NOTES:

A subprogram is an auxiliary program executed within a LightTools Macro program and is used to perform unique or repeated operations. The subprogram is defined by the SUB command. When the CALL command is executed, the program is directed to the statement containing the SUB keyword which is followed by the name of the subprogram. The input and/or output parameters used by the subprogram are passed to the subprogram by the CALL command. They are listed after the subprogram name and must be delimited by commas. The parameter list may be enclosed within parentheses. If the parameter list is enclosed within parentheses, the parameter list and its associated parentheses must be contiguous with the *label\$* argument. The subprogram designated by the *label\$* argument must be defined.

The *parameterk* arguments are optional. If the subprogram called does not require input and/or output parameters, they are not required as arguments. The k character in the *parameterk* arguments is a symbolic index that corresponds to the numerical order of the parameters entered as arguments.

EXAMPLES:

```
1.  A = 3
    B = 2
    CALL MAXPLUS1 (A, B, C)
    PRINT A, B, C
    END

        SUB MAXPLUS1 (X, Y, Z)
            Z = MAX (X, Y) + 1
        END SUB

        3          2          4
```

CEIL

DESCRIPTION:

Returns the whole number with the smallest value that is not less than a number.

TYPE:

Function (Mathematical)

SYNTAX:

`CEIL(X)`

ARGUMENTS:

The *X* argument is the number for which the whole number is returned.

RETURN VALUES:

The smallest whole number not less than the *X* argument.

NOTES:

The *X* argument must be a real number. It can be a constant, variable, function, or expression. The range of values is $-\infty < X < \infty$.

If the *X* argument is a whole number, then the value of `CEIL(X)` is the *X* argument. The `CEIL` function returns a whole number with the same precision as its argument.

EXAMPLES:

```
1.  X = 49.001
    Y = CEIL(X)
    PRINT X, Y

    49.0010000      50

2.  X = 49.000
    Y = CEIL(X)
    PRINT X, Y

    49      49
```

CHR\$

DESCRIPTION:

Returns a one character string whose character corresponds to an ASCII code number.

TYPE:

Function (String)

SYNTAX:

CHR\$(*N*)

ARGUMENTS:

The *N* argument is the ASCII code number.

RETURN VALUES:

The one character string that corresponds to the *N* argument.

NOTES:

The ASCII code number must be a whole number ranging from 0 to 127, inclusive. The range of values for the *N* argument is $-0.999... \leq N \leq 127.999...$. It can be a constant, variable, function, or expression. If the *N* argument is not a whole number, the decimal places in its value are truncated to produce a whole number.

The CHR\$ function is the inverse of the ASC function. For example, $N = \text{CHR}$(\text{ASC}(N))$.

EXAMPLES:

```
1.  N = 65
    S$ = CHR$(N)
    PRINT N, S$

    65      A
```

CLOSE

DESCRIPTION:

Closes a file designated by a device number.

TYPE:

Command (File Input/Output)

SYNTAX:

CLOSE [#] *device-number*

ARGUMENTS:

The *device-number* argument is the device number assigned to the file.

RETURN VALUES:

None.

NOTES:

The file is designated by a device number. The file must be open. The file is opened by the OPEN FOR APPEND|INPUT|OUTPUT AS # commands. Valid device numbers are 0 through 15. The pound (#) character that precedes the *device-number* argument is optional.

EXAMPLES:

```
1.  OPEN "myfile1.txt" FOR INPUT AS #15
    INPUT #15,X
    CLOSE #15
    PRINT X
```

100

Contents of myfile1.txt:

100

COS

DESCRIPTION:

Returns the cosine of an angle which is in radians.

TYPE:

Function (Mathematical)

SYNTAX:

$\text{COS}(X)$

ARGUMENTS:

The X argument is the angle for which the cosine is returned.

RETURN VALUES:

The cosine of the X argument.

NOTES:

The cosine is for the angle in radians represented by the argument of this function. The X argument must be a real number. It can be a constant, variable, function, or expression. The range of values is $0 \leq X \leq \pi$.

The range of values for $\text{COS}(X)$ is $-1 \leq \text{COS}(X) \leq 1$. The returned value of COS has the same precision as its argument. The COS function is the inverse of the ACOS function. For example, $X = \text{COS}(\text{ACOS}(X))$.

EXAMPLES:

```
1.  X = 0.5
    Y = COS (X)
    PRINT X, Y

    0.5000000      0.8775826

2.  X = -0.5
    Y = COS (X)
    PRINT X, Y

    -0.5000000    0.8775826
```

DATA

DESCRIPTION:

Stores numeric and string constants to be accessed by the READ command.

TYPE:

Command (Program Control)

SYNTAX:

DATA *CI* [, *C2* ...]

ARGUMENTS:

The *Ck* arguments are the constants stored.

RETURN VALUES:

None.

NOTES:

The DATA command is only used in conjunction with READ and RESTORE commands. The data stored is accessed by the READ command.

Only the *CI* argument is required. The other *Ck* arguments are optional. The *Ck* arguments can be numbers or strings. They must be constants. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). The *Ck* arguments must be delimited by commas. The k character in the *Ck* arguments is a symbolic index that corresponds to the numerical order of the constants entered as arguments.

EXAMPLES:

```
1.  DATA "ABC", 99
    READ S$, X
    PRINT S$, X

    ABC      99
```

DATE\$

DESCRIPTION:

Returns the current date based on the computer's internal clock.

TYPE:

Function (String)

SYNTAX:

DATE\$()

ARGUMENTS:

The () argument is the null character enclosed within parentheses.

RETURN VALUES:

The current date.

NOTES:

The returned value of DATE\$ is a string in the form "MM-DD-YYYY," where MM is the two digit month, DD is the two digit day, and YYYY is the four digit year.

EXAMPLES:

```
1.  S$ = DATE$()  
    PRINT S$
```

12-31-1999

DEG

DESCRIPTION:

Returns the equivalent number of degrees for an angle in radians.

TYPE:

Function (Mathematical)

SYNTAX:

DEG(*X*)

ARGUMENTS:

The *X* argument is the number of radians in the angle for which the number of degrees is returned.

RETURN VALUES:

The number of degrees in the *X* argument.

NOTES:

The *X* argument must be a real number. It can be a constant, variable, function, or expression. It represents an angle in radians. The range of values is $0 \leq X \leq 2\pi$.

The range of values for DEG(*X*) is $0^\circ \leq \text{DEG}(X) \leq 360^\circ$. The returned value of DEG has the same precision as its argument. The DEG function is the inverse of the RAD function. For example, $X = \text{DEG}(\text{RAD}(X))$.

EXAMPLES:

```
1.  X = 5.555
    Y = DEG(X)
    PRINT X,Y

    5.5550000          318.2780552
```

DIM

DESCRIPTION:

Dimensions variables as one, two, or three dimensional arrays.

TYPE:

Command (Miscellaneous)

SYNTAX:

`DIM VI(NII [, NI2, NI3]) [, V2(N2I [, N22, N23]) ...]`

ARGUMENTS:

The *Vk* arguments are the variables dimensioned as arrays.

The *Nk1*, *Nk2*, and *Nk3* arguments are the dimensions of the arrays.

RETURN VALUES:

None.

NOTES:

The DIM command assigns dimensions to variables that have multiple values ordered in a particular sequence. The variables can be dimensioned as one, two, or three dimensional arrays. The dimension arguments *Nk1*, *Nk2*, and *Nk3* correspond to the first, second, and third dimensions for one, two, and three dimensional arrays, respectively. Only the *VI* and *NII* arguments are required. The other *Vk* and *Nk* arguments are optional. The *Vk* arguments must be delimited by commas, as must the *Nk* arguments. The k character in the *Vk*, *Nk1*, *Nk2*, and *Nk3* arguments is a symbolic index that corresponds to the numerical order of the variables entered as arguments.

The array variables are considered subscripted variables. The array variables have dimensions numbered from 1 to *Nk1*, *Nk2*, or *Nk3* for the first, second, or third dimensions, respectively. The values of an array variable must be all numeric or all string. An array variable cannot have mixed value types.

The dimensions of the arrays must be whole numbers. The range of values for the *Nk1*, *Nk2*, and *Nk3* arguments are $1 \leq Nk1 < \infty$, $1 \leq Nk2 < \infty$, and $1 \leq Nk3 < \infty$. They can be constants, variables, functions, or expressions. If the *Nk1*, *Nk2*, or *Nk3* arguments are not whole numbers, the decimal places in their values are truncated to produce whole numbers.

EXAMPLES:

```
1.  DIM X(5)
    X(1) = 99
    X(2) = 98
    X(3) = 97
    X(4) = 96
    X(5) = 95
    PRINT X(1),X(2),X(3),X(4),X(5)
```

99	98	97	96	95
-----------	-----------	-----------	-----------	-----------

```
2.  DIM Y(2,2)
    Y(1,1) = 99
    Y(1,2) = 98
    Y(2,1) = 97
    Y(2,2) = 96
    PRINT Y(1,1),Y(1,2)
    PRINT Y(2,1),Y(2,2)
```

99	98
97	96

```
3.  DIM Z(2,2,2)
    Z(1,1,1) = 99
    Z(1,1,2) = 98
    Z(1,2,1) = 97
    Z(1,2,2) = 96
    Z(2,1,1) = 95
    Z(2,1,2) = 94
    Z(2,2,1) = 93
    Z(2,2,2) = 92
    PRINT Z(1,1,1),Z(1,1,2)
    PRINT Z(1,2,1),Z(1,2,2)
    PRINT Z(2,1,1),Z(2,1,2)
    PRINT Z(2,2,1),Z(2,2,2)
```

99	98
97	96
95	94
93	92

DO

DESCRIPTION:

Initiates a DO loop which executes a sequence of program statements in a repetitive fashion. The LOOP command is used in conjunction with the DO command to repeat the DO loop.

TYPE:

Command (Program Control)

SYNTAX:

DO

ARGUMENTS:

None.

RETURN VALUES:

None.

NOTES:

The DO command is the first statement in a sequence of statements which constitutes a DO loop. A DO loop can be exited at any point by use of the EXIT DO command. This is the only means of terminating a DO loop. The LOOP command marks the end of a sequence of program statements which constitutes a DO loop. When a DO loop is terminated, the program is directed to the statement immediately following the LOOP command.

EXAMPLES:

```
1.  X = 0
    DELTAX = 25
    DO
        X = X + DELTAX
        IF X >= 100 THEN
            EXIT DO
        END IF
        PRINT "X = ";X
    LOOP

    X = 25
    X = 50
    X = 75
```

DO WHILE

DESCRIPTION:

Initiates a DO WHILE loop which executes a sequence of program statements in a repetitive fashion while a defined criterion is met. The LOOP command is used in conjunction with the DO WHILE command to repeat the DO WHILE loop.

TYPE:

Command (Program Control)

SYNTAX:

DO WHILE *criterion*

ARGUMENTS:

The *criterion* argument is the criterion against which the loop test results are measured.

RETURN VALUES:

None.

NOTES:

The DO WHILE command is used to begin a sequence of program statements that are executed in a repetitive fashion while a defined criterion is met. The criterion is entered as an argument on the command line. Each time a statement containing the item for which the criterion was defined is encountered, a test is made to determine if the criterion has been met. If it has been met, the sequence of program statements continues. If it has not been met, the DO WHILE loop is terminated.

The DO WHILE command is the first statement in a sequence of statements which constitutes a DO WHILE loop. A DO WHILE loop can be exited at any point by use of the EXIT DO command. The LOOP command marks the end of a sequence of program statements which constitutes a DO WHILE loop. When a DO WHILE loop is terminated, the program is directed to the statement immediately following the LOOP command.

EXAMPLES:

```
1.  X = 0
    DELTAX = 25
        DO WHILE  X <= 100
            X = X + DELTAX
            PRINT "X = ";X
        LOOP

X = 25
X = 50
X = 75
X = 100
X = 125
```

DOC

DESCRIPTION:

Writes a documented list of the entire set of commands and functions that are available for use in a LightTools MACRO program to a file.

TYPE:

Command (File Input/Output)

SYNTAX:

DOC *file-name*\$

ARGUMENTS:

The *file-name*\$ argument is the name of the file.

RETURN VALUES:

None.

NOTES:

The list of commands and functions includes all of the entries in this Command/Function Reference. It also includes Return Types and Numbers of Parameters for the functions. The *file-name*\$ argument can be a string constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "myfile.txt").

EXAMPLES:

1. DOC "docfile.txt"

{The file docfile.txt in text format which contains the list.}

ECHO

DESCRIPTION:

Enables or disables the display of executed LightTools MACRO program statements in the LightTools Console window.

TYPE:

Command (Program Control)

SYNTAX:

ECHO ON | OFF

ARGUMENTS:

The ON argument enables the display.

The OFF argument disables the display.

RETURN VALUES:

None.

NOTES:

The ECHO command allows the display of LightTools MACRO program statements in the LightTools Console window as they are being executed. The commands are displayed with three consecutive periods and a blank space as leading characters (e.g., ... X = 1).

EXAMPLES:

```
1.  ECHO ON
    X = 1
    Y = 2
    Z = 3
    ECHO OFF
    X = 4
    Y = 5
    Z = 6

    ... X = 1
    ... Y = 2
    ... Z = 3
    ... ECHO OFF
```

END

DESCRIPTION:

Terminates a LightTools MACRO program and closes all associated files.

TYPE:

Command (Program Control)

SYNTAX:

END

ARGUMENTS:

None.

RETURN VALUES:

None.

NOTES:

The END command is generally the last statement in the program. This command can be used immediately prior to a subprogram to ensure that the subprogram is not executed unless called. When a LightTools MACRO program is terminated with an END command, any files that were opened during the execution of the program are closed.

EXAMPLES:

```
1.  X = 100
    PRINT "X = ";X
    END
      MARINE:
        Z = X + Y
      RETURN

X = 100
```


END FUNCTION

DESCRIPTION:

Terminates a function definition which is initiated by the FUNCTION command.

TYPE:

Command (Program Control)

SYNTAX:

END FUNCTION

ARGUMENTS:

None.

RETURN VALUES:

None.

NOTES:

The END FUNCTION command is the last statement in a sequence of statements which defines a function. It is used in conjunction with the FUNCTION command that commences a sequence of statements which defines a function.

EXAMPLES:

```
1.  A = 3
    B = 2
    PRINT A,B,Z (A,B)
    END
        FUNCTION Z (X,Y)
            Z = X/Y
        END FUNCTION

3      2      1.5000000
```

END IF

DESCRIPTION:

Terminates an IF THEN block which is initiated by the IF THEN command.

TYPE:

Command (Program Control)

SYNTAX:

END IF

ARGUMENTS:

None.

RETURN VALUES:

None.

NOTES:

The END IF command is the last statement in a sequence of statements which constitutes an IF THEN block. It is used in conjunction with the IF THEN command that commences a sequence of statements which constitutes an IF THEN block.

EXAMPLES:

```
1.  X = 100
      IF X >= 100 THEN
      PRINT "X = ";X
      END IF

X = 100
```

END SUB

DESCRIPTION:

Terminates a subprogram which is initiated by the SUB command.

TYPE:

Command (Program Control)

SYNTAX:

END SUB

ARGUMENTS:

None.

RETURN VALUES:

None.

NOTES:

The END SUB command is the last statement in a sequence of statements which constitutes a subprogram. It is used in conjunction with the SUB command that commences the sequence of statements which constitutes the subprogram.

EXAMPLES:

```
1.  A = 3
    B = 2
    CALL MAXPLUS1 (A,B,C)
    PRINT A,B,C
    END
        SUB MAXPLUS1 (X,Y,Z)
            Z = MAX (X,Y) + 1
        END SUB
```

3 2 4

EOF

DESCRIPTION:

Returns the status of an open file with respect to the end of the file.

TYPE:

Function (File Input/Output)

SYNTAX:

EOF(*device-number*)

ARGUMENTS:

The *device-number* argument is the device number assigned to the file.

RETURN VALUES:

The status of the file.

NOTES:

The file is designated by a device number. The file must be open. The file is opened by the OPEN FOR APPEND[INPUT]OUTPUT AS # commands. Valid device numbers are 0 through 15. The *device-number* argument is not preceded by the pound (#) character in the EOF function.

The EOF function returns an integer status value which is indicative of the status of the file. The returned status value would be 1 if the file is at an end of file condition, or 0 if it is not. The returned status value must be assigned to a user-defined variable unless the function is used as an argument or in an expression.

EXAMPLES:

```
1.  FILE1$ = "myfile2.txt"
    FILENUM = 3
    OPEN FILE1$ FOR INPUT AS #FILENUM
        WHILE EOF(FILENUM) <> 1
            INPUT #FILENUM, I
            PRINT "DATUM IS "; I
        WEND
    CLOSE #FILENUM

    DATUM IS 3
    DATUM IS 2
    DATUM IS 4
```

Contents of myfile2.txt:

```
3
2
4
```

ERL

DESCRIPTION:

Returns the line number where the most recent error occurred.

TYPE:

Function (Miscellaneous)

SYNTAX:

ERL()

ARGUMENTS:

The () argument is the null character enclosed within parentheses.

RETURN VALUES:

The line number for the most recent error.

NOTES:

The prerequisite for the ERL function is that an error has occurred. This function can be used as an error handler for sorting out errors during the debugging process for a LightTools Macro program.

EXAMPLES:

```
1.  ON ERROR GOSUB MYERROR
    A = 2
    B = 0
    C = A/B
    PRINT "DONE"
    END

    MYERROR:
        PRINT "ERROR ";ERR;" OCCURRED ON LINE";ERL
    RETURN

ERROR 25 OCCURRED ON LINE 4
DONE
```

ERR

DESCRIPTION:

Returns the error number for the most recent error.

TYPE:

Function (Miscellaneous)

SYNTAX:

ERR()

ARGUMENTS:

The () argument is the null character enclosed within parentheses.

RETURN VALUES:

The error number of the most recent error.

NOTES:

The prerequisite for the ERR function is that an error has occurred. This function can be used as an error handler for sorting out errors during the debugging process for a LightTools Macro program.

EXAMPLES:

```
1.  ON ERROR GOSUB MYERROR
    A = 2
    B = 0
    C = A/B
    PRINT "DONE"
    END

    MYERROR:
        PRINT "ERROR ";ERR;" OCCURRED"
        RETURN

ERROR 25 OCCURRED
DONE
```

ERROR

DESCRIPTION:

Simulates an error and displays the associated error message.

TYPE:

Command (Miscellaneous)

SYNTAX:

`ERROR N`

ARGUMENTS:

The *N* argument is the error number.

RETURN VALUES:

None.

NOTES:

The ERROR command causes the program to behave as if the error occurred. It displays the error message corresponding to the error number.

The error number must be a whole number ranging from 0 to 67, inclusive. The range of values for the *N* argument is $-0.999... \leq N \leq 67.999...$. It can be a constant, variable, function, or expression. If the *N* argument is not a whole number, the decimal places in its value are truncated to produce a whole number.

EXAMPLES:

1. `ERROR 25`

ERROR (1): Divide by zero

EXIT DO

DESCRIPTION:

Exits a DO loop initiated by the DO or DO WHILE commands.

TYPE:

Command (Program Control)

SYNTAX:

EXIT DO

ARGUMENTS:

None.

RETURN VALUES:

None.

NOTES:

The EXIT DO command causes the program to terminate a DO loop prematurely and directs it to the statement immediately following the sequence of statements that define the DO loop.

EXAMPLES:

```
1.  X = 0
    DELTAX = 25
    DO
        X = X + DELTAX
        IF X >= 100 THEN
            EXIT DO
        END IF
        PRINT "X = ";X
    LOOP

    X = 25
    X = 50
    X = 75
```


EXIT FOR

DESCRIPTION:

Exits a FOR loop initiated by the FOR command.

TYPE:

Command (Program Control)

SYNTAX:

EXIT FOR

ARGUMENTS:

None.

RETURN VALUES:

None.

NOTES:

The EXIT FOR command causes the program to terminate a FOR loop prematurely and directs it to the statement immediately following the sequence of statements that define the FOR loop.

EXAMPLES:

```
1.  X = 0
    DELTAX = 25
    FOR I = 1 TO 10
      X = X + DELTAX
      IF X ≥ 100 THEN
        EXIT FOR
      END IF
      PRINT "X = ";X
    NEXT I

    X = 25
    X = 50
    X = 75
```

EXP

DESCRIPTION:

Returns the value of e raised to a power.

TYPE:

Function (Mathematical)

SYNTAX:

`EXP(X)`

ARGUMENTS:

The *X* argument is the power to which e is raised.

RETURN VALUES:

The value of e raised to the *X* argument.

NOTES:

The EXP function is the exponential value, which is the value of the base of the natural logarithm raised to a power. The base of the natural logarithm is the number $e = 2.7182818...$.

The *X* argument must be a real number. It can be a constant, variable, function, or expression. The range of values is $-\infty < X < \infty$.

The range of values for $\text{EXP}(X)$ is $0 < \text{EXP}(X) < \infty$. The returned value of EXP has the same precision as its argument. The EXP function is the inverse of the LOG function. For example, $X = \text{EXP}(\text{LOG}(X))$.

EXAMPLES:

```
1.  X = 2
    Y = EXP(X)
    PRINT X, Y

2          7.3890561
```

EXP10

DESCRIPTION:

Returns the value of 10 raised to a power.

TYPE:

Function (Mathematical)

SYNTAX:

EXP10(*X*)

ARGUMENTS:

The *X* argument is the power to which 10 is raised.

RETURN VALUES:

The value of 10 raised to the *X* argument.

NOTES:

The EXP10 function is the base of the common logarithm raised to a power. The base of the common logarithm is the number 10.

The *X* argument must be a real number. It can be a constant, variable, function, or expression. The range of values is $-\infty < X < \infty$.

The range of values for EXP10(*X*) is $0 < \text{EXP10}(X) < \infty$. The returned value of EXP10 has the same precision as its argument. The EXP10 function is the inverse of the LOG10 function. For example, $X = \text{EXP10}(\text{LOG10}(X))$.

EXAMPLES:

```
1.  X = 2
     Y = EXP10 (X)
     PRINT X,Y

     2          100
```

FLOOR

DESCRIPTION:

Returns the whole number with the largest value that is not greater than a number.

TYPE:

Function (Mathematical)

SYNTAX:

`FLOOR(X)`

ARGUMENTS:

The *X* argument is the number which determines the whole number.

RETURN VALUES:

The largest whole number that is not greater than the *X* argument.

NOTES:

The *X* argument must be a real number. It can be a constant, variable, function, or expression. The range of values is $-\infty < X < \infty$.

If the *X* argument is a whole number, then the value of `FLOOR(X)` is the *X* argument. The `FLOOR` function returns a whole number with the same precision as its argument.

EXAMPLES:

1.

```
X = 49.999
Y = FLOOR(X)
PRINT X,Y
```

49.9990000 49
2.

```
X = 50.000
Y = FLOOR(X)
PRINT X,Y
```

50 50

FOR

DESCRIPTION:

Initiates a FOR loop which is used to execute a sequence of program statements in a repetitive fashion using a counter index. The NEXT command is used in conjunction with the FOR command to repeat the FOR loop.

TYPE:

Command (Program Control)

SYNTAX:

FOR *counter* = *start* TO *finish* [STEP *increment*]

ARGUMENTS:

The *counter* argument is the counter index.

The *start* argument is the initial value of the counter index.

The *finish* argument is the final value of the counter index.

The *increment* argument is the step size of the counter index.

RETURN VALUES:

None.

NOTES:

The FOR command is used to begin a sequence of program statements that are executed in a repetitive fashion using a counter index. The counter index determines the number of times the sequence of program statements is executed. When the counter index exceeds its final value, the FOR loop is terminated. In addition, a STEP increment can be specified for the counter index to alternatively determine the number of times the sequence of program steps is executed. The STEP increment is optional. The default value is 1.

The FOR command is the first statement in a sequence of statements which constitutes a FOR loop. A FOR loop can be exited at any point by use of the EXIT FOR command. The NEXT command is the last statement in the sequence of program statements which constitutes a FOR loop. When the FOR loop is terminated, the program is directed to the statement immediately following the NEXT command.

EXAMPLES:

```
1.  X = 0
    DELTAX = 25
    FOR I = 1 TO 4
        X = X + DELTAX
        PRINT "X = ";X
    NEXT I
```

```
X = 25
X = 50
X = 75
X = 100
```

FUNCTION

DESCRIPTION:

Initiates a function definition block of statements. The END FUNCTION command is used in conjunction with the FUNCTION command to terminate the function definition.

TYPE:

Command (Program Control)

SYNTAX:

```
FUNCTION label$ [ ( parameter1 , ... ) ]
```

ARGUMENTS:

The *label*\$ argument is the name of the function.

The *parameterk* arguments are the parameters passed to the function.

RETURN VALUES:

The value assigned to the function name in the body of the function (see $Z = X/Y$ in the example, below).

NOTES:

The FUNCTION command is used to begin a sequence of program statements that define a function. The function can then be used in program statements in a manner similar to that of predefined functions. It can be evaluated by entering its label and associated input parameters in a program statement. The FUNCTION command is the first statement in a sequence of statements that define a function. The END FUNCTION command is the last statement in the sequence of statements that define a function.

The *parameterk* arguments are optional. If the function does not require input parameters, they are not required as arguments. Only the *label*\$ argument is required. The *parameterk* arguments are designated as local variables. The values assigned to these variables are only used in the evaluation of the defined function. The variables can be assigned numeric or string values. The values can be constants, variables, functions, or expressions. Constant string values must be enclosed within double quotation marks (e.g., "ABCDE"). The *parameterk* arguments must be delimited by commas. The k character in the *parameterk* arguments is a symbolic index that corresponds to the numerical order of the parameters entered as arguments.

EXAMPLES:

```
1.  A = 3
    B = 2
    PRINT A, B, Z (A, B)
    END
        FUNCTION Z (X, Y)
            Z = X/Y
        END FUNCTION

3      2      1.5000000
```

GOSUB/GO SUB

DESCRIPTION:

Directs the program to a subprogram designated by a label or line number.

TYPE:

Command (Program Control)

SYNTAX:

GOSUB *label\$* | *line* or GO SUB *label\$* | *line*

ARGUMENTS:

The *label\$* argument is the label to which the program is directed.

The *line* argument is the line number to which the program is directed.

RETURN VALUES:

None.

NOTES:

The GOSUB command can be entered as one or two keywords (i.e., GOSUB or GO SUB). It directs the program to a statement which begins with a label. The *label\$* argument used in the GOSUB command is a string. The label in the statement that the program is directed to by the GOSUB command is the same string with a trailing colon. The *line* argument used in the GOSUB command is a number. The *label\$* and *line* arguments are mutually exclusive. Only one of these arguments can be used in a single GOSUB command to direct the program.

The sequence of statements which constitutes the subprogram that the program is directed to must have a RETURN command as the last statement. When the RETURN command is executed, the program is directed to the statement immediately following the GOSUB command. If multiple GOSUB commands have been executed prior to encountering the RETURN command, the program is directed to the statement immediately following the most recently executed GOSUB command when the RETURN command is executed.

EXAMPLES:

```
1.  X = 1
    Y = 2
    GOSUB MARINE
    PRINT X,Y,Z
    GO SUB SANDWICH
    PRINT X,Y,Z
    END

    MARINE:
        Z = X/Y
    RETURN
    SANDWICH:
        Z = X*Y
    RETURN
```

1	2	0.5000000
1	2	2

GOTO/GO TO

DESCRIPTION:

Directs the program to a statement designated by a label or line number.

TYPE:

Command (Program Control)

SYNTAX:

GOTO *label\$* | *line* or GO TO *label\$* | *line*

ARGUMENTS:

The *label\$* argument is the label to which the program is directed.

The *line* argument is the line number to which the program is directed.

RETURN VALUES:

None.

NOTES:

The GOTO command can be entered as one or two keywords (i.e., GOTO or GO TO). The *label\$* argument used in the GOTO command is a string. It must be a constant. The label that begins the statement that the program is directed to by the GOTO command is the same string with a trailing colon. The *line* argument used in the GOTO command is a number. The *label\$* and *line* arguments are mutually exclusive. Only one of these arguments can be used in a single GOTO command to direct the program.

EXAMPLES:

```
1.  1 X = 0
    2     Y = 0
    3     X = X + 1
    4     Y = Y + 2
    5     GOTO PROCESS
    6     PRINT X,Y,Z
    7     GO TO 3
    8     PROCESS:
    9         IF X <= 3 THEN
10             Z = X/Y
11         ELSE
12             Z = X*Y
13         END IF
14         IF X > 6 THEN
15             PRINT "PROCESS COMPLETE"
16         END IF
17         IF X > 6 THEN
18             END
19         END IF
20     GO TO 6

1      2      0.5000000
2      4      0.5000000
3      6      0.5000000
4      8      32
5     10      50
6     12      72
PROCESS COMPLETE
```

IF THEN

DESCRIPTION:

Initiates an IF THEN block which is used to evaluate an expression and, if it is true, performs an operation designated by the THEN keyword. The END IF command is used in conjunction with the IF THEN command to terminate the IF THEN block.

TYPE:

Command (Program Control)

SYNTAX:

```
IF expression1 THEN [ statement1 ]  
[ ELSE | ELSEIF expression2 THEN [ statement2 ] ]
```

ARGUMENTS:

The *expression1* and *expression2* arguments are the expressions that are evaluated.

The *statement1* and *statement2* arguments are the operations that are performed.

RETURN VALUES:

None.

NOTES:

The IF THEN command is used to evaluate an expression and, if it is true, performs an operation designated by the THEN keyword. If the expression is false, the program can be directed to perform an operation designated by the ELSE command. Alternatively, a second expression can be introduced and evaluated by the ELSE IF THEN command. This second evaluation is performed upon the determination that the first expression is false. If the second expression is true, an operation designated by the THEN component of the ELSE IF THEN command is performed. If the second expression is false, the program can be directed to perform an operation designated by the ELSE command. The END IF command is used in conjunction with the IF THEN command to terminate the IF THEN block. The program is then directed to the statement immediately following the END IF command.

The ELSE and ELSEIF THEN commands are optional. If they are not specified, the program is directed to the statement immediately following the IF THEN command if *expression1* is false. Only the *expression1* argument is required. The *expression2* argument is only used with the ELSE IF THEN command. The *statement1* and *statement2* arguments are also optional. If the *statement1* argument is not specified, then the program is directed to the statement immediately following the IF THEN command if the *expression1* argument is true. If the *statement2* argument is not specified, then the program is directed to the statement immediately following the ELSE IF THEN command if the *expression2* argument is true.

EXAMPLES:

```
1.  FOR I = 0 TO 4
    J = I - 2
    IF J > 0 THEN
        PRINT "I = ";I,"J is greater than zero."
    ELSEIF J < 0 THEN
        PRINT "I = ";I,"J is less than zero."
    ELSE
        PRINT "I = ";I,"J is equal to zero."
    END IF
NEXT I
END

I = 0    J is less than zero.
I = 1    J is less than zero.
I = 2    J is equal to zero.
I = 3    J is greater than zero.
I = 4    J is greater than zero.
```

INPUT

DESCRIPTION:

Reads values from a LightTools dialog box and assigns them to variables.

TYPE:

Command (Interactive Input/Output)

SYNTAX:

```
INPUT [ prompt-string$ ; ] V1 [ , V2 ... ]
```

ARGUMENTS:

The *prompt-string*\$ argument is the message to be displayed in a LightTools dialog box.

The *Vk* arguments are the variables to which the values entered in a LightTools dialog box are assigned.

RETURN VALUES:

None.

NOTES:

The INPUT command is used to assign values to variables used in the program. The values are entered by way of a LightTools dialog box in which the values are specified manually with the computer terminal. A message describing the input values can be displayed in the dialog box, in order to prompt the program user for the desired input.

The *prompt-string*\$ argument is optional. It must be a string. It can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). The *prompt-string*\$ argument must be delimited by a semicolon. Only the *V1* argument is required. The other *Vk* arguments are optional. The *Vk* arguments can be numeric or string variables. The values must be constants. Constant string values must be enclosed within double quotation marks (e.g., "ABCDE"). The variable type must match the value type. The *Vk* arguments must be delimited by commas. The k character in the *Vk* arguments is a symbolic index that corresponds to the numerical order of the variables entered as arguments.

EXAMPLES:

```
1.  INPUT "Enter X and Y";X,Y
    PRINT "X = ";X,"Y = ";Y
```

```
Enter X and Y      {This is the output in a LightTools dialog box.
Enter the values 1,2 for X and Y in the
LightTools dialog box and click Apply.}
```

```
X = 1   Y = 2
```

INPUT

DESCRIPTION:

Reads values from a file designated by a device number and assigns them to variables.

TYPE:

Command (File Input/Output)

SYNTAX:

INPUT #*device-number* , *V1* [, *V2* ...]

ARGUMENTS:

The *device-number* argument is the device number assigned to the file.

The *Vk* arguments are the variables to which the values are assigned.

RETURN VALUES:

None.

NOTES:

The INPUT # command must have a space between the INPUT keyword and the pound (#) character. The file designated by the *device-number* argument must be opened with the OPEN FOR INPUT AS # command. A valid device number specification is a number preceded by the pound (#) character. Valid device numbers are 0 through 15. Therefore, a maximum of 16 files can be open at any one time.

Only the *device-number* and *V1* arguments are required. The other *Vk* arguments are optional. The *Vk* arguments can be numeric or string variables. The values must be constants. Constant string values must be enclosed within double quotation marks (e.g., "ABCDE"). The variable type must match the value type. All arguments must be delimited by commas. The k character in the *Vk* arguments is a symbolic index that corresponds to the numerical order of the variables entered as arguments.

The INPUT # command is generally used to read data that has been written to a file with the WRITE # command. When the INPUT # command is executed, the program searches for the first character in the file designated by the *device-number* argument that is not a blank space.

EXAMPLES:

```
1.  OPEN "myfile3.txt" FOR INPUT AS #15
      INPUT #15,X,Y,Z
      PRINT X,Y,Z
```

1	2	3
<u>Contents of myfile3.txt:</u>		
1		
2		
3		

INSTR

DESCRIPTION:

Returns the position at which a string pattern occurs within a string.

TYPE:

Function (String)

SYNTAX:

INSTR([*N* ,] *string-searched\$* , *string-pattern\$*)

ARGUMENTS:

The *N* argument is the position at which the search begins.

The *string-searched\$* argument is the string within which the search is performed.

The *string-pattern\$* argument is the string pattern for which the search is performed.

RETURN VALUES:

The position at which the *string-pattern\$* argument is found.

NOTES:

The string can be searched beginning at a designated position. This position is specified by the *N* argument. The *N* argument is optional. Only the *string-searched\$* and *string-pattern\$* arguments are required. The starting position for the search must be a whole number ranging from 1 to 255, inclusive. The range of values for the *N* argument is $1 \leq N \leq 255$. It can be a constant, variable, function, or expression. If the *N* argument is not a whole number, the decimal places in its value are truncated to produce a whole number.

The *string-searched\$* and *string-pattern\$* arguments must be defined strings. They can be constants, variables, functions, or expressions. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE").

If the *string-pattern\$* argument is not found in the *string-searched\$* argument, then the INSTR function returns a value of 0.

If the *string-searched\$* argument is the null string (""), then the INSTR function returns a value of 0. If the *N* argument has been specified, then the INSTR function returns a value of 0, also.

If the *string-pattern\$* argument is the null string (""), and the *string-searched\$* argument is not the null string, then the INSTR function returns a value of 1. If the *N* argument has been specified, then the INSTR function returns a value equal to the *N* argument.

If the *N* argument is greater than the length of the *string-searched\$* argument, then the INSTR function returns a value of 0.

EXAMPLES:

1. S\$ = "ABCDE"
T\$ = "C"
X = INSTR(S\$,T\$)
PRINT S\$,T\$,X

ABCDE C 3

2. S\$ = "ABCDE"
T\$ = "F"
X = INSTR(S\$,T\$)
PRINT S\$,T\$,X

ABCDE F 0

3. S\$ = "ABCDE"
T\$ = ""
X = INSTR(3,S\$,T\$)
PRINT S\$,T\$,X

ABCDE 3

4. S\$ = "ABCDEABCDE"
T\$ = "C"
X = INSTR(4,S\$,T\$)
PRINT S\$,T\$,X

ABCDEABCDE C 8

INT

DESCRIPTION:

Returns the whole number with the largest value that is less than or equal to a number.

TYPE:

Function (Mathematical)

SYNTAX:

`INT(X)`

ARGUMENTS:

The *X* argument is the number for which the largest whole number is returned.

RETURN VALUES:

The whole number with the largest value that is less than or equal to the *X* argument.

NOTES:

The *X* argument must be a real number. The range of values is $-\infty < X < \infty$. It can be a constant, variable, function, or expression.

If the *X* argument is a whole number, then the value of `INT(X)` is the *X* argument. The INT function returns a whole number with the same precision as its argument.

EXAMPLES:

```
1.  X = 49.999
    Y = INT(X)
    PRINT X, Y

    49.9990000    49

2.  X = 49.000
    Y = INT(X)
    PRINT X, Y

    49          49
```

LEFT\$

DESCRIPTION:

Returns a substring composed of the leftmost characters of a string.

TYPE:

Function (String)

SYNTAX:

LEFT\$(*\$\$*, *N*)

ARGUMENTS:

The *\$\$* argument is the string from which the substring is returned.

The *N* argument is the number of characters in the substring.

RETURN VALUES:

A substring composed of the first *N* characters of the *\$\$* argument.

NOTES:

The LEFT\$ function creates a new string, and the string argument remains intact. The substring is composed of a set of characters starting at the left side (or beginning) of the string. The LEFT\$ function is only used with strings. The LEFT\$ function is analogous to the RIGHT\$ and MID\$ functions.

The *\$\$* argument can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE").

The number of characters must be a whole number. The range of values for the *N* argument is

$-0.999... \leq N \leq 255.999...$. It can be a constant, variable, function, or expression. If the *N* argument is not a whole number, the decimal places in its value are truncated to produce a whole number. If the *N* argument is 0, the null string ("") is returned. If the *N* argument is greater than the number of characters in the *\$\$* argument, the entire string is returned.

EXAMPLES:

```
1.  S$ = "ABCDE"
    T$ = LEFT$(S$, 3)
    PRINT S$, T$
```

ABCDE ABC

```
2.  S$ = "ABCDE"
    T$ = LEFT$(S$, 6)
    PRINT S$, T$
```

ABCDE ABCDE

LEN

DESCRIPTION:

Returns the length of a string as the number of characters.

TYPE:

Function (String)

SYNTAX:

LEN(*\$\$*)

ARGUMENTS:

The *\$\$* argument is the string whose length is returned.

RETURN VALUES:

The length of the *\$\$* argument.

NOTES:

The LEN function is only used with string arguments. The strings can be constants, variables, functions, or expressions. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). The length of the string is expressed in terms of the number of characters in the string. All characters are counted, including blank spaces and control characters.

EXAMPLES:

```
1.  S$ = "ABCDE"
    X = LEN(S$)
    PRINT S$,X
```

ABCDE 5

```
2.  S$ = " ABCDE ABCDE "
    X = LEN(S$)
    PRINT S$,X
```

ABCDE ABCDE 13

LET

DESCRIPTION:

Assigns numeric or string values to a variable.

TYPE:

Command (Program Control)

SYNTAX:

[LET] *V* = *value*

ARGUMENTS:

The *V* argument is the variable to which the value is assigned.

The *value* argument is the numeric or string value which is assigned to the variable.

RETURN VALUES:

None.

NOTES:

The LET keyword is optional. A value can be assigned to a variable in the form of an equation (e.g., $X = 2$).

The *V* argument can be a numeric or string variable. Its type must match that of the value assigned. The values can be constants, variables, functions, or expressions. Constant string values must be enclosed within double quotation marks (e.g., "ABCDE"). Only one type of argument can be used in a LET command. An attempt to assign a *value* argument to a *V* argument whose types do not match results in an error.

EXAMPLES:

1. LET X = 25.52
 LET S\$ = "ABCDE"
 PRINT X, S\$

 25.5200000 ABCDE
2. X = 25.52
 S\$ = "ABCDE"
 PRINT X, S\$

 25.5200000 ABCDE

LINE INPUT

DESCRIPTION:

Reads an entire line from a LightTools dialog box and assigns it to a string variable.

TYPE:

Command (Interactive Input/Output)

SYNTAX:

LINE INPUT [*prompt-string* ;] *SS*

ARGUMENTS:

The *prompt-string* argument is the message displayed in a LightTools dialog box.

The *SS* argument is the string variable to which the line entered in a LightTools dialog box is assigned.

RETURN VALUES:

None.

NOTES:

The line is entered by way of a LightTools dialog box in which the text is specified manually with the computer terminal. A message describing the input can be displayed in the dialog box, in order to prompt the program user for the desired input. The message is specified with the LINE INPUT command.

The *prompt-string* argument is optional. It must be a string. It can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). The *prompt-string* argument must be delimited by a semicolon. The *SS* argument must be a string variable.

EXAMPLES:

```
1.  LINE INPUT "Enter title";TITLE$  
    PRINT TITLE$
```

```
Enter Title      {This is the output in a LightTools dialog box.  
                  Enter the title LightTools Lens in the  
                  LightTools dialog box and click Apply.}
```

```
LightToolsLens
```

LINE INPUT

DESCRIPTION:

Reads an entire line from a file designated by a device number and assigns it to a string variable.

TYPE:

Command (File Input/Output)

SYNTAX:

LINE INPUT [#]*device-number* , *S\$*

ARGUMENTS:

The *device-number* argument is the device number assigned to the file.

The *S\$* argument is the string variable to which the line is assigned.

RETURN VALUES:

None.

NOTES:

The LINE INPUT # command must have a space between the LINE INPUT keywords and the pound (#) character. The pound (#) character is optional. The file designated by the *device-number* argument must be opened with the OPEN FOR INPUT AS # command. Valid device numbers are 0 through 15. Therefore, a maximum of 16 files can be open at any one time. The LINE INPUT # command is generally used to read data that has been written with the WRITE # command.

When the LINE INPUT # command is executed, the program reads characters continuously until it encounters a carriage return, an arrow character (indicates the end of the file), or until 255 characters have been read. All of the characters that are read are assigned to the *S\$* argument. The *S\$* argument must be a string variable. Input for any subsequent LINE INPUT # or INPUT # commands reading the same file begins with the first character following the carriage return and its associated line feed or with the 256th character.

EXAMPLES:

```
1.  OPEN "myfile4.txt" FOR INPUT AS #15
    LINE INPUT #15, S$
    PRINT S$
```

ABCDE XYZ

Contents of myfile4.txt:

ABCDE XYZ

LOG

DESCRIPTION:

Returns the natural logarithm of a number.

TYPE:

Function (Mathematical)

SYNTAX:

$\text{LOG}(X)$

ARGUMENTS:

The X argument is the number for which the natural logarithm is returned.

RETURN VALUES:

The natural logarithm of the X argument.

NOTES:

The natural logarithm of a number is the power that the exponential number $e = 2.7182818...$ would have to be raised to in order to produce the number that is the argument of the LOG function.

The X argument must be a real number. It can be a constant, variable, function, or expression. The range of values is $0 < X < \infty$.

The range of values for $\text{LOG}(X)$ is $-\infty < \text{LOG}(X) < \infty$. The returned value of LOG has the same precision as its argument. The LOG function is the inverse of the EXP function. For example, $X = \text{LOG}(\text{EXP}(X))$.

EXAMPLES:

```
1.  X = 2
    Y = LOG(X)
    PRINT X, Y

2      0.6931472
```


LOG10

DESCRIPTION:

Returns the common logarithm of a number.

TYPE:

Function (Mathematical)

SYNTAX:

LOG10(*X*)

ARGUMENTS:

The *X* argument is the number for which the common logarithm is returned.

RETURN VALUES:

The common logarithm of the *X* argument.

NOTES:

The common logarithm of a number is the power that the number 10 would have to be raised to in order to produce the number that is the argument of the LOG10 function.

The *X* argument must be a real number. It can be a constant, variable, function, or expression. The range of values is $0 < X < \infty$.

The range of values for LOG10(*X*) is $-\infty < \text{LOG10}(X) < \infty$. The returned value of LOG10 has the same precision as its argument. The LOG10 function is the inverse of the EXP10 function. For example, $X = \text{LOG10}(\text{EXP10}(X))$.

EXAMPLES:

```
1.  X = 2
    Y = LOG10 (X)
    PRINT X,Y

2          0.3010300
```

LOOP

DESCRIPTION:

Repeats a DO or DO WHILE loop initiated by the DO or DO WHILE commands.

TYPE:

Command (Program Control)

SYNTAX:

LOOP

ARGUMENTS:

None.

RETURN VALUES:

None.

NOTES:

The LOOP command is used in conjunction with the DO or DO WHILE commands to repeat a DO or DO WHILE loop. It is the last statement in a sequence of statements which constitutes a DO or DO WHILE loop. When a DO or DO WHILE loop is terminated, the program is directed to the statement immediately following the LOOP command.

EXAMPLES:

```
1.  X = 0
    DELTAX = 25
    DO
        X = X + DELTAX
        IF X > 100 THEN
            EXIT DO
        END IF
        PRINT "X = ";X
    LOOP
```

```
X = 25
X = 50
X = 75
X = 100
```

```
2.  X = 0
    DELTAX = 25
    DO WHILE X < 100
        X = X + DELTAX
        PRINT "X = ";X
    LOOP
```

```
X = 25
X = 50
X = 75
X = 100
```

LOOP UNTIL

DESCRIPTION:

Repeats a DO loop initiated by the DO command until a defined criterion is met.

TYPE:

Command (Program Control)

SYNTAX:

LOOP UNTIL *criterion*

ARGUMENTS:

The *criterion* argument is the criterion against which the loop test results are measured.

RETURN VALUES:

None.

NOTES:

The criterion is entered as an argument on the command line. Each time a statement containing the item for which the criterion was defined is encountered, a test is made to determine if the criterion has been met. If it has not been met, the sequence of program statements which constitutes the DO loop is repeated. If it has been met, the DO loop is terminated.

The LOOP UNTIL command is used in conjunction with the DO command to repeat a DO loop. The LOOP UNTIL command is the last statement in a sequence of statements which constitutes a DO loop. When a DO loop is terminated, the program is directed to the statement immediately following the LOOP UNTIL command.

EXAMPLES:

```
1.  X = 0
    DELTAX = 25
    DO
        X = X + DELTAX
        PRINT "X = ";X
    LOOP UNTIL  X >= 100
```

```
X = 25
X = 50
X = 75
X = 100
```

LTBSET

DESCRIPTION:

Sets the status of control parameters that affect the execution of a LightTools MACRO program.

TYPE:

Command (LightTools MACRO)

SYNTAX:

LTBSET DIALOG | ERRORECHO | INTERRUPT | MAXERRORS | UPDATE *value*

ARGUMENTS:

The *value* argument is the status to which the control parameter is set.

RETURN VALUES:

None.

NOTES:

Valid parameters and their *value* arguments are:

DIALOG	This parameter is used to enable or disable the display of the LightTools dialog boxes during execution of a LightTools MACRO program. The status is designated by the <i>value</i> argument. A value of 1 enables the display; a value of 0 disables the display. The default value is 1. The dialog box display status is enabled upon completion of the macro program.
ERRORECHO	This parameter is used to enable or disable the echo of statements which produce an error message. If this option is enabled, the statement which produces the error message is printed in the LightTools Console window following the error message. A value of 1 enables the echo; a value of 0 disables the echo. The default value is 0.
INTERRUPT	This parameter is used to specify the number of statements that are to be executed between checks for a user interrupt. The number of statements is designated by the <i>value</i> argument. The default value is 50.
LOOPINT	<p>This parameter determines how often the LightTools MACRO subsystem checks for a user interrupt. It has a value of 1 (on) or 0 (off). When on, the LightTools MACRO subsystem checks for a user interrupt at the bottom of each loop. When off, interrupt checking occurs only when the number of commands processed is a multiple of the value specified with the INTERRUPT parameter. The default value is 1.</p> <p>Note: You can set LOOPINT to 0 (off) to improve macro processing speed; however, a macro will be less responsive in this mode if it is interrupted.</p>
MAXERRORS	This parameter is used to stop execution of a LightTools MACRO program after a designated number of errors. The number of errors is designated by the <i>value</i> argument. The default value is 100.

- DBUPDATE** This parameter is used to enable or disable LightTools database updates during execution of a LightTools MACRO program. The status is designated by the *value* argument. A value of 1 enables the update; a value of 0 disables the update. The default value is 1. The LightTools database update status is retained upon completion of the macro program.
- VIEWUPDATE** This parameter is used to enable or disable LightTools graphic updates during execution of a LightTools MACRO program. The status is designated by the *value* argument. A value of 1 enables the update; a value of 0 disables the update. The default value is 1. The LightTools graphic update status is not retained upon completion of the macro program.

EXAMPLES:

1.

```
UpdateStatus = LTBSET("DIALOG", 0)
IF (UpdateStatus = 0) THEN
    PRINT "Dialog boxes will not appear while this macro is"
    PRINT "running."
END IF
```
2.

```
IF (LTBSET("INTERRUPT", 1000) <> 0) THEN
    PRINT "There was an error changing the interrupt check"
    PRINT "interval to 1000."
ELSE
    PRINT "Interrupt interval successfully changed to 1000"
END IF
```
3.

```
IF (LTBSET("LOOPINT", 0) = 0) THEN
    PRINT "Interrupt checking at the bottom of loops has"
    PRINT "been disabled."
    PRINT "The macro will perform better, but may not be as"
    PRINT "responsive to user interrupts."
END IF
```

LTBSET

DESCRIPTION:

Returns the status of the setting of the status of control parameters that affect the execution of a LightTools MACRO program.

TYPE:

Function (LightTools MACRO)

SYNTAX:

LTBSET("DIALOG | ERRORECHO | INTERRUPT | MAXERRORS | UPDATE" , *value*)

ARGUMENTS:

The *value* argument is the status to which the control parameter is set.

RETURN VALUES:

The status of the setting of the control parameter.

NOTES:

Valid parameters and their *value* arguments are:

DIALOG	This parameter is used to enable or disable the display of the LightTools dialog boxes during execution of a LightTools MACRO program. The status is designated by the <i>value</i> argument. A value of 1 enables the display; a value of 0 disables the display. The default value is 1. The dialog box display status is enabled upon completion of the macro program.
ERRORECHO	This parameter is used to enable or disable the echo of statements which produce an error message. If this option is enabled, the statement which produces the error message is printed in the LightTools Console window following the error message. A value of 1 enables the echo; a value of 0 disables the echo. The default value is 0.
INTERRUPT	This parameter is used to specify the number of statements that are to be executed between checks for a user interrupt. The number of statements is designated by the <i>value</i> argument. The default value is 50.
MAXERRORS	This parameter is used to stop execution of a LightTools MACRO program after a designated number of errors. The number of errors is designated by the <i>value</i> argument. The default value is 100.
DBUPDATE	This parameter is used to enable or disable LightTools database updates during execution of a LightTools MACRO program. The status is designated by the <i>value</i> argument. A value of 1 enables the update; a value of 0 disables the update. The default value is 1. The LightTools database update status is retained upon completion of the macro program.

VIEWUPDATE This parameter is used to enable or disable LightTools graphic updates during execution of a LightTools MACRO program. The status is designated by the *value* argument. A value of 1 enables the update; a value of 0 disables the update. The default value is 1. The LightTools graphics update status is retained upon completion of the macro program.

The LTBSET function returns an integer status value which is indicative of the setting of the control parameter. The returned status value would be 0 if the setting of the control parameter occurred without error, or greater than zero if the setting of the control parameter produced an error. The returned status value must be assigned to a user-defined variable unless the function is used as an argument or in an expression.

EXAMPLES:

1. UpdateStatus = LTBSET("UPDATE",1)
2. IF LTBSET("MAXERRORS",2) = 0 THEN
 PRINT "The maximum number of allowed errors is 2."
 END IF

The maximum number of allowed errors is 2.

3. IF LTBSET("ERRORECHO",1) <> 0 THEN
 PRINT "The error echo has not been turned on."
 END IF

The error echo has not been turned on.

LTCHECKVAR

DESCRIPTION:

Returns the type of a LightTools variable.

TYPE:

Function (LightTools MACRO)

SYNTAX:

LTCHECKVAR(*V\$*)

ARGUMENTS:

The *V\$* argument is the LightTools variable whose type is returned.

RETURN VALUES:

The type of the LightTools variable.

NOTES:

The *V\$* argument must be a string. It can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "XYZ"). LightTools variables can be defined as one of three types: integer, real, or string. The returned values of the LTCHECKVAR function are as follows:

0 - Undefined

1 - Integer

2 - Real

3 - String

EXAMPLES:

```
1.  LTCMD "Define XYZ=123"
    VariableType = LTCHECKVAR("XYZ")
    XYZ = LTGETVAR("XYZ")
    PRINT "XYZ = ";XYZ,"The variable type for XYZ is ";VariableType
```

XYZ = 123 The variable type for XYZ is 2

```
2.  VariableType = LTCHECKVAR("ABC")
    IF VariableType = 0 THEN
    PRINT "The variable ABC is undefined."
    END IF
```

The variable ABC is undefined.

LTCMD

DESCRIPTION:

Executes a LightTools command within a LightTools MACRO program.

TYPE:

Command (LightTools MACRO)

SYNTAX:

LTCMD *command*\$

ARGUMENTS:

The *command*\$ argument is the LightTools command to be executed.

RETURN VALUES:

None.

NOTES:

The LightTools command that is designated by the *command*\$ argument is executed by the LightTools command interpreter. The *command*\$ argument must be a quoted string. It can be composed of constants, variables, functions, or expressions. Constant string arguments must be enclosed within double quotation marks (e.g., "Define X=1"). The functions LTCOORD2\$, LTCOORD3\$, and LTSTR\$ return quoted strings suitable for use as an argument for the LTCMD command.

The entire set of LightTools commands are executable within a LightTools MACRO program with the LTCMD command. The entire set of LightTools commands and their functionality and syntax can be found in the *LightTools Command Reference Guide*, which is an online document provided with the LightTools software.

The LTCMD command also has the capability to execute LightTools script files (.ltc). The name of the script file is used as the *command*\$ argument. For example, the script file LightTools.1.ltc can be executed by using the command LTCMD "LightTools" in a macro program.

EXAMPLES:

1. LTCMD "Union"
2. LTCMD "More " + lenslets\$
3. LTCMD LTCOORD3\$(0,0,0)

LTCMD

DESCRIPTION:

Returns the status of a LightTools command executed within a LightTools MACRO program.

TYPE:

Function (LightTools MACRO)

SYNTAX:

LTCMD(*command\$*)

ARGUMENTS:

The *command\$* argument is the LightTools command to be executed.

RETURN VALUES:

The status of the LightTools command executed.

NOTES:

The LightTools command that is designated by the *command\$* argument is executed by the LightTools command interpreter. The *command\$* argument must be a quoted string. It can be composed of constants, variables, functions, or expressions. Constant string arguments must be enclosed within double quotation marks (e.g., "Define X=1"). The functions LTCOORD2\$, LTCOORD3\$, and LTSTR\$ return quoted strings suitable for use as an argument for the LTCMD command.

The LTCMD function returns an integer status value which is indicative of the execution of the LightTools command. The returned status value would be 0 if the execution of the command occurred without error, or greater than zero if the execution of the command produced an error. The returned status value must be assigned to a user-defined variable unless the function is used as an argument or in an expression.

The entire set of LightTools commands are executable within a LightTools MACRO program with the LTCMD command. The entire set of LightTools commands and their functionality and syntax can be found in the *LightTools Command Reference Guide*, which is an online document provided with the LightTools software.

The LTCMD function also has the capability to execute LightTools script files (.ltc). The name of the script file is used as the *command\$* argument. For example, the script file LightTools.1.ltc can be executed by using the function LTCMD("LightTools") in a macro program.

EXAMPLES:

1. CommandStatus = LTCMD("Union")
2. IF LTCMD("More " + lenslets\$) <> 0 THEN
PRINT "The object to be selected does not exist."
END IF
3. PositionStatus = LTCMD(LTCOORD3\$(0,0,0))

LTCOORD2\$

DESCRIPTION:

Returns a formatted string of two dimensional coordinates.

TYPE:

Function (LightTools MACRO)

SYNTAX:

LTCOORD2\$(*X1* , *X2*)

ARGUMENTS:

The *X1* argument is the first coordinate.

The *X2* argument is the second coordinate.

RETURN VALUES:

The string as a two dimensional set of coordinates.

NOTES:

The LTCOORD2\$ function can be used to specify input coordinates for the 2D Design view in LightTools and the LightTools commands that require two dimensional input coordinates. The two dimensional set of coordinates is a string that is formatted in a manner that is appropriate for input in a two dimensional coordinate system.

EXAMPLES:

1. LTCMD LTCOORD2\$(0,0)
2. LTCMD LTCOORD2\$(X1,Y1)

LTCOORD3\$

DESCRIPTION:

Returns a formatted string of three dimensional coordinates.

TYPE:

Function (LightTools MACRO)

SYNTAX:

LTCOORD3\$(*X1* , *X2* , *X3*)

ARGUMENTS:

The *X1* argument is the first coordinate.

The *X2* argument is the second coordinate.

The *X3* argument is the third coordinate.

RETURN VALUES:

The string as a three dimensional set of coordinates.

NOTES:

The LTCOORD3\$ function can be used to specify input coordinates for the 3D Design view in LightTools and the LightTools commands that require three dimensional input coordinates. The three dimensional set of coordinates is a string that is formatted in a manner that is appropriate for input in a three dimensional coordinate system.

EXAMPLES:

1. LTCMD LTCOORD3\$(0,0,0)
2. LTCMD LTCOORD3\$(Solid1X,Solid1Y,Solid1Z)

LTDBGET

DESCRIPTION:

Returns the data value of a numeric attribute corresponding to an object key and data element.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBGET(*objectkey\$* , *data_element\$*)

ARGUMENTS:

The *objectkey\$* argument is the object key for the attribute.

The *data_element\$* argument is the data element for the attribute.

RETURN VALUES:

The data value of the attribute.

NOTES:

The LTDBGET function can be used to assign the numeric data value of a specific LightTools system attribute to a numeric variable. The attribute is specified with *objectkey\$* and *data_element\$* arguments. Constant *objectkey\$* and *data_element\$* arguments must be enclosed within double quotation marks (e.g., "COMPONENTS[1].SOLID[1]" ; "ALPHA").

EXAMPLES:

1. SolidList\$ = LTDBLIST\$("COMPONENTS[1]" , "SOLID")
Solid1\$ = LTLISTNEXT\$(SolidList\$)
Solid1Alpha = LTDBGET(Solid1\$, "Alpha")
2. Solid1X = LTDBGET(Solid1\$, "X")
Solid1Y = LTDBGET(Solid1\$, "Y")
Solid1Z = LTDBGET(Solid1\$, "Z")
Solid2\$ = LTLISTNEXT\$(SolidList\$)
LTCMD "Select " + LTSTR\$(LTDBGET\$(Solid2\$, "Name"))
LTCMD "Move " + LTCOORD3\$(Solid1X, Solid1Y, Solid1Z)

LTDBGET\$

DESCRIPTION:

Returns the data value of a string attribute corresponding to an object key and data element.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBGET\$(*objectkey\$* , *data_element\$*)

ARGUMENTS:

The *objectkey\$* argument is the object key for the attribute.

The *data_element\$* argument is the data element for the attribute.

RETURN VALUES:

The data value of the attribute.

NOTES:

The LTDBGET\$ function can be used to assign the string data value of a specific LightTools system attribute to a string variable. The attribute is specified with *objectkey\$* and *data_element\$* arguments. Constant *objectkey\$* and *data_element\$* arguments must be enclosed within double quotation marks (e.g., "COMPONENTS[1].SOLID[1]" ; "ALPHA").

EXAMPLES:

1. SolidList\$ = LTDBLIST\$("COMPONENTS[1]" , "SOLID")
Solid1\$ = LTLISTNEXT\$(SolidList\$)
Solid1Name\$ = LTDBGET\$(Solid1\$, "Name")
2. LTCMD "Select " + LTSTR\$(LTDBGET\$(Solid1\$, "Name"))

LTDBGETI

DESCRIPTION:

Returns the data value of a numeric attribute in a one dimensional array corresponding to an object key, data element, and index number.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBGETI(*objectkey\$* , *data_element\$* , *index*)

ARGUMENTS:

The *objectkey\$* argument is the object key for the attribute.

The *data_element\$* argument is the data element for the attribute.

The *index* argument is the array index number for the attribute.

RETURN VALUES:

The data value of the attribute.

NOTES:

The LTDBGETI function can be used to assign the numeric data value of a specific LightTools system attribute to a numeric variable. The attribute is a component of a one dimensional array. It is specified with *objectkey\$*, *data_element\$*, and *index* arguments. Constant *objectkey\$* and *data_element\$* arguments must be enclosed within double quotation marks (e.g., "COMPONENTS[1].SOLID[1]" ; "ALPHA").

LTDBGETIS

DESCRIPTION:

Returns the data value of a string attribute in a one dimensional array corresponding to an object key, data element, and index number.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBGETIS(*objectkey\$* , *data_element\$* , *index*)

ARGUMENTS:

The *objectkey\$* argument is the object key for the attribute.

The *data_element\$* argument is the data element for the attribute.

The *index* argument is the array index number for the attribute.

RETURN VALUES:

The data value of the attribute.

NOTES:

The LTDBGETIS function can be used to assign the string data value of a specific LightTools system attribute to a string variable. The attribute is a component of a one dimensional array. The attribute is specified with *objectkey\$*, *data_element\$*, and *index* arguments. Constant *objectkey\$* and *data_element\$* arguments must be enclosed within double quotation marks (e.g., "COMPONENTS[1].SOLID[1]" ; "ALPHA").

LTDBGETIJ

DESCRIPTION:

Returns the data value of a numeric attribute in a two dimensional array corresponding to an object key, data element, and two index numbers.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBGETIJ(*objectkey\$* , *data_element\$* , *index1* , *index2*)

ARGUMENTS:

The *objectkey\$* argument is the object key for the attribute.

The *data_element\$* argument is the data element for the attribute.

The *index1* argument is the first array index number for the attribute.

The *index2* argument is the second array index number for the attribute.

RETURN VALUES:

The data value of the attribute.

NOTES:

The LTDBGETIJ function can be used to assign the numeric data value of a specific LightTools system attribute to a numeric variable. The attribute is a component of a two dimensional array. The attribute is specified with *objectkey\$*, *data_element\$*, and *index* arguments. Constant *objectkey\$* and *data_element\$* arguments must be enclosed within double quotation marks (e.g., "COMPONENTS[1].SOLID[1]" ; "ALPHA").

USAGE EXAMPLE 1: Accessing Mesh Values

The LTDBGETIJ function can be used to access mesh values (e.g., illuminance or intensity data) and some auxiliary data for all receiver mesh bins after you run a simulation. The values are accessible using the LTDBGETIJ command with the following keywords:

- CellValue - the value of the data in the bin (e.g., illuminance data for illuminance mesh)
- CellNumberOfRays - the number of rays collected by a mesh bin
- CellSumPower - the sum of ray powers in the bin
- CellVariance - the square of the sum of ray powers collected by a bin
- CellErrorEstimate - the error estimate for the specific bin.

MACRO Example for Accessing Mesh Values

```
REM ... assume a valid MeshKey$....

MeshXDim = LTDBGET(MeshKey$, "X_Dimension")
MeshYDim = LTDBGET(MeshKey$, "Y_Dimension")

FOR I=1 TO MeshXDim
  FOR J=1 TO MeshYDim
    MeshValue = LTDBGETIJ("ILLUMINANCE_MESH[1]", "CellValue", 3, 4)
    NumberOfRays = LTDBGETIJ(MeshKey$, "CellNumberOfRays", I, J)
    SumPower = LTDBGETIJ(MeshKey$, "CellSumPower", I, J)
    Variance = LTDBGETIJ(MeshKey$, "CellVariance", I, J)
    ErrorEstimate = LTDBGETIJ(MeshKey$, "CellErrorEstimate", I, J)

    PRINT "MeshValue[";I;"][";J;"] = ";MeshValue
    PRINT "NumberOfRays[";I;"][";J;"] = ";NumberOfRays
    PRINT "SumPower[";I;"][";J;"] = ";SumPower
    PRINT "Variance[";I;"][";J;"] = ";Variance
    PRINT "ErrorEstimate[";I;"][";J;"] = ";ErrorEstimate

  NEXT J

NEXT I
```

USAGE EXAMPLE 2: Accessing Saved Filter Data

The LTDBGETIJ function can be used to access saved filter data. The values are accessible using the LTDBGETIJ command with the keyword ExtraRayData.

This keyword takes two integer arguments. The first argument, called the *filter index* identifies which type of filter-specific data you'd like to access. The table below gives the filter indices that correspond to the different filter types, as well as the meanings of the return values of the keyword ExtraRayData.

Filter Index	Filter Type	Return Value
1	Source	1.0 if filter condition met; otherwise, 0.0.
2	Surface	1.0 if filter condition met; otherwise, 0.0.
3	Surface After	1.0 if filter condition met; otherwise, 0.0.
4	Element	1.0 if filter condition met; otherwise, 0.0.
5	Element After	1.0 if filter condition met; otherwise, 0.0.
6	Property Zone	1.0 if filter condition met; otherwise, 0.0.
7	Property Zone After	1.0 if filter condition met; otherwise, 0.0.
8	Hit Number	Number of times ray hit the receiver, as a real.
9	Ray Magnitude	Ray magnitude of ray.
10	Wavelength	Wavelength of ray (in nm).

Filter Index	Filter Type	Return Value
11	Incident Angle	Angle of incidence of ray on receiver (in degrees).
12	Exit Angle	Angle of exit of ray from receiver (in degrees).
13	Path Transmittance	Path transmittance of ray.
14	Volume Interface	1.0 if filter condition met; otherwise, 0.0.
15	Volume Interface After	1.0 if filter condition met; otherwise, 0.0.
16	Optical Path Length	Cumulative optical path length of ray at receiver.
17	Optical Property	1.0 if filter condition met; otherwise, 0.0.
18	Optical Property After	1.0 if filter condition met; otherwise, 0.0.
19	Polarization Data	The value of the polarization type specified for the filter (e.g., Stokes or Jones)
20	Ray Path	Base 0 Index of the Ray Paths (e.g., 0 would correspond to PathIndex=1 in the LightTools ray path)
21	Filter Group	1.0 if filter condition met; otherwise, 0.0.

The filters appear in the same order as in the above list in the Receiver Filter dialog box. If ExtraRayData is used to access data for a filter that has is not defined for the particular receiver, 0.0 is returned. If multiple Source, Surface, Element, Property Zone, or Volume Interface filters have been defined for a given receiver, the return value will apply only to the first filter defined for the particular type. Two filter types (RayDataWavelength and RayDataMagnitude) also have their own macro access functions.

The second argument for ExtraRayData is the index of the ray sample on the receiver.

Note that the data for a particular ray is accumulated at the receiver regardless of whether or not the ray passes the filters. This is different from the data displayed in the illumination chart views, which has had the filters applied.

MACRO Examples for Accessing Saved Filter Data

The following example shows the macro command for getting the path transmittance of the 2nd ray sample on a receiver:

```
pathT = LTDBGETIJ( ReceiverKey$, "ExtraRayData", 13, 2)
```

The following example shows the macro command to determine if the surface filter condition has been met for the 4th ray sample on the receiver:

```
surfaceHit = LTDBGETIJ( ReceiverKey$, "ExtraRayData", 2, 4)
```

LTDBGETIJ\$

DESCRIPTION:

Returns the data value of a string attribute in a two dimensional array corresponding to an object key, data element, and two index numbers.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBGETIJ\$(*objectkey\$* , *data_element\$* , *index1* , *index2*)

ARGUMENTS:

The *objectkey\$* argument is the object key for the attribute.

The *data_element\$* argument is the data element for the attribute.

The *index1* argument is the first array index number for the attribute.

The *index2* argument is the second array index number for the attribute.

RETURN VALUES:

The data value of the attribute.

NOTES:

The LTDBGETIJ\$ function can be used to assign the string data value of a specific LightTools system attribute to a string variable. The attribute is a component of a two dimensional array. The attribute is specified with *objectkey\$*, *data_element\$*, and *index* arguments. Constant *objectkey\$* and *data_element\$* arguments must be enclosed within double quotation marks (e.g., "COMPONENTS[1].SOLID[1]" ; "ALPHA").

LTDBGETSURFDATA

DESCRIPTION:

Retrieves the points used to define a user-defined spline-based lens surface.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBGETSURFDATA(*objectkey\$* , *data_array_2D* or *data_array_3D*)

ARGUMENTS:

objectkey\$ is a dataKey to a surface of type SPLINESWEEP or SPLINEPATCH.

The second argument depends on the surface type for *objectkey\$*, as follows:

Surface type for <i>objectkey\$</i>	Second Argument	Argument Description
SPLINESWEEP	<i>data_array_2D</i>	2-dimensional array containing the YZ coordinates of points used to define the lens profile curve. The second dimension of the array is always 2.
SPLINEPATCH	<i>data_array_3D</i>	3-dimensional array containing the XYZ coordinates of points used to define the lens surface patch. The third dimension of the array is always 3.

RETURN VALUES:

Points used to define a user-defined spline-based lens surface.

LTDBGETSURFVEC

DESCRIPTION:

Used to obtain the tangent vector from the start or end of the curve that was used to define a spline sweep lens surface. See *LTSURFSPLINESWEEP* on page 140 for more information.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBGETSURFVEC(*surfKey\$* , *startEndFlag* , *Y* , *Z*)

ARGUMENTS:

surfKey\$ (input) identifies the spline sweep surface of interest

startEndFlag (input) 1=start tangent, 2=end tangent.

Y (output) contains the Y component of the tangent vector,

Z (output) contains the Z component of the tangent vector.

RETURN VALUES:

0 - success. Y and Z values are correctly set.

1 - invalid *surfKey\$* specified.

2 - invalued value of *startEndFlag* specified.

3 - invalid third parameter, *Y*.

4 - invalid fourth parameter, *Z*.

LTDBKEYDUMP

DESCRIPTION:

Returns a list of data elements corresponding to an object key and writes them to the LightTools Console window or a file.

TYPE:

Function (LightTools Data Access)

SYNTAX:

```
LTDBKEYDUMP( objectkey$ [ , file-name$ ] )
```

ARGUMENTS:

The *objectkey*\$ argument is the object key for the data elements.

The *file-name*\$ argument is the file to which the data elements are written.

RETURN VALUES:

The status of the writing of the list.

NOTES:

The LTDBKEYDUMP function can be used to generate a list of data elements for a specific LightTools system attribute which can be assigned to a string variable. The data elements are specified with a *objectkey*\$ argument. Constant *objectkey*\$ arguments must be enclosed within double quotation marks (e.g., "COMPONENTS[1].SOLID[1]"). The data elements determined must be valid for the *objectkey*\$ argument. When the list of data elements is determined, it is printed in the LightTools Console window. In addition, the list of data elements can be written to a designated file. The *file-name*\$ argument can be a string constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "myfile.txt").

The LTDBKEYDUMP function returns an integer status value which is indicative of the success of writing the list. The returned status value would be 0 if the writing of the list occurred without error, or greater than zero if the writing of the list produced an error. The returned status value must be assigned to a user-defined variable unless the function is used as an argument or in an expression.

EXAMPLES:

1. `Status = LTDBKEYDUMP("COMPONENTS[1]", "myLTfile1.txt")`
2. `Status = LTDBKEYDUMP("SOURCES[1]")`

LTDBKEYSTR\$

DESCRIPTION:

Returns an equivalent data access string translated from a corresponding object key.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBKEYSTR\$(*objectkey*\$)

ARGUMENTS:

The *objectkey*\$ argument is the object key for which the data access string is returned.

RETURN VALUES:

The data access string.

NOTES:

The LTDBKEYSTR\$ function returns a data access string minus a data element translated from an object key. The translated data access string can be combined with data elements to construct complete data access strings for use as input in LightTools MACRO commands and data access functions. If the LTDBKEYSTR\$ function is used with a literal object key, it returns the name of the object as listed in a Table View. Constant *objectkey*\$ arguments must be enclosed within double quotation marks (e.g., "COMPONENTS[1].SOLID[1]").

EXAMPLES:

1. SolidList\$ = LTDBLIST\$("COMPONENTS[1]", "SOLID")
Solid1\$ = LTLISTNEXT\$(SolidList\$)
Solid1String\$ = LTDBKEYSTR\$(Solid1\$)
2. Prim1Name\$ = LTDBKEYSTR\$("COMPONENTS[1].LENS_PRIMITIVE[1]")

LTDBLIST\$

DESCRIPTION:

Returns a list key corresponding to an object key and filter.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBLIST\$(*objectkey\$* , *filter\$*)

ARGUMENTS:

The *objectkey\$* argument is the object key for which the list is created.

The *filter\$* argument is the filter for which the list is created.

RETURN VALUES:

The list key.

NOTES:

The LTDBLIST\$ function can be used to establish a list of object keys to provide input for LightTools MACRO data access functions. The list is compiled from all objects in the LightTools database.

Constant *objectkey\$* and *filter\$* arguments must be enclosed within double quotation marks (e.g., "COMPONENTS[1]" ; "SOLID").

EXAMPLES:

```
1. SolidList$ = LTDBLIST$ ("COMPONENTS[1]" , "SOLID")
```

LTDBQUICKRAYAIM

DESCRIPTION:

Enables you to quickly aim and trace a single NS ray (up to 20 times faster than the regular NS ray command), and store the ray trace data internally.

Note: This ray is not visible in any graphics view or table view. Only one ray of this type can exist at any time. Re-executing the LTDBQUICKRAYAIM command will redefine the data for this ray.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBQUICKRAYAIM(*rayVec*, *surfKey\$*, *outVec*)

ARGUMENTS:

The *rayVec* argument is a variable dimensioned to at least 6 that holds the position and direction of the ray: X, Y, Z, L, M, N.

The *surfKey\$* argument is a dataKey indicating the surface of interest that the ray will likely hit.

The *outVec* argument is a variable dimensioned to at least 10 that will hold the following values for the segment of the ray that first hits the surface of interest: X, Y, Z, L, M, N, PathTransmittance, opticalPathLength, AngleOfIncidence, and AngleOfExit.

RETURN VALUES:

The number of ray segments that hit the surface of interest.

EXAMPLES:

```
1.  DIM rayVec(6),outVec(10)
    rayVec(1) = 0
    rayVec(2) = 0
    rayVec(3) = 0
    rayVec(4) = 0
    rayVec(5) = 0
    rayVec(6) = 1
    segcount = LTDBQUICKRAYAIM(rayVec, surfKey$, outVec)
```

LTDBQUICKRAYQUERY

DESCRIPTION:

Permits queries against data computed by LTDBQUICKRAYAIM. Output data as described above can be obtained for any valid segment or surface specified. The surface of interest need not be the same as in the initial call to LTDBQUICKRAYAIM.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBQUICKRAYQUERY(*segId*, *surfKey\$*, *outvec*)

ARGUMENTS:

The *segId* argument is the segment for which output should be obtained. The first segment is segment 1, the second is 2, and so on.

The *surfKey\$* argument is a dataKey indicating the surface of interest that the ray will likely hit.

The *outvec* argument is a variable dimensioned to at least 10 that will hold the following values for the segment of the ray that first hits the surface of interest: X, Y, Z, L, M, N, PathTransmittance, opticalPathLength, AngleOfIncidence, and AngleOfExit.

RETURN VALUES:

The number of ray segments that hit the surface of interest.

NOTES:

segId=0 returns data for the initial ray segment. If segId is 0, the value of surfKey\$ is ignored.

EXAMPLES:

```
1.  DIM outVec(10)
    segId = 0
    segcount = LTDBQUICKRAYQUERY(segId, surfKey$, outVec)
```

LTDBSET

DESCRIPTION:

Assigns a numeric or string data value to an attribute.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBSET(*objectkey\$* , *data_element\$* , *data_value*)

ARGUMENTS:

The *objectkey\$* argument is the object key for the attribute.

The *data_element\$* argument is the data element for the attribute.

The *data_value* argument is the data value assigned to the attribute.

Note: Partial strings can be accepted for the *data_value*. For example, you can input “Y” or “y” for “Yes.”

RETURN VALUES:

The status of the data value assignment.

NOTES:

The LTDBSET function can be used to modify a specific LightTools system attribute. The attribute is specified by *objectkey\$* and *data_element\$* arguments. Constant *objectkey\$* and *data_element\$* arguments must be enclosed within double quotation marks (e.g., "COMPONENTS[1].SOLID[1]" ; "ALPHA"). The *data_value* arguments can be constants, variables, functions, or expressions. Constant string *data_value* arguments must be enclosed within double quotation marks (e.g., "Circular"). The attribute type must match the data value type.

The LTDBSET function returns an integer status value which is indicative of the success of the assignment. The returned status value would be 0 if the assignment occurred without error, or greater than zero if the assignment produced an error. The returned status value must be assigned to a user-defined variable unless the function is used as an argument or in an expression.

EXAMPLES:

1. Status = LTDBSET(Primitive\$, "ALPHA", 45)
2. Status = LTDBSET(Primitive\$, "NAME", "Lens Primitive")

LTDBSETI

DESCRIPTION:

Assigns a numeric or string data value to an attribute in a one dimensional array.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBSETI(*objectkey\$* , *data_element\$* , *index* , *data_value*)

ARGUMENTS:

The *objectkey\$* argument is the object key for the attribute.

The *data_element\$* argument is the data element for the attribute.

The *index* argument is the array index number for the attribute.

The *data_value* argument is the data value assigned to the attribute.

Note: Partial strings can be accepted for the *data_value*. For example, you can input “Y” or “y” for “Yes.”

RETURN VALUES:

The status of the data value assignment.

NOTES:

The LTDBSETI function can be used to modify a specific LightTools system attribute. The attribute is a component of a one dimensional array. The attribute is specified by *objectkey\$*, *data_element\$*, and *index* arguments. Constant *objectkey\$* and *data_element\$* arguments must be enclosed within double quotation marks (e.g., "COMPONENTS[1].SOLID[1]" ; "ALPHA"). The *data_value* arguments can be constants, variables, functions, or expressions. Constant string *data_value* arguments must be enclosed within double quotation marks (e.g., "Circular"). The attribute type must match the data value type.

The LTDBSETI function returns an integer status value which is indicative of the success of the assignment. The returned status value would be 0 if the assignment occurred without error, or greater than zero if the assignment produced an error. The returned status value must be assigned to a user-defined variable unless the function is used as an argument or in an expression.

LTDBSETSURFVEC

DESCRIPTION:

Used to change the tangent vector (start or end) of the curve which is used to define a spline sweep lens surface. See *LTSURFSPLINESWEEP* on page 140 for more information.

TYPE:

Function (LightTools Data Access)

SYNTAX:

stat = LTDBSETSURFVEC(*surfKey\$*, *startEndFlag*, *Y*, *Z*)

ARGUMENTS:

surfKey\$ - (input) identifies the spline sweep surface of interest.

startEndFlag - (input) 1=start tangent, 2=end tangent.

Y - (input) specifies the y component of the tangent vector.

Z - (input) specifies the z component of the tangent vector.

RETURN VALUES:

0 - success. Y and Z values are correctly set, and the surface is regenerated.

1 - bad parameter. An invalid value was specified for either *surfKey\$*, *startEndFlag*, *Y*, or *Z*.

LTDBTYPE

DESCRIPTION:

Returns the status of an object key as appropriate or not for a filter.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTDBTYPE(*objectkey\$* , *filter\$*)

ARGUMENTS:

The *objectkey\$* argument is the object key for which the status is returned.

The *filter\$* argument is the filter for which the status is returned.

RETURN VALUES:

The status of the object key.

NOTES:

The LTDBTYPE function can be used to determine if a specific LightTools system attribute is being accessed in an appropriate manner. Constant *objectkey\$* and *filter\$* arguments must be enclosed within double quotation marks (e.g., "COMPONENTS[1]" ; "SOLID").

The LTDBTYPE function returns an integer status value which is indicative of the validity of the object key for the filter. The returned status value would be 1 if the object key is valid, or 0 if the object key is not valid. The returned status value must be assigned to a user-defined variable unless the function is used as an argument or in an expression.

EXAMPLES:

```
1. SolidList$ = LTDBLIST$ ("COMPONENTS[1]" , "SOLID")
   Solid$ = LTLISTNEXT$ (SolidList$)
   SolidTest = LTDBTYPE (Solid$ , "SOLID")
   PRINT "SolidTest = " ; SolidTest
   PolylineTest = LTDBTYPE (Solid$ , "POLYLINE")
   PRINT "PolylineTest = " ; PolylineTest

   SolidTest = 1
   PolylineTest = 0
```


LTEVAL

DESCRIPTION:

Returns the numeric value of a string evaluated by LightTools.

TYPE:

Function (LightTools MACRO)

SYNTAX:

`LTEVAL($$)`

ARGUMENTS:

The *\$\$* argument is the string evaluated.

RETURN VALUES:

The numeric value of the evaluated string.

NOTES:

The LTEVAL function can be used to perform mathematical computations within a LightTools MACRO program. The string can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "X + Y"). The string is passed to the LightTools string evaluator. The evaluator determines the numeric value of the string.

EXAMPLES:

```
1. TotalAlpha = LTEVAL("{SOLID[1].Alpha} + {SOLID[2].Alpha}")
```

LTGETLASTMSG\$

DESCRIPTION:

Returns the last message that LightTools printed out.

TYPE:

Function (LightTools MACRO)

SYNTAX:

msg\$ = LTGETLASTMSG\$(*n*)

ARGUMENTS:

- n* - (input) identifies what type of message to return.
- 1 (default) means return last error message.
 - 2 means return last warning message.
 - 3 means return last info message.

RETURN VALUES:

The message string.

LTGETPHOTOPICFUNCTION

DESCRIPTION:

Returns the photopic spectral luminous efficiency for a given wavelength.

TYPE:

Function (LightTools MACRO)

SYNTAX:

LTGETPHOTOPICFUNCTION(*wavelength*)

ARGUMENTS:

The *wavelength* argument is the wavelength value, given in nanometers.

RETURN VALUES:

The photopic spectral luminous efficiency.

NOTES:

EXAMPLES:

```
1. PhotopicResponse = LTGETPHOTOPICFUNCTION(550)
```

LTGETSCOTOPICFUNCTION

DESCRIPTION:

Returns the scotopic spectral luminous efficiency for a given wavelength.

TYPE:

Function (LightTools MACRO)

SYNTAX:

LTGETSCOTOPICFUNCTION(*wavelength*)

ARGUMENTS:

The *wavelength* argument is the wavelength value, given in nanometers.

RETURN VALUES:

The scotopic spectral luminous efficiency.

NOTES:

EXAMPLES:

```
1. ScotopicResponse = LTGETPHOTOPICFUNCTION(550)
```

LTGETSTAT

DESCRIPTION:

Returns the status of the most recently executed LightTools command.

TYPE:

Function (LightTools MACRO)

SYNTAX:

LTGETSTAT()

ARGUMENTS:

The () argument is the null character enclosed within parentheses.

RETURN VALUES:

The status of the most recently executed LightTools command.

NOTES:

The LTGETSTAT function can be used to determine the results of the execution of a LightTools command. The entire set of LightTools commands are executable within a LightTools MACRO program with the LTCMD command. The entire set of LightTools commands and their functionality and syntax can be found in the *LightTools Command Reference Guide*, which is an online document provided with the LightTools software.

The LTGETSTAT function returns an integer status value which is indicative of the success of the execution of the LightTools command. The returned status value would be 0 if the execution of the command occurred without error, or greater than zero if the execution of the command produced an error. The returned status value must be assigned to a user-defined variable unless the function is used as an argument or in an expression.

EXAMPLES:

```
1. SolidList$ = LTDBLIST$("COMPONENTS[1]","SOLID")
   Solid1$ = LTLISTNEXT$(SolidList$)
   Solid1Name$ = LTDBGET$(Solid1$,"Name")
   LTCMD "Select " + LTSTR$(Solid1Name$)
   CommandStatus = LTGETSTAT()
   PRINT "CommandStatus = ";CommandStatus

   CommandStatus = 0
```

LTGETVAR

DESCRIPTION:

Returns the value of a LightTools numeric variable (predefined or user-defined).

TYPE:

Function (LightTools MACRO)

SYNTAX:

LTGETVAR(*V*)

ARGUMENTS:

The *V* argument is the LightTools variable whose value is returned.

RETURN VALUES:

The value of the LightTools variable.

NOTES:

The LTGETVAR function can be used to access the result of a specific LightTools system operation. The *V* argument must be a LightTools numeric variable (predefined or user-defined).

EXAMPLES:

```
1.  LTCMD "Define X=5"
    Value = LTGETVAR("X")
    PRINT "Value = ";Value

    Value = 5
```

LTGETVAR\$

DESCRIPTION:

Returns the value of a LightTools string variable (predefined or user-defined).

TYPE:

Function (LightTools MACRO)

SYNTAX:

LTGETVAR\$(*\$\$*)

ARGUMENTS:

The *\$\$* argument is the LightTools variable whose value is returned.

RETURN VALUES:

The value of the LightTools variable.

NOTES:

The LTGETVAR\$ function can be used to access the result of a specific LightTools system operation. The *\$\$* argument must be a LightTools string variable (predefined or user-defined).

EXAMPLES:

```
1.  LTCMD "Define Name=" + LTSTR$("NewLens")
    NewName$ = LTGETVAR$("Name")
    PRINT "The system is named ";NewName$;"."
```

The system is named NewLens.

LTLISTATPOS\$

DESCRIPTION:

Returns the object key at a position in a list of object keys.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTLISTATPOS\$(*listkey\$* , *N*)

ARGUMENTS:

The *listkey\$* argument is the list of object keys.

The *N* argument is the position from which the object key is returned.

RETURN VALUES:

The object key at the position.

NOTES:

The LTLISTATPOS\$ function can be used to determine the object key for a LightTools system attribute at a specific position in the LightTools system. The position must be a whole number. The range of values for the *N* argument is $1 \leq N \leq \lceil \text{LTLISTSIZE}(\textit{listkey\$}) + 0.999... \rceil$. It can be a constant, variable, function, or expression. If the *N* argument is not a whole number, the decimal places in its value are truncated to produce a whole number.

EXAMPLES:

```
1.  LensList$ = LTDBLIST$("COMPONENTS[1]", "CIRC_LENS_PRIMITIVE")
    LensKey$ = LTLISTATPOS$(LensList$, 5)
```


LTLISTBYNAME\$

DESCRIPTION:

Returns an object key from a list of object keys corresponding to an object name.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTLISTBYNAME\$(*listkey\$* , *name\$*)

ARGUMENTS:

The *listkey\$* argument is the list from which the object key is returned.

The *name\$* argument is the object for which the object key is returned.

RETURN VALUES:

The object key for the object.

NOTES:

The LTLISTBYNAME\$ function can be used to access a LightTools system attribute for a specific object. The object key corresponds to an object with a designated name.

EXAMPLES:

```
1. SurfaceList$ = LTDBLIST$("LENS_PRIMITIVE[1]","SURFACE")
   RearSurfaceKey$ = LTLISTBYNAME$(SurfaceList$,"LensRearSurface")
```

LTLISTDELETE

DESCRIPTION:

Deletes the memory contents for a list key.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTLISTDELETE(*listkey*\$)

ARGUMENTS:

The *listkey*\$ argument is the list which is deleted from memory.

RETURN VALUES:

The status of the list deletion.

NOTES:

The LTLISTDELETE function can be used to clear a portion of memory that is no longer required by a LightTools MACRO program.

The LTLISTDELETE function returns an integer status value which is indicative of the success of the deletion. The returned status value would be 0 if the deletion occurred without error, or greater than zero if the deletion produced an error. The returned status value must be assigned to a user-defined variable unless the function is used as an argument or in an expression.

EXAMPLES:

```
1. ListStatus = LTLISTDELETE(SolidList$)
```

LTLISTGETPOS

DESCRIPTION:

Returns the current position of the given list.

TYPE:

Function (LightTools Data Access)

SYNTAX:

pos = LTLISTGETPOS(*listKey\$*)

ARGUMENTS:

listKey\$ - (input) identifies the list of interest.

RETURN VALUES:

The current position of the list.

LTLISTLAST\$

DESCRIPTION:

Returns the object key for the last position of a list key.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTLISTLAST\$(*listkey*\$)

ARGUMENTS:

The *listkey*\$ argument is the list for which the object key is returned.

RETURN VALUES:

The object key for the last position in the list.

NOTES:

The LTLISTLAST\$ function can be used to determine the type of a specific LightTools system attribute in a sequence of system attributes.

EXAMPLES:

```
1. SurfaceList$ = LTDBLIST$("PRIMITIVE[1]","SURFACE")
   LastSurfaceKey$ = LTLISTLAST(list1$)
```

LTLISTNEXT\$

DESCRIPTION:

Returns the object key at the current position of a list key and sets the list key to the next position.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTLISTNEXT\$(*listkey\$*)

ARGUMENTS:

The *listkey\$* argument is the list for which the object key is returned.

RETURN VALUES:

The object key for the current position in the list.

NOTES:

The LTLISTNEXT\$ function can be used to determine the sequential order of specific LightTools system attributes. It returns the object key at the current position and advances the list by one position with respect to the current position.

When a list is created, and its current position has not been modified, an LTLISTNEXT\$ function returns the object key at position 1 in the list. Subsequent LTLISTNEXT\$ functions return the object keys at positions 2, 3, 4, and so on. When the object key at the last position in the list has been returned by an LTLISTNEXT\$ function, the list is not reset to position 1. Subsequent LTLISTNEXT\$ functions returns a null string as an object key.

EXAMPLES:

```
1. SurfaceList$ = LTDBLIST$("PRIMITIVE[1]","SURFACE")
   FirstSurfaceKey$ = LTLISTNEXT$(SurfaceList$)
```

LTLISTSETPOS

DESCRIPTION:

Sets the position of a list key.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTLISTSETPOS(*listkey\$* , *N*)

ARGUMENTS:

The *listkey\$* argument is the list for which the position is set.

The *N* argument is the position to which the list is set.

RETURN VALUES:

The status of the list setting.

NOTES:

The LTLISTSETPOS function can be used to access a LightTools system attribute at a specific position in the system. The position must be a whole number. The range of values for the *N* argument is $1 \leq N \leq [\text{LTLISTSIZE}(\text{listkey\$}) + 0.999...]$. It can be a constant, variable, function, or expression. If the *N* argument is not a whole number, the decimal places in its value are truncated to produce a whole number.

The LTLISTSETPOS function returns an integer status value which is indicative of the success of the setting. The returned status value would be 0 if the setting occurred without error, or greater than zero if the setting produced an error. The returned status value must be assigned to a user-defined variable unless the function is used as an argument or in an expression.

EXAMPLES:

```
1.  Status = LTLISTSETPOS(SurfaceList$,4)
    FourthSurfaceKey$ = LTLISTNEXT$(SurfaceList$)
```

LTLISTSIZE

DESCRIPTION:

Returns the number of object keys in a list key.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTLISTSIZE(*listkey\$*)

ARGUMENTS:

The *listkey\$* argument is the list whose size is returned.

RETURN VALUES:

The number of object keys in the list.

NOTES:

The LTLISTSIZE function can be used to determine the number of components of a specific type in a LightTools system.

EXAMPLES:

```
1. TotalSolids = LTLISTSIZE(SolidList$)
```

LTSELECTLIST\$

DESCRIPTION:

Returns a list key corresponding to a filter.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTSELECTLIST\$(*filter\$*)

ARGUMENTS:

The *filter\$* argument is the filter for which the list is created.

RETURN VALUES:

The list key.

NOTES:

The LTSELECTLIST\$ function can be used to determine the components in a LightTools system that correspond to a specific data access filter. The list of object keys returned by the LTSELECTLIST\$ function is created from the objects that are in the currently active view window. The currently active view window can be changed at any point in the execution of a macro program using the LTCMD command with the \V window command (e.g., LTCMD "\V<3D_untyped>").

The data access filter used as the *filter\$* argument of the LTSELECTLIST\$ function can be at any level of the filter hierarchy. Constant *filter\$* arguments must be enclosed within double quotation marks (e.g., "ENTITY").

EXAMPLES:

```
1. FilterList$ = LTSELECTLIST$("ENTITY")
```


LTSETVAR

DESCRIPTION:

Assigns a numeric or string value to a user-defined LightTools variable

TYPE:

Function (LightTools MACRO)

SYNTAX:

LTSETVAR(*V* , *value*)

ARGUMENTS:

The *V* argument is the LightTools variable to which the value is assigned.

The *value* argument is the value assigned to the LightTools variable.

RETURN VALUES:

The status of the value assignment.

NOTES:

The LTSETVAR function can be used to modify a specific LightTools system attribute. The attribute is specified by the LightTools variable.

The *V* argument must be a LightTools variable (user-defined only). The variable name must be enclosed within double quotation marks. It can be a numeric or string variable. Its type must match that of the value assigned. The values can be constants, variables, functions, or expressions. String values must be enclosed within double quotation marks (e.g., "ABCDE"). Only one type of argument can be used in a LTSETVAR command. An attempt to assign a *value* argument to a *V* argument whose types do not match results in an error.

The LTSETVAR function returns an integer status value which is indicative of the success of the assignment. The returned status value would be 0 if the assignment occurred without error, or greater than zero if the assignment produced an error. The returned status value must be assigned to a user-defined variable unless the function is used as an argument or in an expression.

EXAMPLES:

```
1.  Status = LTSETVAR("X",5)
    Value = LTGETVAR("X")
    PRINT "Value = ";Value

    Value = 5
```

LTSTR\$

DESCRIPTION:

Returns a quoted string formatted as input for the LightTools specific commands and functions.

TYPE:

Function (LightTools MACRO)

SYNTAX:

LTSTR\$(*\$\$*)

ARGUMENTS:

The *\$\$* argument is the string for which the quoted string is returned.

RETURN VALUES:

A quoted string.

NOTES:

The LTSTR\$ function can be used for executing LightTools commands in a repetitive fashion with the LTCMD command. The entire command line can be represented by a variable in order to reduce the amount of repetitive program input. The *command\$* argument of the LTCMD command requires enclosure within double quotation marks. The LTSTR\$ function can also be used to supply input for the LightTools specific functions such as LTSETVAR.

The *\$\$* argument can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "Lens_1").

EXAMPLES:

1. `LensName$ = LTSTR$(LTDBGET$(LensKey$, "Name"))`
2. `LTCMD "Select " + LTSTR$(LTDBGET$("PRIMITIVE[1]", "Name"))`

LTSURFSPLINEPATCH

DESCRIPTION:

Defines a two-dimensional mesh of points to which LightTools fits a surface.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTSURFSPLINEPATCH(*objectkey\$* , *data_array_3D* , *data_value1* , *data_value2*)

ARGUMENTS:

objectkey\$ is a dataKey to a surface of type SPLINEPATCH.

data_array_3D is a 3-dimensional array containing the XYZ coordinates of points used to define the lens surface patch. The third dimension of the array is always 3.

data_value1 is the number of data points in the U direction.

data_value2 is the number of data points in the V direction.

RETURN VALUES:

0 - Good.

LTSURFSPLINESWEEP

DESCRIPTION:

Defines a spline curve that is swept about the Z axis of a lens surface to create a rotationally symmetric lens surface.

TYPE:

Function (LightTools Data Access)

SYNTAX:

LTSURFSPLINESWEEP(*objectkey\$* , *data_array_2D* , *data_value* , *data_array1* , *data_array2*)

ARGUMENTS:

objectkey\$ is a dataKey to a surface of type SPLINESWEEP.

data_array_2D is a 2-dimensional array containing the YZ coordinates of points used to define the lens profile curve. The second dimension of the array is always 2.

data_value is the number of data points supplied in *data_array_2D*.

data_array1 a vector specifying a unit vector tangent to the start of the profile curve. This argument is OPTIONAL; if it is not specified, the starting tangent is computed as the vector between the first two points specified in *data_array_2D*.

data_array2 is a vector specifying a unit vector tangent to the end of the profile curve. This argument is OPTIONAL; if it is not specified, the ending tangent is computed as the vector between the last two points specified in *data_array_2D*.

RETURN VALUES:

0 - Good.

LTVERSION\$

DESCRIPTION:

Returns a string composed of information regarding the version of LightTools.

TYPE:

Function (LightTools MACRO)

SYNTAX:

LTVERSION\$(*N*)

ARGUMENTS:

The *N* argument specifies the type of information returned.

RETURN VALUES:

The string composed of information regarding the version of LightTools.

NOTES:

The information contained in the string returned by the LTVERSION\$ function is the same as that found by selecting the About... option under the Help menu item. The *N* argument has four numeric value options which specify the following information:

- 0 - Returns the version number only (default value).
- 1 - Returns the version string.
- 2 - Returns the version string and build number.
- 3 - Returns the build number.

EXAMPLES:

1.

```
VersionDate$ = LTVERSION$(1)
PRINT "This version of LightTools was built on ";VersionDate$;"."
```

This version of LightTools was built on 10-12-2018.
2.

```
VersionNumber$ = LTVERSION$( )
PRINT "This is version number ";VersionNumber$;" of LightTools."
```

This is version number 8.5.0 of LightTools.

MAX

DESCRIPTION:

Returns the higher value of two numbers.

TYPE:

Function (Mathematical)

SYNTAX:

`MAX(X , Y)`

ARGUMENTS:

The *X* and *Y* arguments are the numbers for which the higher value is returned.

RETURN VALUES:

The higher value of the *X* and *Y* arguments.

NOTES:

The *X* and *Y* arguments must be real numbers. They can be constants, variables, functions, or expressions. The ranges of values are $-\infty < X < \infty$ and $-\infty < Y < \infty$.

The range of values for `MAX(X,Y)` is $-\infty < \text{MAX}(\textit{X},\textit{Y}) < \infty$. The returned value of MAX has the same precision as its arguments.

EXAMPLES:

```
1.  X = 49.99
    Y = 50.00
    Z = MAX(X,Y)
    PRINT X,Y,Z
```

49.9900000	50	50
------------	----	----

MERGE

DESCRIPTION:

Appends a LightTools MACRO program file to the program currently being executed.

TYPE:

Command (Program Control)

SYNTAX:

MERGE *file-name*\$

ARGUMENTS:

The *file-name*\$ argument is the name of the LightTools MACRO program file which is to be appended to the program currently being executed.

RETURN VALUES:

None.

NOTES:

The MERGE command appends the statements in a program file to the program that is currently being executed and is resident in memory. The statements are appended following the final statement of the program in memory.

The *file-name*\$ argument must be a LightTools MACRO program file. It can be a string constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "myfile.1.ltb"). The program file designated by the *file-name*\$ argument remains intact in disk storage. If the two programs have line numbers in common, the lines in the current program are replaced by the corresponding lines in the appended program.

When the two programs are merged together, all variables are cleared from memory (see LET command), all data is restored (see RESTORE command), all dimensioned arrays become undimensioned (see DIM command), all open files are closed (see OPEN FOR APPEND | INPUT | OUTPUT AS # commands), and all error handling is disabled (see ON ERROR GOSUB command).

The MERGE command can be used to append a program in a manner similar to that of a subprogram. If the program currently being executed terminates with an END command, the appended program can only be executed as a subprogram.

The MERGE command can be used as a database access technique. If the appended program is structured as a data file (see DATA command), the data can be transferred to the program currently being executed.

EXAMPLES:

```
1.  FOR I = 0 TO 4
    J = I - 2
    IF J > 0 THEN
        PRINT "I = ";I,"J is greater than zero."
    ELSEIF J < 0 THEN
        PRINT "I = ";I,"J is less than zero."
    ELSE
        PRINT "I = ";I,"J is equal to zero."
    END IF
NEXT I
MERGE "Conclusion.1.ltb"

I = 0    J is less than zero.
I = 1    J is less than zero.
I = 2    J is equal to zero.
I = 3    J is greater than zero.
I = 4    J is greater than zero.
J has been evaluated for all I.
```

Contents of Conclusion.1.ltb:

```
PRINT "J has been evaluated for all I."
```


MID\$

DESCRIPTION:

Returns a substring with a designated length from a designated position in a string.

TYPE:

Function (String)

SYNTAX:

MID\$(*SS* , *M* [, *N*])

ARGUMENTS:

The *SS* argument is the string for which the substring is returned.

The *M* argument is the position in the string at which the substring commences.

The *N* argument is the number of characters in the substring.

RETURN VALUES:

The substring with the designated length.

NOTES:

The MID\$ function creates a new string, and the string argument remains intact. The substring is composed of the characters from a specific position on in the string argument. Alternatively, the number of characters in the substring can be specified. The MID\$ function is only used with strings. The MID\$ function is analogous to the RIGHT\$ and LEFT\$ functions.

The *SS* argument can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE").

The position must be a whole number. The range of values for the *M* argument is $1 \leq M \leq 255.999...$. It can be a constant, variable, function, or expression. If the *M* argument is not a whole number, the decimal places in its value are truncated to produce a whole number. If the *M* argument is 1 and the *N* argument is not specified, the entire string is returned. If the *M* argument is greater than the number of characters in the string, the null string ("") is returned.

The number of characters must be a whole number. The range of values for the *N* argument is

$1 \leq N \leq 255.999...$. It can be a constant, variable, function, or expression. If the *N* argument is not a whole number, the decimal places in its value are truncated to produce a whole number. If the *N* argument is greater than the number of characters from the position specified by the *M* argument to the end of the string, only those characters are contained in the returned string.

EXAMPLES:

1. S\$ = "ABCDE"
T\$ = MID\$(S\$, 3)
PRINT S\$, T\$

ABCDE CDE

2. S\$ = "ABCDE"
T\$ = MID\$(S\$, 3, 2)
PRINT S\$, T\$

ABCDE CD

MIN

DESCRIPTION:

Returns the lower value of two numbers.

TYPE:

Function (Mathematical)

SYNTAX:

$\text{MIN}(X, Y)$

ARGUMENTS:

The X and Y arguments are the numbers for which the lower value is returned.

RETURN VALUES:

The lower value of the X and Y arguments.

NOTES:

The X and Y arguments must be real numbers. They can be constants, variables, functions, or expressions. The ranges of values are $-\infty < X < \infty$ and $-\infty < Y < \infty$.

The range of values for $\text{MIN}(X, Y)$ is $-\infty < \text{MIN}(X, Y) < \infty$. The returned value of MIN has the same precision as its arguments.

EXAMPLES:

```
1.  X = 49.99
    Y = 50.00
    Z = MIN(X, Y)
    PRINT X, Y, Z
```

49.9900000

50

49.9900000

NEXT

DESCRIPTION:

Continues a FOR loop initiated by the FOR command.

TYPE:

Command (Program Control)

SYNTAX:

NEXT *counter*

ARGUMENTS:

The *counter* argument is the counter index

RETURN VALUES:

None.

NOTES:

The NEXT command increments the counter index specified by the FOR command and continue with the next repetition of the FOR loop until the counter index exceeds its final value. It is used in conjunction with the FOR command. The NEXT command is the last statement in a sequence of statements which constitutes a FOR loop. When the FOR loop is terminated, the program is directed to the statement immediately following the NEXT command.

EXAMPLES:

```
1.  X = 0
    DELTAX = 25
      FOR I = 1 TO 5
        X = X + DELTAX
        PRINT "X = ";X
      NEXT I

    X = 25
    X = 50
    X = 75
    X = 100
    X = 125
```

ON ERROR GOSUB/ON ERROR GO SUB

DESCRIPTION:

Directs the program to a subprogram when an error occurs.

TYPE:

Command (Program Control)

SYNTAX:

ON ERROR GOSUB *label\$*

ARGUMENTS:

The *label\$* argument is the label for the subprogram.

RETURN VALUES:

None.

NOTES:

The ON ERROR GOSUB command directs the program to a line specified by a label when an error occurs (except when the DIVIDE BY ZERO error occurs). The label identifies a subprogram. The *label\$* argument used in the ON ERROR GOSUB command is a string. The label in the statement that the program is directed to by the ON ERROR GOSUB command is the same string as the *label\$* argument with a trailing colon. The GOSUB component of this command can be entered as one or two keywords (i.e., GOSUB or GO SUB).

The sequence of statements which constitutes the subprogram that the program is directed to must have a RETURN command as the last statement. When the RETURN command is executed, the program is directed to the statement immediately following the statement in which the error occurred. If multiple ON ERROR GOSUB commands have been executed prior to encountering the RETURN command, the program is directed to the statement immediately following the statement in which the most recent error occurred when the RETURN command is executed.

The ON ERROR GOSUB command establishes an error handler, which is an operation that is performed when an error is encountered. The error handler established can be modified with subsequent ON ERROR GOSUB commands. Alternatively, the error handler established can be disabled and the system default error handler restored with the ON ERROR GOSUB 0 command.

EXAMPLES:

```
1.  X = -3
    Y = -3
    ON ERROR GOSUB FIX
      FOR I = 1 TO 5
        X = X + 1
        Y = Y + 1
        Z =
        PRINT "X = ";X,"Y = ";Y,"Z = ";Z
      NEXT I
    END
    FIX:
      Z = X/Y
    RETURN

X = -2  Y = -2  Z = 1
X = -1  Y = -1  Z = 1
ERROR (12): Divide by zero
X = 0   Y = 0   Z = 0
X = 1   Y = 1   Z = 1
X = 2   Y = 2   Z = 1
```

ON GOSUB/ON GO SUB

DESCRIPTION:

Directs the program to a subprogram designated by the value of a variable.

TYPE:

Command (Program Control)

SYNTAX:

ON *V* GOSUB *line1* [, *line2* ...]

ARGUMENTS:

The *V* argument is the variable which designates the subprogram.

The *linek* arguments are the line numbers for the subprograms.

RETURN VALUES:

None.

NOTES:

The ON GOSUB command directs the program to a specific subprogram based upon the value of a variable. The GOSUB component of this command can be entered as one or two keywords (i.e., GOSUB or GO SUB). The sequence of statements which constitutes the subprogram that the program is directed to must have a RETURN command as the last statement. When the RETURN command is executed, the program is directed to the statement immediately following the ON GOSUB command. If multiple ON GOSUB commands have been executed prior to encountering the RETURN command, the program is directed to the statement immediately following the most recent ON GOSUB command when the RETURN command is executed.

The *V* argument must be a whole number. The range of values is $1 \leq V < \infty$. If the *V* argument is not a whole number, the decimal places in its value are truncated to produce a whole number. The line numbers to which the program is directed are designated by the *V* argument. If *V* is 1, the program is directed to the line number designated by the *line1* argument. If *V* is 2, the program is directed to the line number designated by the *line2* argument. This procedure is followed for all values of the *V* argument and their corresponding *linek* arguments. Only the *V* and *line1* arguments are required. The other *linek* arguments are optional. The *linek* arguments used in the ON GOSUB command are numbers. The *linek* arguments must be delimited by commas. The k character in the *linek* arguments is a symbolic index that corresponds to the numerical order of the line numbers entered as arguments.

EXAMPLES:

```
1.  1 X = 1
    2      Y = 1
    3      FOR I = 1 TO 4
    4          ON I GOSUB 8,11,14,17
    5          PRINT "X = ";X,"Y = ";Y,"Z = ";Z
    6      NEXT I
    7  END
    8  REM      ADDITION
    9          Z = X + Y
   10          RETURN
   11 REM      SUBTRACTION
   12          Z = X - Y
   13          RETURN
   14 REM      MULTIPLICATION
   15          Z = X * Y
   16          RETURN
   17 REM      DIVISION
   18          Z = X / Y
   19          RETURN

X = 1   Y = 1   Z = 2
X = 1   Y = 1   Z = 0
X = 1   Y = 1   Z = 1
X = 1   Y = 1   Z = 1
```


ON GOTO/ON GO TO

DESCRIPTION:

Directs the program to a line designated by the value of a variable.

TYPE:

Command (Program Control)

SYNTAX:

ON *V* GOTO *line1* [, *line2* ...]

ARGUMENTS:

The *V* argument is the variable which designates the program line.

The *linek* arguments are the line numbers of the program lines.

RETURN VALUES:

None.

NOTES:

The ON GOTO command directs the program to a specific line based upon the value of a variable. The GOTO component of this command can be entered as one or two keywords (i.e., GOTO or GO TO).

The *V* argument must be a whole number. The range of values is $1 \leq V < \infty$. If the *V* argument is not a whole number, the decimal places in its value are truncated to produce a whole number. The line numbers to which the program is directed are designated by the *V* arguments. If *V* is 1, the program is directed to the line number designated by the *line1* argument. If *V* is 2, the program is directed to the line number designated by the *line2* argument. This procedure is followed for all values of the *V* argument and their corresponding *linek* arguments. Only the *line1* argument is required. The other *linek* arguments are optional. The *linek* arguments must be delimited by commas. The k character in the *linek* arguments is a symbolic index that corresponds to the numerical order of the line numbers entered as arguments.

EXAMPLES:

```
1.  1 X = 1
    2      Y = 1
    3      Z = 2
    4      ON Z GOTO 5,7,9,11
    5      Z = X + Y
    6      GOTO 12
    7      Z = X - Y
    8      GOTO 12
    9      Z = X*Y
   10      GOTO 12
   11      Z = X/Y
   12      PRINT "X = ";X,"Y = ";Y,"Z = ";Z

X = 1   Y = 1   Z = 0
```

OPEN FOR APPEND AS

DESCRIPTION:

Opens an existing file designated by a device number for writing data which is appended to the contents of the file.

TYPE:

Command (File Input/Output)

SYNTAX:

OPEN *file-name*\$ FOR APPEND AS [#]*device-number*

ARGUMENTS:

The *file-name*\$ argument is the name of the file.

The *device-number* argument is the device number.

RETURN VALUES:

None.

NOTES:

The file designated by the *file-name*\$ argument must exist. The *file-name*\$ argument must be a string. It can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "myfile.txt"). The original contents of the file remain intact. The data written to the file is appended to the existing file. The file is assigned a device number when opened. Valid device numbers are 0 through 15. Therefore, a maximum of 16 files can be open at any one time. The OPEN FOR APPEND AS # command must have a space between the OPEN FOR APPEND AS keywords and the pound (#) character. The pound (#) character is optional.

EXAMPLES:

```
1.  OPEN "myfile5.txt" FOR APPEND AS #15
    X = 11
    Y = 12
    Z = 13
    WRITE #15,X,Y,Z
```

Original contents of myfile5.txt:

```
1
2
3
```

Modified contents of myfile5.txt:

```
1
2
3
11
12
13
```

OPEN FOR INPUT AS

DESCRIPTION:

Opens an existing file designated by a device number for reading data from the file.

TYPE:

Command (File Input/Output)

SYNTAX:

OPEN *file-name\$* FOR INPUT AS [#]*device-number*

ARGUMENTS:

The *file-name\$* argument is the name of the file.

The *device-number* argument is the device number.

RETURN VALUES:

None.

NOTES:

The file designated by the *file-name\$* argument must exist. The *file-name\$* argument must be a string. It can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "myfile.txt"). The file is assigned a device number when opened. Valid device numbers are 0 through 15. Therefore, a maximum of 16 files can be open at any one time. The OPEN FOR INPUT AS # command must have a space between the OPEN FOR INPUT AS keywords and the pound (#) character. The pound (#) character is optional.

EXAMPLES:

```
1.  OPEN "myfile6.txt" FOR INPUT AS #15
      INPUT #15,X,Y,Z
      PRINT X,Y,Z
```

1	2	3
---	---	---

Contents of myfile6.txt:

1
2
3

OPEN FOR OUTPUT AS

DESCRIPTION:

Opens an existing file designated by a device number for writing data over the contents in the file.

TYPE:

Command (File Input/Output)

SYNTAX:

OPEN *file-name*\$ FOR OUTPUT AS [#]*device-number*

ARGUMENTS:

The *file-name*\$ argument is the name of the file.

The *device-number* argument is the device number.

RETURN VALUES:

None.

NOTES:

The *file-name*\$ argument must be a string. It can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "myfile.txt"). The data written to the file replaces the data in the existing file. The original contents of the file are erased from memory. The file is assigned a device number when opened. Valid device numbers are 0 through 15. Therefore, a maximum of 16 files can be open at any one time. The OPEN FOR OUTPUT AS # command must have a space between the OPEN FOR OUTPUT AS keywords and the pound (#) character. The pound (#) character is optional.

EXAMPLES:

```
1.  OPEN "myfile7.txt" FOR OUTPUT AS #15
    X = 11
    Y = 12
    Z = 13
    WRITE #15,X,Y,Z
```

Original contents of myfile7.txt:

1
2
3

Modified contents of myfile7.txt:

11, 12, 13

POW

DESCRIPTION:

Returns the value of a number raised to a power.

TYPE:

Function (Mathematical)

SYNTAX:

`POW(X , Y)`

ARGUMENTS:

The *X* argument is the number to be raised to the power.

The *Y* argument is the power to which the number is to be raised.

RETURN VALUES:

The value of the *X* argument raised to the power of the *Y* argument.

NOTES:

The *X* and *Y* arguments must be real numbers. They can be constants, variables, functions, or expressions. The ranges of values are $-\infty < X < \infty$ and $-\infty < Y < \infty$.

The range of values for `POW(X,Y)` is $-\infty < \text{POW}(X,Y) < \infty$. The returned value of `POW` has the same precision as its arguments.

EXAMPLES:

```
1.  X = 2
    Y = 3
    Z = POW (X, Y)
    PRINT X, Y, Z

    2          3          8
```

PRINT

DESCRIPTION:

Prints numbers and/or strings to the LightTools Console window and the LightTools log file.

TYPE:

Command (Interactive Input/Output)

SYNTAX:

```
PRINT AI [ , | ; | + A2 ... ]
```

ARGUMENTS:

The *Ak* arguments are the numbers and/or strings.

RETURN VALUES:

None.

NOTES:

The *Ak* arguments can be constants, variables, functions, or expressions. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). Only the *AI* argument is required. The other *Ak* arguments are optional. The *Ak* arguments must be delimited by a comma (prints tabbed output), a semicolon (prints immediate sequential output), or a plus sign (prints immediate sequential output by string concatenation). Arguments delimited by blank spaces or tabs are not supported. The *k* character in the *Ak* arguments is a symbolic index that corresponds to the numerical order of the items entered as arguments.

EXAMPLES:

- ```
1. PRINT "MACRO",123

 MACRO 123
```
- ```
2.  S$ = "MACRO"
     X = 123
     PRINT S$;X

      MACRO123
```
- ```
3. S$ = "MACRO"
 T$ = "123"
 X = 100
 Y = 20
 Z = 3
 PRINT S$+T$,X+Y+Z

 MACRO123 123
```



# PRINT USING

## DESCRIPTION:

Prints numbers and/or strings to the LightTools Console window and the LightTools log file using a defined format.

## TYPE:

Command (Interactive Input/Output)

## SYNTAX:

PRINT USING *format-string* ; *A1* [ , | ; *A2* ... ]

## ARGUMENTS:

The *format-string* argument is the format.

The *Ak* arguments are the numbers and/or strings.

## RETURN VALUES:

None.

## NOTES:

The *format-string* argument must be a string. It can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). There must be a format specification for each *Ak* argument within the *format-string* argument. The *format-string* argument can contain several different formatting characters. The formatting characters available with LightTools MACRO are as follows:

|      |                                                                                                        |
|------|--------------------------------------------------------------------------------------------------------|
| !    | Prints the first character of a string.                                                                |
| \\   | Prints 2 + X characters of a string, where X is the number of spaces between the backslash characters. |
| &    | Prints a variable length string.                                                                       |
| #    | Prints a single digit in a numeric value.                                                              |
| .    | Prints a decimal point in a numeric value.                                                             |
| +    | Prints the sign of a numeric value (positive or negative).                                             |
| -    | Prints a minus sign following a numeric value.                                                         |
| **   | Prints asterisks in the leading spaces of a numeric value.                                             |
| \$\$ | Prints a dollar sign preceding a numeric value.                                                        |
| ^^   | Prints a numeric value in exponential format.                                                          |
| _    | Prints the following character literally.                                                              |

The ***AK*** arguments can be constants, variables, functions, or expressions. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). Only the ***AI*** argument is required. The other ***AK*** arguments are optional. The ***AK*** arguments must be delimited by a comma or semicolon. Arguments delimited by blank spaces, tabs, or plus signs are not supported. The k character in the ***AK*** arguments is a symbolic index that corresponds to the numerical order of the items entered as arguments.

## EXAMPLES:

1. PRINT USING "#####.#####";123

**123.00000**

2. PRINT USING "^^,^^";123,321

**1.230000e+002,3.210000e+002**

3. F\$ = "\\ "  
PRINT USING F\$;"ABCDE"

**AB**

# PRINT #

## DESCRIPTION:

Prints numbers and/or strings to a file designated by a device number.

## TYPE:

Command (File Input/Output)

## SYNTAX:

PRINT #*device-number* , *A1* [ , | ; | + *A2* ... ]

## ARGUMENTS:

The *device-number* argument is the device number.

The *Ak* arguments are the numbers and/or strings.

## RETURN VALUES:

None.

## NOTES:

The file designated by the *device-number* argument must be open. The file is assigned a device number when opened. A valid device number specification is a number preceded by the pound (#) character. Valid device numbers are 0 through 15. Therefore, a maximum of 16 files can be open at any one time. A device number of -1 prints the numbers and/or strings in the LightTools Console window. The PRINT # command must have a space between the PRINT keyword and the pound (#) character.

Only the *device-number* and *A1* arguments are required. The other *Ak* arguments are optional. The *Ak* arguments can be constants, variables, functions, or expressions. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). The *Ak* arguments must be delimited by a comma (prints tabbed output), a semicolon (prints immediate sequential output), or a plus sign (prints immediate sequential output by string concatenation). Arguments delimited by blank spaces or tabs are not supported. The k character in the *Ak* arguments is a symbolic index that corresponds to the numerical order of the items entered as arguments.

## EXAMPLES:

1. OPEN "myfile8.txt" FOR OUTPUT AS #15  
PRINT #15,"X = 1","Y = 2","Z = 3"

Original contents of myfile8.txt:

1  
2  
3

Modified contents of myfile8.txt:

X = 1          Y = 2          Z = 3

# PRINT # USING

## DESCRIPTION:

Prints numbers and/or strings to a file designated by a device number using a defined format.

## TYPE:

Command (File Input/Output)

## SYNTAX:

PRINT #*device-number* , USING *format-string*\$ ; *A1* [ , | ; *A2* ... ]

## ARGUMENTS:

The *device-number* argument is the device number.

The *format-string*\$ argument is the format.

The *Ak* arguments are the numbers and/or strings.

## RETURN VALUES:

None.

## NOTES:

The file designated by the *device-number* argument must be open. A valid device number specification is a number preceded by the pound (#) character. Valid device numbers are 0 through 15. Therefore, a maximum of 16 files can be open at any one time. A device number of -1 prints the output text in the LightTools Console window. The PRINT # USING command must have a space between the PRINT USING keywords and the pound (#) character.

The *format-string*\$ argument must be a string. It can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). There must be a format specification for each *Ak* argument within the *format-string*\$ argument. The *format-string*\$ argument can contain several different formatting characters. The formatting characters available with LightTools MACRO are as follows:

|    |                                                                                                        |
|----|--------------------------------------------------------------------------------------------------------|
| !  | Prints the first character of a string.                                                                |
| \\ | Prints 2 + X characters of a string, where X is the number of spaces between the backslash characters. |
| &  | Prints a variable length string.                                                                       |
| #  | Prints a single digit in a numeric value.                                                              |
| .  | Prints a decimal point in a numeric value.                                                             |
| +  | Prints the sign of a numeric value (positive or negative).                                             |
| -  | Prints a minus sign following a numeric value.                                                         |
| ** | Prints asterisks in the leading spaces of a numeric value.                                             |

|                   |                                                 |
|-------------------|-------------------------------------------------|
| <code>\$\$</code> | Prints a dollar sign preceding a numeric value. |
| <code>^^</code>   | Prints a numeric value in exponential format.   |
| <code>_</code>    | Prints the following character literally.       |

The ***Ak*** arguments can be constants, variables, functions, or expressions. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). Only the ***AI*** argument is required. The other ***Ak*** arguments are optional. The ***Ak*** arguments must be delimited by a comma or semicolon. Arguments delimited by blank spaces, tabs, or plus signs are not supported. The *k* character in the ***Ak*** arguments is a symbolic index that corresponds to the numerical order of the items entered as arguments.

## EXAMPLES:

1. `OPEN "myfile9.txt" FOR OUTPUT AS #15`  
`PRINT #15,USING "&,&,&";"X = 1","Y = 2","Z = 3"`

Original contents of myfile9.txt:

1  
2  
3

Modified contents of myfile9.txt:

X = 1,Y = 2,Z = 3

# RAD

## DESCRIPTION:

Returns the equivalent number of radians for an angle in degrees.

## TYPE:

Function (Mathematical)

## SYNTAX:

`RAD( X )`

## ARGUMENTS:

The *X* argument is the number of degrees in the angle for which the number of radians is returned.

## RETURN VALUES:

The number of radians in the *X* argument.

## NOTES:

The *X* argument must be a real number. It can be a constant, variable, function, or expression. It represents an angle in degrees. The range of values is  $0^\circ \leq X \leq 360^\circ$ .

The range of values for `RAD(X)` is  $0 \leq \text{RAD}(X) \leq 2\pi$ . The returned value of `RAD` has the same precision as its argument. The `RAD` function is the inverse of the `DEG` function. For example, `X = RAD( DEG(X) )`.

## EXAMPLES:

```
1. X = 5.555
 Y = RAD(X)
 PRINT X,Y

 5.5550000 0.096953
```

# RANDOMIZE

## DESCRIPTION:

Reseeds the pseudo-random number generator with a modified starting point.

## TYPE:

Command (Miscellaneous)

## SYNTAX:

RANDOMIZE *N*

## ARGUMENTS:

The *N* argument is the number used to reseed the pseudo-random number generator.

## RETURN VALUES:

None.

## NOTES:

The RANDOMIZE command reseeds the pseudo-random number generator with a number. This causes the starting point in the list of numbers from which a pseudo-random number is chosen to be modified. It can be used in conjunction with the RND function. Sequential RND functions return numbers from the list in the sequential order in which they are listed. The reseeding of the pseudo-random number generator causes sequential RND functions to return a different sequence of pseudo-random numbers than the previously returned sequence of pseudo-random number. The sequential order of the numbers in the list of pseudo-random numbers remains intact. Only the starting point in the list of pseudo-random numbers is modified. Therefore, if the same *N* argument is used in multiple RANDOMIZE commands, the same starting point in the list of pseudo-random numbers is generated in each case. Therefore, the RND function returns the same pseudo-random number in each of these cases. The number used to reseed the pseudo-random number generator must be a whole number. The range of values for the *N* argument is  $-9999999999999999.999... \leq N \leq 9999999999999999.999...$ . It can be a constant, variable, function, or expression. If the *N* argument is not a whole number, the decimal places in its value are truncated to produce a whole number.

## EXAMPLES:

```
1. X = RND
 PRINT "X = ";X
 RANDOMIZE 333
 Y = RND
 PRINT "Y = ";Y
 RANDOMIZE -333
 Z = RND
 PRINT "Z = ";Z

 X = 0.7090976
 Y = 0.0343638
 Z = 0.9680166
```



# READ

## DESCRIPTION:

Reads numeric and/or string constants that are stored by the DATA command and assigns them to variables.

## TYPE:

Command (Program Control)

## SYNTAX:

```
READ V1 [, V2 ...]
```

## ARGUMENTS:

The *Vk* arguments are the variables.

## RETURN VALUES:

None.

## NOTES:

The READ command is only used in conjunction with DATA and RESTORE commands. The data type of the *Vk* arguments in the READ command must correspond to the data type of the *Ck* arguments in the DATA command in the order that they are specified. The *Vk* arguments can be numeric or string variables. The *Ck* arguments can have constant numeric and/or string values. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). Only the *V1* argument is required for the READ command. The other *Vk* arguments are optional. The *Vk* arguments must be delimited by commas. The k character in the *Vk* arguments is a symbolic index that corresponds to the numerical order of the variables entered as arguments.

## EXAMPLES:

```
1. DATA "ABC", 99
 READ S$, X
 PRINT S$, X

 ABC 99
```

# REM

## DESCRIPTION:

Designates a statement as a remark which is used for commentary purposes in a LightTools MACRO program.

## TYPE:

Command (Miscellaneous)

## SYNTAX:

REM [ *remark*\$ ]

## ARGUMENTS:

The *remark*\$ argument is the remark.

## RETURN VALUES:

None.

## NOTES:

The REM command has no functionality in the execution of the program. Its purpose is to provide annotation for the programming statements. It designates a program line as a comment. The program line is not executed. It is included in the program as a reference for program operations. It is only displayed in the actual program file. A program line containing the REM command is, however, counted as a program line for the purpose of tracking line numbers in a program.

The *remark*\$ argument is specified by a string. The *remark*\$ argument is optional. The REM command can be used with no argument to insert blank lines in a program. This provides a method for distinguishing program statement groupings in the program file. The first three characters of a remark statement must be the REM keyword, and it must be followed by one or more blank spaces. If a remark is to be continued on the following program line, the following program line must begin with the REM command.

## EXAMPLES:

```
1. 1 REM BRANCH ON Z
 2 X = 1
 3 Y = 1
 4 Z = 2
 5 ON Z GOTO 6,9,12,15
 6 REM ADDITION
 7 Z = X + Y
 8 GOTO 17
 9 REM SUBTRACTION
 10 Z = X - Y
 11 GOTO 17
 12 REM MULTIPLICATION
 13 Z = X*Y
 14 GOTO 17
 15 REM DIVISION
 16 Z = X/Y
 17 PRINT "X = ";X,"Y = ";Y,"Z = ";Z

X = 1 Y = 1 Z = 0
```

# RESTORE

## DESCRIPTION:

Resets the line and position counters employed by the DATA and READ commands.

## TYPE:

Command (Program Control)

## SYNTAX:

RESTORE [ *line* ]

## ARGUMENTS:

The *line* argument is the line number of the line to which the line and position counters are reset.

## RETURN VALUES:

None.

## NOTES:

The RESTORE command is only used in conjunction with the DATA and READ commands. The *line* argument is optional. If no *line* argument is specified, the line and position counters are reset to the beginning of the LightTools MACRO program file. If the *line* argument is specified, the line and position counters are reset to the beginning of the designated line. The line is specified by a line number.

## EXAMPLES:

```
1. DATA "ABC", 99
 READ S$, X
 PRINT S$, X
 DATA "XYZ", 11
 RESTORE
 READ S$, X
 PRINT S$, X
```

```
ABC 99
ABC 99
```

# RETURN

## DESCRIPTION:

Directs the program to a statement after a subprogram has been executed.

## TYPE:

Command (Program Control)

## SYNTAX:

RETURN

## ARGUMENTS:

None.

## RETURN VALUES:

None.

## NOTES:

The RETURN command directs the program to the statement immediately following the statement that directed the program to the subprogram. The RETURN command is the last statement in a sequence of statements which constitutes a subprogram which is not defined using the SUB command. It is used in conjunction with the GOSUB, ON GOSUB, and ON ERROR GOSUB commands.

## EXAMPLES:

```
1. ON Z GOSUB ADD, SUB, MUL, DIV
 PRINT Z
 END
 ADD:
 Z = X + Y
 RETURN
 SUB:
 Z = X - Y
 RETURN
 MUL:
 Z = X*Y
 RETURN
 DIV:
 Z = X/Y
 RETURN
```

{VALUE OF Z}

# RIGHT\$

## DESCRIPTION:

Returns a substring composed of the rightmost characters of a string.

## TYPE:

Function (String)

## SYNTAX:

RIGHT\$(*SS* , *N* )

## ARGUMENTS:

The *SS* argument is the string from which the substring is returned.

The *N* argument is the number of characters in the substring.

## RETURN VALUES:

A substring composed of the last *N* characters of the *SS* argument.

## NOTES:

The RIGHT\$ function creates a new string, and the string argument remains intact. The substring is composed of a set of characters starting at the right side (or end) of the string. The RIGHT\$ function is only used with strings. The RIGHT\$ function is analogous to the LEFT\$ and MID\$ functions.

The *SS* argument can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE").

The number of characters must be a whole number. The range of values for the *N* argument is

$-\infty < N \leq 255.999...$  . It can be a constant, variable, function, or expression. If the *N* argument is not a whole number, the decimal places in its value are truncated to produce a whole number. If the *N* argument is greater than the number of characters in the *SS* argument, the entire string is returned. If the *N* argument is less than or equal to 0, the null string ("" ) is returned.

## EXAMPLES:

```
1. S$ = "ABCDE"
 T$ = RIGHT$(S$, 3)
 PRINT S$, T$
```

**ABCDE    CDE**

```
2. S$ = "ABCDE"
 T$ = RIGHT$(S$, 6)
 PRINT S$, T$
```

**ABCDE    ABCDE**

# RND

## DESCRIPTION:

Returns a pseudo-random number from the pseudo-random number generator.

## TYPE:

Function (Mathematical)

## SYNTAX:

RND

## ARGUMENTS:

None.

## RETURN VALUES:

The pseudo-random number at the starting point in the list of pseudo-random numbers.

## NOTES:

The RND function is used to return a pseudo-random number from a list of numbers accessed by the pseudo-random number generator. Sequential RND functions return numbers from the list in the sequential order in which they are listed. The RND function can be used in conjunction with the RANDOMIZE command, which reseeds the pseudo-random number generator. This causes the starting point in the list of numbers from which pseudo-random numbers are chosen to be modified. The reseeding of the pseudo-random number generator causes sequential RND functions to return a different sequence of pseudo-random numbers than the previously returned sequence of pseudo-random number. When the RANDOMIZE command is executed, the sequential order of the numbers in the list of pseudo-random numbers remains intact. Only the starting point in the list of pseudo-random numbers is modified. Therefore, if the same *X* argument is used in multiple RANDOMIZE commands, the same starting point in the list of pseudo-random numbers is generated in each case. Therefore, the RND function returns the same pseudo-random number in each of these cases.

## EXAMPLES:

```
1. X = RND
 PRINT "X = ";X
 RANDOMIZE 333
 Y = RND
 PRINT "Y = ";Y
 RANDOMIZE -333
 Z = RND
 PRINT "Z = ";Z

 X = 0.7090976
 Y = 0.0343638
 Z = 0.9680166
```

# SGN

## DESCRIPTION:

Returns the mathematical sign of a number as a numeric value.

## TYPE:

Function (Mathematical)

## SYNTAX:

$\text{SGN}(X)$

## ARGUMENTS:

The  $X$  argument is the number whose mathematical sign is returned.

## RETURN VALUES:

The numeric value designated by the mathematical sign of the  $X$  argument.

## NOTES:

The  $X$  argument must be a real number. The range of values is  $-\infty < X < \infty$ .

The returned values are:

|    |                       |
|----|-----------------------|
| +1 | for $0 < X < \infty$  |
| -1 | for $-\infty < X < 0$ |
| 0  | for $X = 0$           |

## EXAMPLES:

```
1. X = 99
 Y = SGN(X)
 PRINT X, Y
```

```
99 1
```

```
2. X = -99
 Y = SGN(X)
 PRINT X, Y
```

```
-99 -1
```

```
3. X = 0
 Y = SGN(X)
 PRINT X, Y
```

```
0 0
```



# SIN

## DESCRIPTION:

Returns the sine of an angle which is in radians.

## TYPE:

Function (Mathematical)

## SYNTAX:

`SIN( X )`

## ARGUMENTS:

The *X* argument is the angle for which the sine is returned.

## RETURN VALUES:

The sine of the *X* argument.

## NOTES:

The sine is for the angle in radians represented by the argument of this function. The *X* argument must be a real number. It can be a constant, variable, function, or expression. The range of values is  $-\pi/2 \leq X \leq \pi/2$ .

The range of values for `SIN(X)` is  $-1 \leq \text{SIN}(X) \leq 1$ . The returned value of SIN has the same precision as its argument. The SIN function is the inverse of the ASIN function. For example,  $X = \text{SIN}(\text{ASIN}(X))$ .

## EXAMPLES:

```
1. X = 0.5
 Y = SIN(X)
 PRINT X,Y

 0.5000000 0.4794255

2. X = -0.5
 Y = SIN(X)
 PRINT X,Y

 -0.5000000 -0.4794255
```

# SPACE\$

## DESCRIPTION:

Returns a string with a designated length composed of blank spaces.

## TYPE:

Function (String)

## SYNTAX:

SPACE\$( [ *N* ] )

## ARGUMENTS:

The *N* argument is the number of blank spaces in the string (i.e., length).

## RETURN VALUES:

The string of blank spaces.

## NOTES:

The SPACE\$ function returns a string composed of a number of blank spaces specified by the *N* argument. The number of blank spaces must be a whole number. The range of values for the *N* argument is  $-0.999... \leq N \leq 255.999...$ . It can be a constant, variable, function, or expression. If the *N* argument is not a whole number, the decimal places in its value are truncated to produce a whole number. If the *N* argument is 0, the null string ("" ) is returned. The *N* argument is optional. If it is not specified, the null string ("" ) is returned.

## EXAMPLES:

```
1. X = 1
 S$ = SPACE$(5)
 Y = 2
 PRINT X;S$;Y
```

```
1 2
```

# SQR

## DESCRIPTION:

Returns the square root of a number.

## TYPE:

Function (Mathematical)

## SYNTAX:

`SQR(X)`

## ARGUMENTS:

The *X* argument is the number for which the square root is returned.

## RETURN VALUES:

The square root of the *X* argument.

## NOTES:

The square root of a number is a number which when multiplied by itself produces the number for which it is the square root.

The *X* argument must be a real number. It can be a constant, variable, function, or expression. The range of values is  $0 \leq X < \infty$ .

The range of values for `SQR(X)` is  $0 \leq \text{SQR}(X) < \infty$ . The returned value of `SQR` has the same precision as its argument.

## EXAMPLES:

```
1. X = 99.99
 Y = SQR(X)
 PRINT X, Y

 99.9900000 9.9995000
```

# STR\$

## DESCRIPTION:

Returns a string which is the decimal (base 10) representation of a number.

## TYPE:

Function (String)

## SYNTAX:

STR\$(*X*)

## ARGUMENTS:

The *X* argument is the number for which the string is returned.

## RETURN VALUES:

The string which is the decimal (base 10) representation of the *X* argument.

## NOTES:

The *X* argument must be a real number. It can be a constant, variable, function, or expression. The range of values is  $-\infty < X < \infty$ .

## EXAMPLES:

```
1. S$ = STR$(3E+02)
 PRINT S$
```

**300**

```
2. X = 1
 Y = 2
 S$ = STR$(X + Y)
 PRINT X,Y,S$
```

**1            2            3**

# STRING\$

## DESCRIPTION:

Returns a string with a designated length composed of the first character of a string.

## TYPE:

Function (String)

## SYNTAX:

STRING\$( *N* , *S\$* )

## ARGUMENTS:

The *N* argument is the length of the string to be returned.

The *S\$* argument is the string whose first character is used.

## RETURN VALUES:

The string composed of the first character of the *S\$* argument.

## NOTES:

The STRING\$ function returns a string composed of *N* characters which are the first character of the *S\$* argument. The number of characters must be a whole number. The range of values for the *N* argument is  $-0.999... \leq N \leq 255.999...$ . It can be a constant, variable, function, or expression. If the *N* argument is not a whole number, the decimal places in its value are truncated to produce a whole number. If the *N* argument is 0, the entire string is returned.

The *S\$* argument can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE").

## EXAMPLES:

```
1. S$ = "ABCDE"
 T$ = STRING$(5, S$)
 PRINT S$, T$
```

**ABCDE    AAAAA**

```
2. T$ = STRING$(5, "ABCDE")
 PRINT T$
```

**AAAAA**

# SUB

## DESCRIPTION:

Initiates a subprogram. The END SUB command is used in conjunction with the SUB command to terminate the subprogram.

## TYPE:

Command (Program Control)

## SYNTAX:

```
SUB label$ [(parameter1 , ...)]
```

## ARGUMENTS:

The *label*\$ argument is the name of the subprogram.

The *parameterk* arguments are the parameters passed to the subprogram.

## RETURN VALUES:

None.

## NOTES:

The SUB command is the first statement in a sequence of statements which constitutes a subprogram. The subprogram is specified by a label. Once a subprogram is defined, it can be executed by entering its label and associated input and/or output parameters in a CALL command. The program is directed to the subprogram by the CALL command. The END SUB command is the last statement in a sequence of statements which constitutes a subprogram.

The *parameterk* arguments are optional. If the subprogram does not require input and/or output parameters, they are not required as arguments. Only the *label*\$ argument is required. The *parameterk* arguments are designated as local variables. The values assigned to these variables are only used in the execution of the subprogram. The *parameterk* arguments can be assigned numeric or string values. The values can be constants, variables, functions, or expressions. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). The *parameterk* arguments must be delimited by commas and enclosed within parentheses. The k character in the *parameterk* arguments is a symbolic index that corresponds to the numerical order of the parameters entered as arguments.

## EXAMPLES:

```
1. CALL MARINE (3, 2, XYZ)
 PRINT "XYZ = "; XYZ
 END

 SUB MARINE (X, Y, Z)
 Z = X/Y
 END SUB

 XYZ = 1.5000000
```

# SWAP

## DESCRIPTION:

Exchanges the values of two variables.

## TYPE:

Command (Miscellaneous)

## SYNTAX:

SWAP *V1* , *V2*

## ARGUMENTS:

The *V1* and *V2* arguments are the variables.

## RETURN VALUES:

None.

## NOTES:

The variables can be numeric or string variables. Their types must match. The *V1* and *V2* arguments are delimited by a comma.

## EXAMPLES:

```
1. V1 = 11
 V2 = 99
 PRINT V1,V2
 SWAP V1,V2
 PRINT V1,V2

 11 99
 99 11
```

# SYSTEM

## DESCRIPTION:

Allows execution of an operating system command during execution of a LightTools macro program.

## TYPE:

Command (Program Control)

## SYNTAX:

SYSTEM *SS*

## ARGUMENTS:

The *SS* argument is a string which is the command to be executed.

## RETURN VALUES:

None.

## NOTES:

When the SYSTEM command is executed, execution of the LightTools macro program pauses. This places the user at the operating system level from which the specified command is executed. Once the command has completed execution, the user is returned to LightTools and the macro program resumes. The *SS* argument is a character string and must be enclosed within double quotation marks.

The restrictions which apply to the SYSTEM command are as follows:

- Simultaneous execution of SYSTEM commands is not allowed.
- The operating system command may not be executed in the background.
- The operating system command may not be executed on a remote computer.
- The operating system command may not write data to the LightTools database.
- The operating system command may not direct any output to the LightTools window.
- If the operating system command causes the computer to lock, LightTools does not respond.
- The LightTools Interrupt Operation option is inactive during execution of a SYSTEM command.

## EXAMPLES:

```
1. SYSTEM "REN LENS.5.lts LENS.1.lts"
```



# TAN

## DESCRIPTION:

Returns the tangent of an angle which is in radians.

## TYPE:

Function (Mathematical)

## SYNTAX:

TAN(*X*)

## ARGUMENTS:

The *X* argument is the angle for which the tangent is returned.

## RETURN VALUES:

The tangent of the *X* argument.

## NOTES:

The tangent is for the angle in radians represented by the argument of this function. The *X* argument must be a real number. It can be a constant, variable, function, or expression. The range of values is  $-\pi/2 < X < \pi/2$ .

The range of values for TAN(*X*) is  $-\infty < \text{TAN}(X) < \infty$ . The returned value of TAN has the same precision as its argument. The TAN function is the inverse of the ATN function. For example,  $X = \text{TAN}(\text{ATN}(X))$ .

## EXAMPLES:

```
1. X = 0.4
 Y = TAN(X)
 PRINT X, Y

 0.4000000 0.4227932

2. X = -0.4
 Y = TAN(X)
 PRINT X, Y

 -0.4000000 -0.4227932
```

# TIME\$

## DESCRIPTION:

Returns the current time based on the computer's internal clock.

## TYPE:

Function (String)

## SYNTAX:

TIME\$()

## ARGUMENTS:

The () argument is the null character enclosed within parentheses.

## RETURN VALUES:

The current time.

## NOTES:

The returned value of the TIME\$ function is a string in the form "HH-MM-SS", where HH is the two digit hour, MM is the two digit minute, and SS is the two digit second.

## EXAMPLES:

```
1. S$ = TIME$()
 PRINT S$
```

**13:13:13**

# VAL

## DESCRIPTION:

Returns the numeric value of the leading numeric characters in a string.

## TYPE:

Function (String)

## SYNTAX:

VAL(*S\$*)

## ARGUMENTS:

The *S\$* argument is the string.

## RETURN VALUES:

The numeric value of the leading numeric characters of the *S\$* argument.

## NOTES:

The *S\$* argument can be a constant, variable, function, or expression. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE").

If the *S\$* argument does not contain a leading numeric character, a value of 0 is returned.

The VAL function is the converse of the STR\$ function. For example, for a numeric argument *X* in decimal representation, *X* = VAL( STR\$(*X*) ).

## EXAMPLES:

1.   S\$ = "123ABC"  
      X = VAL(S\$)  
      PRINT S\$,X  
  
      **123ABC 123**
2.   X = VAL("ABCDE")  
      PRINT X  
  
      **0**

# WEND

## DESCRIPTION:

Repeats a WHILE loop initiated by the WHILE command.

## TYPE:

Command (Program Control)

## SYNTAX:

WEND

## ARGUMENTS:

None.

## RETURN VALUES:

None.

## NOTES:

The WEND command is used in conjunction with the WHILE command to repeat the WHILE loop. The WEND command is the last statement in a sequence of statements which constitutes a WHILE loop. When the WHILE loop is terminated, the program is directed to the statement immediately following the WEND command.

## EXAMPLES:

```
1. FOR I = 1 TO 5
 X = -3
 X = X + I
 WHILE X < 0
 X = ABS(X)
 WEND
 PRINT "X = ";X
NEXT I

X = 2
X = 1
X = 0
X = 1
X = 2
```

# WHILE

Initiates a WHILE loop which is used to perform a sequence of program statements in a repetitive fashion while a defined criterion is met. The WEND command is used in conjunction with the WHILE command to repeat the WHILE loop.

## TYPE:

Command (Program Control)

## SYNTAX:

WHILE *criterion*

## ARGUMENTS:

The *criterion* argument is the criterion.

## RETURN VALUES:

None.

## NOTES:

The WHILE command is used to perform a sequence of program statements in a repetitive fashion while a defined criterion is met. The criterion is entered as an argument on the command line. Each time a statement containing the quantity for which the criterion was defined is encountered, a test is made to determine if the criterion has been met. If it has been met, the sequence of program statements is repeated. If it has not been met, the WHILE loop is terminated.

The WHILE command is the first statement in a sequence of statements which constitutes a WHILE loop. The WEND command is the last statement in a sequence of program statements which constitutes a WHILE loop. When the WHILE loop is terminated, the program is directed to the statement immediately following the WEND command.

## EXAMPLES:

```
1. FOR I = 1 TO 5
 X = -3
 X = X + I
 WHILE X < 0
 X = ABS(X)
 WEND
 PRINT "X = ";X
NEXT I
```

```
x = 2
x = 1
x = 0
x = 1
x = 2
```

# WIDTH #

## DESCRIPTION:

Specifies the width of a file designated by a device number in number of characters.

## TYPE:

Command (File Input/Output)

## SYNTAX:

WIDTH #*device-number* , *N*

## ARGUMENTS:

The *device-number* argument is the device number.

The *N* argument is the number of characters per line.

## RETURN VALUES:

None.

## NOTES:

The WIDTH # command formats a file so that it has a designated number of characters per line. The file designated by the *device-number* argument must be open. A valid device number specification is a number preceded by the pound (#) character. Valid device numbers are 0 through 15. Therefore, a maximum of 16 files can be open at any one time. A device number of -1 specifies the number of characters per line for the LightTools Console window. The WIDTH # command must have a space between the WIDTH keyword and the pound (#) character.

The number of characters per line must be a whole number. The range of values for the *N* argument is  $1 \leq N \leq 255.999...$ . It can be a constant, variable, function, or expression. If the *N* argument is not a whole number, the decimal places in its value are truncated to produce a whole number.

## EXAMPLES:

```
1. OPEN "myfile10.txt" FOR OUTPUT AS #15
 WIDTH #15,2
 X = 111
 Y = 112
 Z = 113
 WRITE #15,X,Y,Z
```

Original contents of myfile10.txt:

```
1
2
3
```

Modified contents of myfile10.txt:

```
1
11
,
11
2,
1
13
```

# WRITE

## DESCRIPTION:

Writes numbers and/or strings to the LightTools Console window and the LightTools log file.

## TYPE:

Command (Interactive Input/Output)

## SYNTAX:

WRITE *A1* [ , *A2* ... ]

## ARGUMENTS:

The *Ak* arguments are the numbers and/or strings.

## RETURN VALUES:

None.

## NOTES:

The *Ak* arguments can be constants, variables, functions, or expressions. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). Only the *A1* argument is required. The other *Ak* arguments are optional. The *Ak* arguments must be delimited by a comma. Arguments delimited by semicolons, blank spaces or tabs are not supported. The k character in the *Ak* arguments is a symbolic index that corresponds to the numerical order of the items entered as arguments.

String values written to the Console window and log file are enclosed within double quotation marks. All values are delimited by commas within the Console window and log file.

## EXAMPLES:

1. WRITE "MACRO",123

"MACRO",123

2. S\$ = "MACRO"

X = 123

WRITE S\$,X

"MACRO",123

3. S\$ = "MACRO"

T\$ = "123"

X = 100

Y = 20

Z = 3

WRITE S\$+T\$,X+Y+Z

"MACRO123",123



# WRITE #

## DESCRIPTION:

Writes numbers and/or strings to a file designated by a device number.

## TYPE:

Command (File Input/Output)

## SYNTAX:

WRITE #*device-number* , *A1* [ , *A2* ... ]

## ARGUMENTS:

The *device-number* argument is the device number.

The *Ak* arguments are the numbers and/or strings.

## RETURN VALUES:

None.

## NOTES:

The file designated by the *device-number* argument must be open. A valid device number specification is a number preceded by the pound (#) character. Valid device numbers are 0 through 15. Therefore, a maximum of 16 files can be open at any one time. The WRITE # command name must have a space between the WRITE keyword and the pound (#) character.

Only the *device-number* and *A1* arguments are required. The other *Ak* arguments are optional. The *Ak* arguments can be constants, variables, functions, or expressions. Constant string arguments must be enclosed within double quotation marks (e.g., "ABCDE"). All arguments must be delimited by commas. Arguments delimited by semicolons, blank spaces or tabs are not supported. The k character in the *Ak* arguments is a symbolic index that corresponds to the numerical order of the items entered as arguments.

String values written to the file are enclosed within double quotation marks. All values are delimited by commas within the file.

## EXAMPLES:

```
1. OPEN "myfile11.txt" FOR OUTPUT AS #15
 X = 11
 Y = 12
 Z = 13
 WRITE #15,X,Y,Z
```

Original contents of myfile11.txt:

1  
2  
3

Modified contents of myfile11.txt:

11,12,13

## Chapter 5 MACRO by Example

This exercise demonstrates the use of the LightTools MACRO feature to construct, modify, analyze, and process a LightTools model. During the course of this process, you will create a mirror, modify the mirror's optical properties, access the mirror's optical properties via the LightTools database, trace rays, perform an illumination simulation, and export the simulation results to a file. Each step in this process is illustrated with examples of LightTools MACRO commands and functions, and their use to perform the desired operations.

### STEP 1: Create a Mirror Using LTCMD

The simplest form of a LightTools macro is one that duplicates the actions of an interactive user. During an interactive LightTools session, the mouse is used to select menu items, click buttons to build objects, select data fields in dialog boxes, etc. Several of these actions also build LightTools commands that are shown at the command line, some requiring input from the user. All of these mouse actions and commands are also written to the current LightTools recover file (.ltr). The LTCMD command allows you to include these commands in a macro. You simply perform all of the desired operations in LightTools, and then edit the recover file in such a manner that is suitable for use as a macro program. For example, a LightTools session which simply opens a 3D Design view and creates a mirror, as shown in the following figure, would create the following .ltr file:

```
#LightTools Version 8.5.0 (Mon Dec 12 16:52:21 PST 2018)
\VConsole
New3DDesign
\V3D_untitled
Mirror4Pt XYZ 0,0,0 XYZ 0,1,-0.5 XYZ 0,1,0 XYZ 0,0,0.5 XYZ 0,0,0.5
```



Figure 6. Concave mirror created with the Mirror4Pt command

The **\VConsole** and **\V3D\_untitled** commands are window commands which access the Console and 3D Design view windows, respectively. The **New3DDesign** command opens a new 3D Design view. The **Mirror4Pt** command creates a mirror. The recover file can now be edited to create a macro to perform the same operations. For example,

```
LTCMD "New3DDesign"
LTCMD "\V3D_untitled"
LTCMD "Mirror4Pt XYZ 0,0,0 XYZ 0,1,-0.5 XYZ 0,1,0 XYZ 0,0,0.5 XYZ 0,0,0.5"
```

## STEP 2: Modify the Mirror

If you want to modify the radius of the mirror in terms of its radius value rather than its curvature value, you must make sure that the LightTools system is operating in Radius mode.

Using the graphical user interface, the radius would be modified directly in the mirror's Properties dialog box. In a macro, you must access the lens surface parameter data directly in the database. To learn how to access the data directly in the database, first use the graphical user interface, then look at the commands in the recover file. This will give you the database access language and syntax.

```
LTCMD "\VConsole"
LTCMD "New3DDesign"
LTCMD "\V3D"
LTCMD "Mirror4Pt XYZ 0,0,0 XYZ 0,1,-0.5 XYZ 0,1,0 XYZ 0,0,0.5 XYZ 0,0,0.5"

LTCMD "\OLENS_MANAGER[untitled.1]"
LTCMD LTSTR$ ("Radius Mode") + "=Radius"
LTCMD "\Q"

LTCMD "\V3D"
LTCMD C + LTSTR$ ("Lens_1.tag_0")

LTCMD
"\OLENS_MANAGER[1].COMPONENTS[Components].SOLID[1].CIRC_LENS_PRIMITIVE[1].SPHERICAL_LENS_SURFACE[LensFrontSurface]"
LTCMD "Radius=2"
LTCMD "\Q"
```



**Figure 7. Concave mirror with modified front surface radius**

\OLENS\_MANAGER allows you to send messages to the lens manager where you can set the mode to radius. You can then terminate messages to the lens manager with \Q.

Now you can modify the radius of the front surface of the mirror. \OLENS\_MANAGER . . .  
SPHERICAL\_LENS\_SURFACE[LensFrontSurface] allows you to send messages to the lens manager and set the radius property of the front surface of the lens to 2.

### STEP 3: Modify the Mirror Using an Input Variable

You can also construct the macro program so that object modifications can be made in an interactive fashion. The **INPUT** command produces a dialog box which allows you to enter numerical data which is then assigned to a variable. The variable can then be used as input for the LightTools MACRO commands and functions. For example, the modification of the front surface radius of the mirror can be accomplished in the following manner:

```
INPUT "Enter the front surface radius of the mirror";Radius
```

The **INPUT** command produces the MACRO Input dialog box which allows you to enter the radius value. The message enclosed within quotation marks is displayed in the dialog box to inform you as to the values you are to enter. The radius value entered is assigned to the Radius variable. Now this variable can be used with database access to modify the radius value. The modified macro is shown below with the additional commands in bold type face.

```
INPUT "Enter the front surface radius of the mirror"; Radius

LTCMD "\VConsole"
LTCMD "New3DDesign"
LTCMD "\V3D"
LTCMD "Mirror4Pt XYZ 0,0,0 XYZ 0,1,-0.5 XYZ 0,1,0 XYZ 0,0,0.5 XYZ 0,0,0.5"

LTCMD "\OLENS_MANAGER[untitled.1]"
LTCMD LTSTR$("Radius Mode") + "=Radius"
LTCMD "\Q"

LTCMD "\V3D"
LTCMD "Select " + LTSTR$("Lens_1.tag_0")

LTCMD
"\OLENS_MANAGER[1].COMPONENTS[Components].SOLID[1].CIRC_LENS_PRIMITIVE[1].SPHERICAL_LENS_SURFACE[LensFrontSurface]"
LTCMD "Radius=" + STR$(Radius)
LTCMD "\Q"
```

### STEP 4: Modify the Mirror Using Data Access Filters

Using data access filters, the radius is modified by accessing its value in the LightTools database, and making the necessary changes to the value. For more details, see the section entitled Data Access Filters in Chapter 3, *LightTools Extensions to the BASIC Interpreter*. The modification of the radius value is done with the **LTDBSET** function.

Since the **LTDBSET** function returns a numeric value, the value must be assigned to a variable (i.e., FrontRadiusStatus). The returned value is indicative of the status of the function execution. The object key SOLID[1].SURFACE[1] points to the front surface of the mirror, and the data element Radius points to the radius value, which is set to the data value 2. Note that it is not necessary to set LightTools to Radius mode since the radius value is being accessed directly in the LightTools database.

Since the mirror is the only object in the LightTools model, the SOLID filter is superfluous at this point. However, if there were more than one object with surfaces in the LightTools model, you would have to consider the order in which the objects were created in order to modify the properties of a particular surface. This could be done by using the SURFACE filter with the SOLID filter, and an index number

which referred to the desired solid corresponding to the order in which the solids were created. Alternatively, this could be done with the SURFACE filter alone, and an index number which referred to the desired surface corresponding to the order in which the surfaces were created.

The values in the LightTools database can also be retrieved and printed within a macro program. The values are retrieved with the **LTDBGET** function. The value of the front surface radius of the mirror is assigned to the FrontRadius variable. This value can then be printed to the Console window and the LightTools log file (.log) using the **PRINT** command which prints the message enclosed within the quotation marks followed by the value of the FrontRadius variable.

```
LTCMD "\VConsole"
LTCMD "New3DDesign"
LTCMD "\V3D"
LTCMD "Mirror4Pt XYZ 0,0,0 XYZ 0,1,-0.5 XYZ 0,1,0 XYZ 0,0,0.5 XYZ 0,0,0.5"
FrontRadiusStatus = LTDBSET("SOLID[1].SURFACE[1]", "Radius", 2.0)
FrontRadius = LTDBGET("SOLID[1].SURFACE[1]", "Radius")
PRINT "The front surface radius of the mirror is ";FrontRadius
```

Now you will modify the reflectance value for the front surface of the mirror interactively using data access filters. The first step is to specify the reflectance value. This can be done with the **INPUT** command which produces the MACRO Input dialog box allowing you to enter the desired reflectance value. This value is assigned to the Reflectance variable. Now you can modify the reflectance value in the database. This is done with the **LTDBSET** function. The reflectance of the Reflect/Transmit amplitude zone of the front bare surface of the mirror is now set to the value of the Reflectance variable. The modified macro is shown below with the additional commands in bold type face.

```
LTCMD "\VConsole"
LTCMD "New3DDesign"
LTCMD "\V3D"
LTCMD "Mirror4Pt XYZ 0,0,0 XYZ 0,1,-0.5 XYZ 0,1,0 XYZ 0,0,0.5 XYZ 0,0,0.5"
FrontRadiusStatus = LTDBSET("SOLID[1].SURFACE[1]", "Radius", 2.0)
FrontRadius = LTDBGET("SOLID[1].SURFACE[1]", "Radius")
PRINT "The front surface radius of the mirror is ";FrontRadius
INPUT "Enter the reflectance value for the mirror";Reflectance
ReflectanceStatus = LTDBSET("ZONE[zone_1]", "Reflectance Percent", Reflectance)
if (ReflectanceStatus <> 0) then PRINT
"ReflectanceStatus=";ReflectanceStatus
PRINT "Set Reflectance Percent to "; LTDBGET("ZONE[zone_1]", "Reflectance
Percent")
```

## STEP 5: Trace Rays and Output Ray Data

Now you will trace a fan of rays and print data for the rays to a file. The ray fan is traced with the **NSFanFromPoint** command, as shown in the following figure. Notice that all of the rays in the fan are reflected back to their point of origin. This is due to the fact that the point of origin of the ray fan is at the center of curvature of the mirror. Now you can write ray data to a file. The first step is to open the file. This is done with the **OPEN FOR OUTPUT AS #** command which opens the file RAYDATA.dat as an output file capable of receiving data from the macro program. The file is assigned a device number of 1. Now you can create the header for the table of ray data to be written to the file. This is done with the **PRINT #** command.

Now you can create a list of rays in the fan. This is done with the **LTDBLIST\$** function which returns the RayList\$ list key which allows you to access the individual rays in the fan. The list key points to a list of object keys which correspond to the rays in the fan. This approach is preferable when accessing data for several objects. If data is to be accessed for only a few objects, the use of data access filters to construct object keys may be preferable to creating and accessing lists of object keys.

The list is accessed with the **LTLISTNEXT\$** function which returns the object key for the next ray in the list. Since this is the first time the list has been accessed, the next ray in the list is the first ray in the list. Now you can loop through the list of rays and acquire the ray height, angle of incidence, and exit angle at the front surface of the mirror for each ray in the list. This is done with the **WHILE** and **WEND** commands which together constitute a WHILE loop. The **WHILE** command initiates the WHILE loop, and stipulates the condition for continuing the loop. The loop is repeated as long as the condition specified by the **WHILE** command is valid. The condition in this case is that the object key for the ray is not a null character string (i.e., the list of rays still has object keys which have not been accessed).

Now you can create a list of ray segments for each ray in the fan. This is also done with the **LTDBLIST\$** function. This function returns the RaySegmentList\$ list key which allows you to access the individual ray segments in the fan. In this case, you will access the last ray segment in the list. This is done with the **LTLISTLAST\$** function which returns the object key for the last ray segment in the list (i.e., the ray segment following the front surface of the mirror). This object key allows you to access the desired ray data for this ray segment. This is done with the **LTDBGET** function which assigns the Global Y, Angle of Incidence, and Exit Angle values to the Y, AngleIn, and AngleOut variables, respectively.

Now you can print the values of these variables to the file RAYDATA.dat. This is done with the **PRINT # USING** command which prints the values to the file using the format specified by the character string following the USING keyword.

Now you can access the next ray in the list of rays. This is done with the **LTLISTNEXT\$** function which returns the object key for the second ray in the list and the process of retrieving the data for this ray begins. Before the process continues, a test is performed to determine if this object key is not a null character string. If the object key passes the test, the WHILE loop continues. If the object key fails the test, the WHILE loop is terminated. This is done with the **WEND** command.

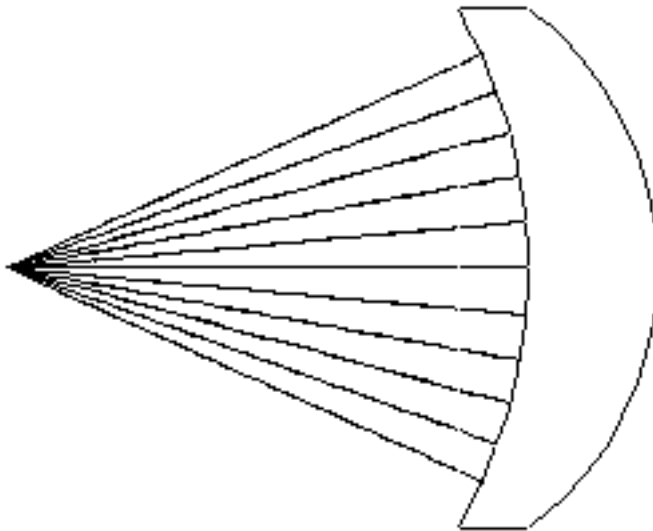
Now that the WHILE loop has been completed, and all of the ray data has been written to the file, you can close the file RAYDATA.dat. This is done with the **CLOSE #** command which closes the file and releases the device number 1, which can now be assigned to another file. The modified macro is shown below with the additional commands in bold type face.



```

LTCMD "\VConsole"
LTCMD "New3DDesign"
LTCMD "\V3D"
LTCMD "Mirror4Pt XYZ 0,0,0 XYZ 0,1,-0.5 XYZ 0,1,0 XYZ 0,0,0.5 XYZ 0,0,0.5"
FrontRadiusStatus = LTDBSET("SOLID[1].SURFACE[1]","Radius",2.0)
FrontRadius = LTDBGET("SOLID[1].SURFACE[1]","Radius")
PRINT "The front surface radius of the mirror is ";FrontRadius
INPUT "Enter the reflectance value for the mirror";Reflectance
ReflectanceStatus = LTDBSET("AMPLITUDE_ZONE[1]","Reflectance",Reflectance)
LTCMD "NSFanFromPoint XYZ 0,0,-2 XYZ 0,0.75,-0.333 XYZ 0,-0.75,-0.333"
OPEN "RAYDATA.dat" FOR OUTPUT AS #1
PRINT #1, " Y Inc. Angle Exit Angle "
PRINT #1, " ----- ----- "
RayList$ = LTDBLIST$("NS_RAYS[1]","NS_RAY")
RayKey$ = LTLISTNEXT$(RayList$)
WHILE RayKey$ <> ""
 RaySegmentList$ = LTDBLIST$(RayKey$,"NS_SEGMENT")
 SegmentKey$ = LTLISTLAST$(RaySegmentList$)
 Y = LTDBGET(SegmentKey$,"Global Y")
 AngleIn = LTDBGET(SegmentKey$,"Angle of Incidence")
 AngleOut = LTDBGET(SegmentKey$,"Exit Angle")
 PRINT #1, USING "###.### ###.### ###.### ";Y,AngleIn,AngleOut
 RayKey$ = LTLISTNEXT$(RayList$)
WEND
CLOSE #1

```



**Figure 8. Fan of non-sequential rays reflected from the front surface of the mirror**

#### Contents of RAYDATA.dat file:

| Y      | Inc. Angle | Exit Angle |
|--------|------------|------------|
| 0.821  | 0.000      | 0.000      |
| 0.677  | 0.000      | 0.000      |
| 0.521  | 0.000      | 0.000      |
| 0.354  | 0.000      | 0.000      |
| 0.179  | 0.000      | 0.000      |
| -0.000 | 0.000      | 0.000      |
| -0.179 | 0.000      | 0.000      |
| -0.354 | 0.000      | 0.000      |
| -0.521 | 0.000      | 0.000      |
| -0.677 | 0.000      | 0.000      |
| -0.821 | 0.000      | 0.000      |

### STEP 6: Modify the Shape of the Mirror

Now you will change the shape of the front surface of the mirror. This is done with the **LTDBSET** function which sets the shape of the front surface of the mirror to Conic. The **PRIMITIVE** filter is used in this case because **FRONT SURFACE SHAPE** is a data element that is accessible with the **PRIMITIVE** filter. Both the **SOLID** and **PRIMITIVE** filters will point to the mirror, since both filters apply to the object type of the mirror. However, data elements are accessible by particular filters, and, in this case, modifying the shape of the mirror required using the **PRIMITIVE** filter. The **SOLID** filter was suitable in the previous commands because you were accessing data elements that were accessible with the **SURFACE** filter.

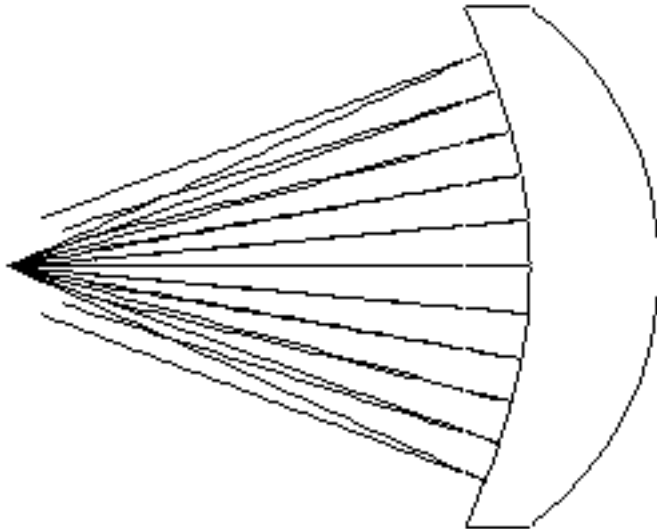
If there had been more than one type of primitive in the LightTools model, depending on the quantity and type of other primitives in the model, it would have been more efficient to use the **LENS\_PRIMITIVE** filter, or the **CIRC\_LENS\_PRIMITIVE** filter, if there were more than one type of lens primitive in the model. This is due to the fact that LightTools must perform a database search to find the data values of data elements for particular objects. If you begin the search by pointing LightTools in a more specific direction as far as object type is concerned, the search time will be held to a minimum. For more details, see the sections on filters in the appendices in the *LightTools MACRO Reference Guide*.

Now you can set the value of the conic constant of the conic surface of the mirror. This is done with the **LTDBSET** function which sets the conic constant of the front surface of the mirror to  $-1.0$ , which is the conic constant value for a parabola. Notice that the rays are now reflected at different angles, as shown in the following figure. Also, note that the **CONIC CONSTANT** data element is accessible with the **SURFACE** filter. The modified macro is shown below with the additional commands in bold type face.

```

LTCMD "\VConsole"
LTCMD "New3DDesign"
LTCMD "\V3D"
LTCMD "Mirror4Pt XYZ 0,0,0 XYZ 0,1,-0.5 XYZ 0,1,0 XYZ 0,0,0.5 XYZ 0,0,0.5"
FrontRadiusStatus = LTDBSET("SOLID[1].SURFACE[1]", "Radius", 2.0)
FrontRadius = LTDBGET("SOLID[1].SURFACE[1]", "Radius")
PRINT "The front surface radius of the mirror is ";FrontRadius
INPUT "Enter the reflectance value for the mirror";Reflectance
ReflectanceStatus = LTDBSET("AMPLITUDE_ZONE[1]", "Reflectance", Reflectance)
LTCMD "NSFanFromPoint XYZ 0,0,-2 XYZ 0,0.75,-0.333 XYZ 0,-0.75,-0.333"
OPEN "RAYDATA.dat" FOR OUTPUT AS #1
PRINT #1, " Y Inc. Angle Exit Angle "
PRINT #1, " ----- ----- "
RayList$ = LTDBLIST$("NS_RAYS[1]", "NS_RAY")
RayKey$ = LTLISTNEXT$(RayList$)
WHILE RayKey$ <> ""
 RaySegmentList$ = LTDBLIST$(RayKey$, "NS_SEGMENT")
 SegmentKey$ = LTLISTLAST$(RaySegmentList$)
 Y = LTDBGET(SegmentKey$, "Global Y")
 AngleIn = LTDBGET(SegmentKey$, "Angle of Incidence")
 AngleOut = LTDBGET(SegmentKey$, "Exit Angle")
 PRINT #1, USING "###.### ###.### ###.### ";Y,AngleIn,AngleOut
 RayKey$ = LTLISTNEXT$(RayList$)
WEND
CLOSE #1
SurfaceShapeStatus = LTDBSET("PRIMITIVE[1]", "FRONT SURFACE SHAPE", "Conic")
ConicConstantStatus = LTDBSET("SURFACE[1]", "CONIC CONSTANT", -1.0)

```



**Figure 9. Fan of non-sequential rays reflected from the front surface of the conic mirror**

## STEP 7: Run an Illumination Simulation

Now you will perform an illumination simulation. The first step is to create an illumination source. This is done with the **PtSource** command which creates a point source at a location with XYZ coordinates of 0, 0, -1. Now you can set the angular limits of the aim sphere of the point source so that the Monte Carlo ray trace will be aimed at the mirror. This is done with the **LTDBSET** function which sets the upper and lower angles of the aim sphere to 0° and 45°, respectively. When the point source was created, the Z axis of the aim sphere was coincident with the Z axis of the mirror and the Z axis of the global coordinate system. By setting the aim sphere angle limits to these values, the point source will emit a cone of rays with an apex angle of 90° whose axis is coincident with Z axis of the point source aim sphere, mirror, and global coordinate system.

Now you can add an illumination receiver to the LightTools model. This is done by creating an object and placing a surface receiver on the appropriate surface of the object. To create the object, you can use the **Sketch3PtLens** command which creates a plane parallel plate whose front surface faces the mirror. The front surface vertex is located at a point with XYZ coordinates of 0, 0, -5. The plate has a diameter of 4.0 and a thickness of 1.0. Now you can place a receiver on the front surface of the plate. This is done with the **SelectReceiver** command which places the receiver at the tag point of the plate with an index number of 0, which is on the front surface of the plate.

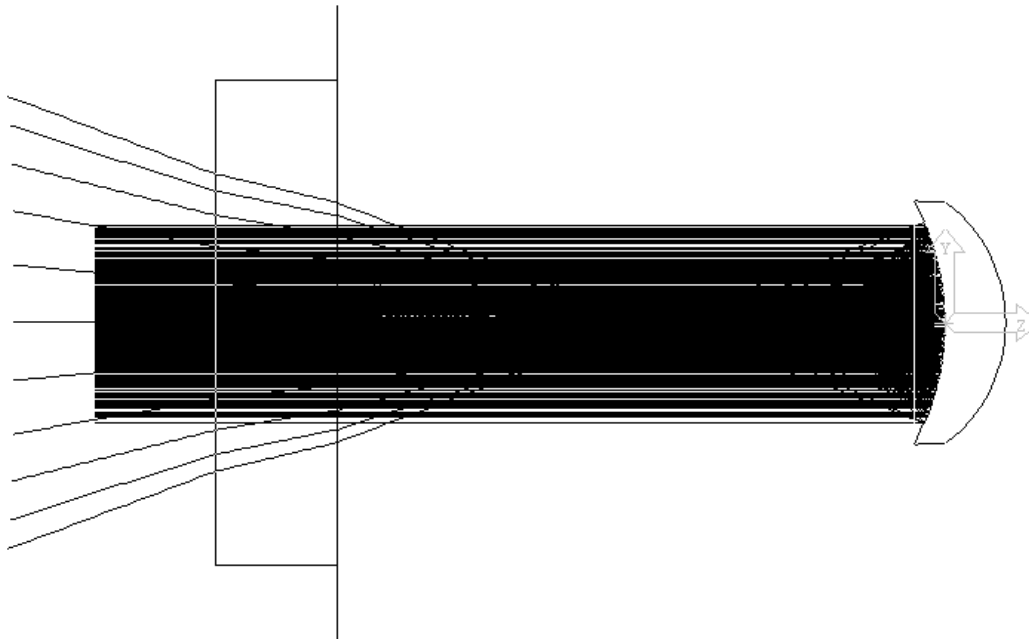
Now you can perform the illumination simulation. The first step is to set up the simulation. This is done with the **ForwardSim** command which initializes the illumination simulation to be performed in the forward direction. At this point, you can set the operating environment conditions for the simulation. For instance, to view the rays as they are traced, you can turn the ray preview on. This is done by changing the **Show Preview Rays** option in the Illumination Simulation Input dialog box. When Show Preview Rays is on, LightTools displays the rays as they are traced in the design view. The display is updated at the end of each one of the ray interrupt intervals. This interval, as well as the total number of rays traced, can be set to a desired value. This is also done in the Simulation Input dialog box. In order to make these changes to the forward illumination simulation setup via macro, you need to access the simulation object and set its properties accordingly. This example illustrates how to change the following properties: (1) set the total number of rays to be traced to 300 and (2) set the ray interrupt interval to 75. Each time 75 rays have been traced, the graphical display of the traced rays is updated.

Now you can return to the 3D Design view window. This is done with the **\V3D** window command. Now you can start the simulation. This is done with the **StartSim** command which starts the illumination simulation and performs a Monte Carlo ray trace originating at the point source. You will be able to see the rays in the 3D Design view window as they are traced. Notice that the rays reflected from the mirror are all parallel, as shown in the following figure. This is due to the fact that the point source is located at the focus of the mirror.

```

LTCMD "\VConsole"
LTCMD "New3DDesign"
LTCMD "\V3D"
LTCMD "Mirror4Pt XYZ 0,0,0 XYZ 0,1,-0.5 XYZ 0,1,0 XYZ 0,0,0.5 XYZ 0,0,0.5"
FrontRadiusStatus = LTDBSET("SOLID[1].SURFACE[1]", "Radius", 2.0)
SurfaceShapeStatus = LTDBSET("PRIMITIVE[1]", "FRONT SURFACE SHAPE", "Conic")
ConicConstantStatus = LTDBSET("SURFACE[1]", "CONIC CONSTANT", -1.0)
LTCMD "PtSource XYZ 0,0,-1"
AimUpperAngleStatus = LTDBSET("SOURCE[1].AIM_SPHERE[1]", "UPPER ANGLE", 0)
AimLowerAngleStatus = LTDBSET("SOURCE[1].AIM_SPHERE[1]", "LOWER ANGLE", 45)
LTCMD "Sketch3PtLens XYZ 0,0,-5 XYZ 0,2,-5 XYZ 0,0,-6 XYZ 0,0,-6"
LTCMD "\OSOLID[2].SURFACE[1]"
LTCMD LTSTR$("Add Receiver")+""=""
LTCMD "\Q"
LTCMD "Fit"
LTCMD "IllumInfo"
LTCMD "\OLENS_MANAGER[1].ILLUM_MANAGER[Illumination_Manager].SIMULATIONS[1]"
LTCMD LTSTR$("Generate Ray Preview ")+""=" yes"
LTCMD "MaxProgress = 300"
LTCMD "\Q"
LTCMD "StartSim"

```



**Figure 10. Illumination simulation with point source at the focus of the conic mirror**

## STEP 8: Move the Source and Rerun the Illumination Simulation

Now you will reposition the point source and rerun the illumination simulation. There are two ways to accomplish this: One is to select the point source and then use the move command. Use of the select command requires that you know the name of the object. The other way is to change the coordinates of

the existing point source directly in the database. Directly changing the coordinate property of the existing point source does not require that you know the object name, only that you know how to get access to the object. If you run this macro multiple times, the point source name may change, however if it is the only point source in the model, then the data access technique may be easier since there is only one point source. This example uses the direct database access technique.

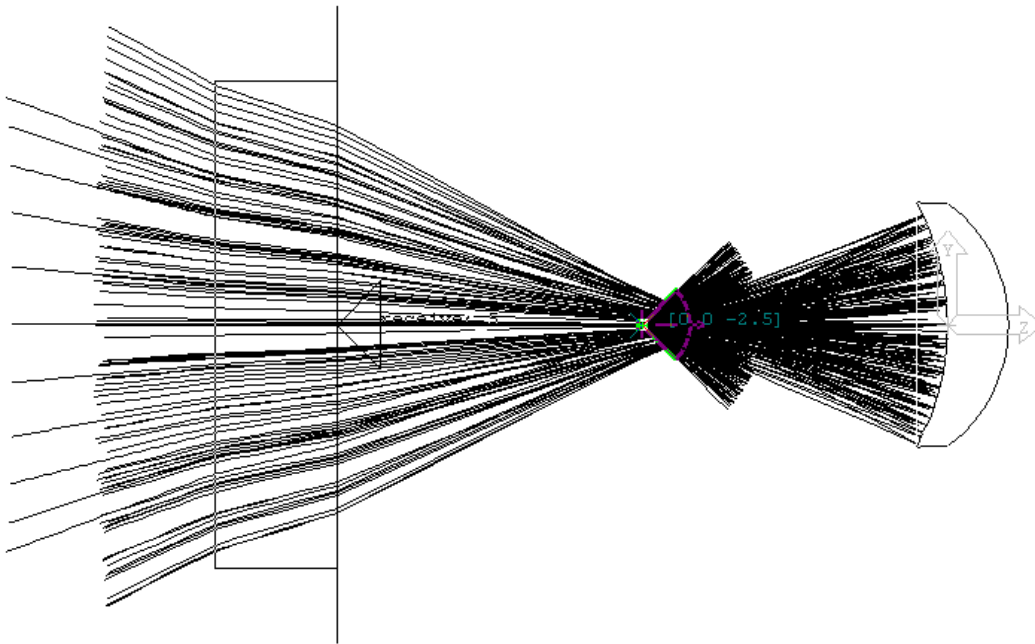
After moving the point source, you can rerun the simulation. This is done with the StartSim command. There is a RerunSim command available in LightTools which reruns an illumination simulation that has been previously performed. However, this command is only available if there have been no modifications made to the LightTools model. Because the point source was relocated, this command is not available at this point in the illumination process.

The StartSim command runs an illumination simulation with the point source at its new location. Notice that the rays reflected from the mirror are no longer parallel, as shown in the following figure. This is due to the fact that the point source is no longer at the focus of the mirror. Also notice that not all of the rays are reflected from the front surface of the mirror. This is due to the fact that the point source is now at a point far enough away from the mirror that the angle at which these rays are traced causes them to miss the mirror altogether. You will see that all of these rays terminate at a relatively short distance from the point source. This is due to the fact that these rays do not intersect any objects in the LightTools model and are therefore terminated at a predetermined distance from the point source.

```

LTCMD "\VConsole"
LTCMD "New3DDesign"
LTCMD "\V3D"
LTCMD "Mirror4Pt XYZ 0,0,0 XYZ 0,1,-0.5 XYZ 0,1,0 XYZ 0,0,0.5 XYZ 0,0,0.5"
FrontRadiusStatus = LTDBSET("SOLID[1].SURFACE[1]", "Radius", 2.0)
SurfaceShapeStatus = LTDBSET("PRIMITIVE[1]", "FRONT SURFACE SHAPE", "Conic")
ConicConstantStatus = LTDBSET("SURFACE[1]", "CONIC CONSTANT", -1.0)
LTCMD "PtSource XYZ 0,0,-1"
AimUpperAngleStatus = LTDBSET("SOURCE[1].AIM_SPHERE[1]", "UPPER ANGLE", 0)
AimLowerAngleStatus = LTDBSET("SOURCE[1].AIM_SPHERE[1]", "LOWER ANGLE", 45)
LTCMD "Sketch3PtLens XYZ 0,0,-5 XYZ 0,2,-5 XYZ 0,0,-6 XYZ 0,0,-6"
LTCMD "\OSOLID[2].SURFACE[1]"
LTCMD LTSTR$("Add Receiver")+""
LTCMD "\Q"
LTCMD "Fit"
LTCMD "IllumInfo"
LTCMD "\OLENS_MANAGER[1].ILLUM_MANAGER[Illumination_Manager].SIMULATIONS[1]"
LTCMD LTSTR$("Generate Ray Preview ") + "= yes"
LTCMD "MaxProgress = 300"
LTCMD "\Q"
LTCMD "\OSOURCE[1]"
LTCMD "z = -2.5"
LTCMD "\Q"
LTCMD "StartSim"

```



**Figure 11. Illumination simulation with the relocated point source**

## STEP 9: View the Illumination Simulation Data and Export to a File

Now you will view the results of the illumination simulation. In this case, you will examine the illuminance raster chart. This is done with the **MeshIllum** command which opens the 2DChart\_Illuminance\_Data window which contains the 2D Illuminance Raster Chart. To access the window, you can use the **\V2DChart\_Illuminance\_Data** window command which gives you access to the window which allows you to export the illumination simulation data to a file. The first step is to select the raster chart in the window. This is done with the **Select** command which selects the two dimensional pseudo-color display which represents the illumination data contained in the illuminance mesh of the surface receiver, as shown in the following figure.

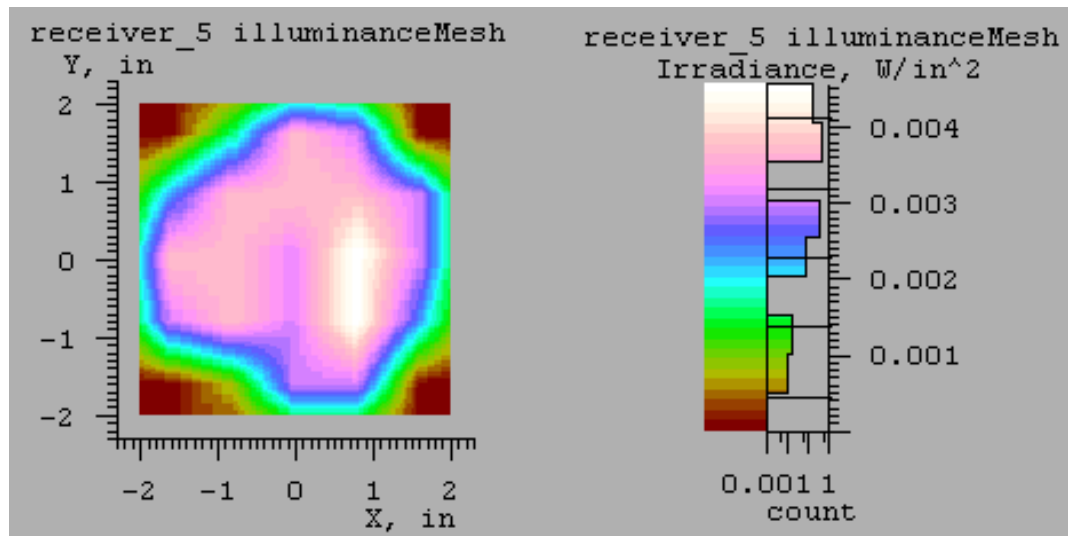
Now you can export this data to a file. This is done with the **Export** command which exports the illumination data in the selected chart to the specified file in text format. If this is the first version of the file, the exported file will be assigned a version number of 1. If a previous version of the file already exists, the exported file will be assigned the next highest version number. LightTools automatically assigns a .txt extension to these types of files.

```
LTCMD "\VConsole"
LTCMD "New3DDesign"
LTCMD "\V3D"
LTCMD "Mirror4Pt XYZ 0,0,0 XYZ 0,1,-0.5 XYZ 0,1,0 XYZ 0,0,0.5 XYZ 0,0,0.5"
FrontRadiusStatus = LTDBSET("SOLID[1].SURFACE[1]", "Radius", 2.0)
FrontRadius = LTDBGET("SOLID[1].SURFACE[1]", "Radius")
SurfaceShapeStatus = LTDBSET("PRIMITIVE[1]", "CONIC CONSTANT", -1.0)
AimUpperAngleStatus = LTDBSET("SOURCE[1].AIM_SPHERE[1]", "UPPER ANGLE", 0)
AimLowerAngleStatus = LTDBSET("SOURCE[1].AIM_SPHERE[1]", "LOWER ANGLE", 0)
LTCMD "Sketch3PtLens XYZ 0,0,-5 XYZ 0,2,-5 XYZ 0,0,-6 XYZ 0,0,-6"
LTCMD "\OSOLID[2].SURFACE[1]"
```

```

LTCMD "LTSTR$("Add Receiver ") + "="
LTCMD "\Q"
LTCMD "Fit"
LTCMD "IllumInfo"
LTCMD "\OLENS_MANAGER[1].ILLUM_MANAGER[Illumination_Manager].SIMULATIONS[1]"
LTCMD "LTSTR$("Generate Ray Preview ") + "= yes"
LTCMD "MaxProgress = 300"
LTCMD "\Q"
LTCMD "\OSOURCE[1]"
LTCMD "z = -2.5"
LTCMD "\Q"
LTCMD "BeginAllSimulations"
LTCMD "MeshIllum"
LTCMD "\V2DChart_Illuminance_Data"
LTCMD "Select " + LTSTR$("illuminance Mesh_shade.Chart-1.tag_-1")
LTCMD "Export " + LTSTR$(C:\LTUser\Chart")

```



**Figure 12. The Illuminance Raster Chart for the conic mirror with a point source**



# Appendix A Specific Data Access Filters

This table contains the filters that are used to access specific object types in the LightTools database. The filters are used to construct object keys or in conjunction with object keys for the data access techniques described in Chapter 3, *LightTools Extensions to the BASIC Interpreter*. The hierarchy of objects as represented in a Table view is reflected in the indentation of the filters. The object hierarchy is indicative of the level of containment that the objects have with respect to each other in the LightTools database.

## Filters for USER\_MATERIALS Database

- MATERIAL
- USER\_MATERIAL
  - LIBRARY\_MATERIAL\_SUMMARY
  - MATERIAL\_ABSORPTION
  - OPTICAL\_DENSITY\_ABSORPTION
  - TRANSMISSION\_ABSORPTION
  - EXTINCTION\_COEFFICIENT
  - WAVELENGTH\_DATA
  - MATERIAL\_INDEX
  - CAUCHY\_INDEX
  - LAURENT\_INDEX
  - GMLAURENT\_INDEX
  - GM\_SELLMEIER
  - HARTMANN\_INDEX
  - STANDARD\_SELLMEIER
  - INTERPOLATED\_INDEX
  - CONSTANT\_INDEX
  - SELFOC\_INDEX
  - USER\_GRIN
  - ANISOTROPIC\_INTERPOLATED\_INDEX
  - XY\_DATA
- GLASS\_CATALOG
  - GLASS\_MATERIAL

## Filters for PREFERENCES\_MANAGER Database

- CONTACT\_MANAGER
  - GENERAL\_PREFERENCES
  - AMBIENTMATERIAL\_PREFERENCES
- DEFAULT\_PREFERENCES
- MATERIAL\_PREFERENCES
- REPAIR\_GEOMETRY
- DEFAULT\_MATERIALS
- VIEW\_PREFERENCES

## Filters for PROPERTY\_MANAGER Database

- PROPERTY\_USAGE
- PROPERTY

**Filters for COMPONENTS Database**

- ENTITY
- RAYTRACEABLE
- SOLID
- SW\_PART\_SOLID
- SURFACE\_SET
- CADFILE\_SOLID
  - VOLUME\_SPATIAL\_GRID\_APODIZER
- GENERIC\_SOLID
- SWEPT\_SOLID
  - PRIMITIVE
  - LENS\_PRIMITIVE
  - CIRC\_LENS\_PRIMITIVE
  - RECT\_LENS\_PRIMITIVE
  - CUBE\_PRIMITIVE
  - SPHERE\_PRIMITIVE
  - ELLIPSE\_PRIMITIVE
  - ELLIPTICAL\_FIBER\_PRIMITIVE
  - TOROID\_PRIMITIVE
  - REVOLUTION\_PRIMITIVE
  - EXTRUSION\_PRIMITIVE
  - REGULAR\_POLYGON\_EXTRUSION\_PRIMITIVE
  - DOVE\_PRISM\_EXTRUSION\_PRIMITIVE
  - PENTA\_PRISM\_EXTRUSION\_PRIMITIVE
  - PORRO\_PRISM\_EXTRUSION\_PRIMITIVE
  - RTANGLE\_PRISM\_EXTRUSION\_PRIMITIVE
  - CROSS\_SECTION\_POLYGON
  - CROSS\_SECTION\_VERTEX
  - PROFILE\_CURVE
  - POLAR\_PROFILE\_CURVE
  - CIRCULAR\_PROFILE\_CURVE
  - ELLIPTICAL\_PROFILE\_CURVE
  - SQUARE\_PROFILE\_CURVE
  - RECT\_PROFILE\_CURVE
  - PROFILE\_CURVE\_LIST
  - POLAR\_PROFILE\_CURVE\_LIST
  - CIRCULAR\_PROFILE\_CURVE\_LIST
  - ELLIPTICAL\_PROFILE\_CURVE\_LIST
  - SQUARE\_PROFILE\_CURVE\_LIST
  - RECT\_PROFILE\_CURVE\_LIST
  - CURVE
  - POLYLINE\_CURVE
  - FITTED\_CURVE
  - BEZIER\_CURVE
  - CIRCULAR\_ARC
  - SWEPT\_PROFILE
  - SWEEP\_PATH

INTERPOLATED\_PATH  
WIREFRAME\_PATH  
HALF\_PLANE\_PRIMITIVE  
CYLINDER\_PRIMITIVE  
SKIN\_PRIMITIVE  
CPC\_PRIMITIVE  
CPCINVOLUTE\_PRIMITIVE  
FREEFORM\_PRIMITIVE  
SWEPT\_PRIMITIVE  
GENERIC\_PRIMITIVE  
SW\_PART\_PRIMITIVE  
CADFILE\_PRIMITIVE  
FUNGEO\_PRIMITIVE  
VOLUME\_INTERFACE  
ANISOTROPIC\_MATERIAL\_ORIENTATION  
VOLUME\_SCATTER\_LOG  
CS\_ATTRIBUTE  
POINT\_ATTRIBUTE  
POLYLINE\_ATTRIBUTE  
SW\_FEATURE  
SW\_COORDINATESYSTEM  
SW\_POINT  
SW\_REFAXIS  
SW\_WIREBODY  
SW\_PARAMETER  
SURFACE  
SURFACE\_EVALUATOR  
PLANAR\_SURFACE  
LENS\_SURFACE  
SPHERICAL\_LENS\_SURFACE  
CONIC\_LENS\_SURFACE  
ASPHERE\_LENS\_SURFACE  
POLYNOMIAL\_LENS\_SURFACE  
ODDPOLYNOMIAL\_LENS\_SURFACE  
ANAMORPHIC\_LENS\_SURFACE  
XYPOLYNOMIAL\_LENS\_SURFACE  
ZERNIKE\_LENS\_SURFACE  
SUPERCONIC\_LENS\_SURFACE  
XTOROID\_LENS\_SURFACE  
YTOROID\_LENS\_SURFACE  
XCYLINDER\_LENS\_SURFACE  
YCYLINDER\_LENS\_SURFACE  
SPLINESWEEP\_LENS\_SURFACE  
SPLINEPATCH\_LENS\_SURFACE  
CYLINDER\_SURFACE  
ELLIPTICAL\_FIBER\_SURFACE  
FULLSPHERE\_SURFACE  
FULLELLIPSOID\_SURFACE

FULLTOROID\_SURFACE  
GENERIC\_SURFACE  
MESH\_SURFACE  
SKIN\_SURFACE  
SWEPT\_SURFACE  
SWEPT\_FILLET\_SURFACE  
FREEFORM\_SURFACE  
    ZONE  
        DIRECTION\_ZONE  
        DOE\_DIRECTION\_ZONE  
        RADPOLY\_DOE\_DIRECTION\_ZONE  
        RULED\_DOE\_DIRECTION\_ZONE  
        XYPOLY\_DOE\_DIRECTION\_ZONE  
        SPECULAR\_DIRECTION\_ZONE  
        DOMINANT\_DIRECTION\_ZONE  
        FRESNEL\_DIRECTION\_ZONE  
        RADIAL\_FRESNEL\_DIRECTION\_ZONE  
        CYLINDRICAL\_FRESNEL\_DIRECTION\_ZONE  
        AMPLITUDE\_ZONE  
        FRESNEL\_AMPLITUDE\_ZONE  
        RT\_AMPLITUDE\_ZONE  
        QWAR\_COATING\_AMPLITUDE\_ZONE  
        THIN\_FILM\_STACK\_COATING\_AMPLITUDE\_ZONE  
            THIN\_FILM\_STACK\_EVALUATOR  
            THIN\_FILM\_LOCAL\_MATERIAL  
            THIN\_FILM\_THICKNESS\_SCALING  
        USER\_COATING\_AMPLITUDE\_ZONE  
        IDEAL\_POLARIZER\_AMPLITUDE\_ZONE  
        IDEAL\_RETARDER\_AMPLITUDE\_ZONE  
        JONES\_MATRIX\_AMPLITUDE\_ZONE  
        MUELLER\_MATRIX\_AMPLITUDE\_ZONE  
        SCATTER\_AMPLITUDE\_ZONE  
            MEASURED\_BSDF\_AMPLITUDE\_ZONE  
            ZONE\_EVALUATOR  
            SCATTER\_EVALUATOR  
            MICROFACET\_SCATTER\_DATA  
        COSN\_SCATTER\_AMPLITUDE\_ZONE  
        GAUSSIAN\_SCATTER\_AMPLITUDE\_ZONE  
        LAMBERTIAN\_SCATTER\_AMPLITUDE\_ZONE  
        AOI\_SCATTER\_AMPLITUDE\_ZONE  
        USER\_SCATTER\_AMPLITUDE\_ZONE  
        ELLIPTICAL\_GAUSSIAN\_SCATTER\_AMPLITUDE\_ZONE  
        MIXED\_SCATTER\_AMPLITUDE\_ZONE  
        USER\_DEFINED\_AMPLITUDE\_ZONE  
        DIFFRACTIVE\_OPTICAL\_PROPERTIES  
        DIFFRACTION\_EFFICIENCY\_ZONE  
        PARTICLE  
        LUMINESCENCE

MIE\_SCATTERER  
     MIE\_EVALUATOR  
 HG\_SCATTERER  
 GEGENBAUER\_SCATTERER  
 USER\_DEFINED\_VOLUME\_SCATTERER  
 USER\_DEFINED\_VOLUME\_SCATTERER\_SERVER  
 INERT\_PARTICLE  
     VOLUME\_SCATTER\_PARTICLE\_EVALUATOR  
 ZONE\_EXTENT  
 CIRC\_ZONE\_EXTENT  
 RECT\_ZONE\_EXTENT  
 2D\_RECT\_ARRAY\_ZONE\_EXTENT  
 2D\_RECT\_ARRAY\_ARCS\_ZONE\_EXTENT  
 2D\_RECT\_ARRAY\_CIRCLES\_ZONE\_EXTENT  
 2D\_RECT\_ARRAY\_ELLIPSES\_ZONE\_EXTENT  
 2D\_RECT\_ARRAY\_RECTS\_ZONE\_EXTENT  
 3D\_ARRAY\_ZONE\_EXTENT  
 3D\_HEX\_ARRAY\_ZONE\_EXTENT  
 3D\_RECT\_ARRAY\_ZONE\_EXTENT  
 3D\_SHIFT\_RECT\_ARRAY\_ZONE\_EXTENT  
 TEXTURE\_ZONE\_EXTENT  
     UNIT\_CELL  
         SPHERICAL\_UNIT\_CELL  
         CONE\_UNIT\_CELL  
         4\_SIDED\_PYRAMID\_UNIT\_CELL  
         PRISM\_UNIT\_CELL  
         CYLINDER\_UNIT\_CELL  
         LIBRARY\_ELEMENT\_UNIT\_CELL  
         SOLID\_HOLDER  
         RECTANGLE\_UNIT\_CELL  
         CIRCLE\_UNIT\_CELL  
         ELLIPSE\_UNIT\_CELL  
         TEXTURE\_PARAMETER  
         TEXTURE\_PLACEMENT  
         REFERENCE\_SURFACE  
         PLANAR\_REFERENCE\_SURFACE  
         UV\_REFERENCE\_SURFACE  
 CEMENTED\_INTERFACE  
 POLYLINE  
 WIREBODY  
 NURBS\_CURVE  
 INTERPOLATED\_CURVE  
     CURVE\_EVALUATOR  
 COORDINATE\_SYSTEM  
 POINT  
 STROKE\_TEXT  
 GROUP  
 SW\_ASSEMBLY

DUMMY\_SURFACE  
PLANE\_DUMMY\_SURFACE  
SPHERE\_DUMMY\_SURFACE  
MESH\_VIS

**Filters for SPECTRAL\_REGIONS Database**

SPECTRAL\_REGION  
WAVELENGTH

**Filters for NS\_RAYS Database**

NS\_FAN  
NS\_GRID  
NS\_RAY  
NS\_BRANCH  
NS\_SEGMENT  
REGIONS  
REGION

**Filters for OPT\_MANAGER Database**

OPT\_VARIABLES  
OPT\_VARIABLE  
OPT\_DBITEM  
OPT\_DBVARIABLE  
OPT\_USERVARIABLECOLLECTION  
OPT\_USERVARIABLE  
OPT\_CONSTRAINTS  
OPT\_CONSTRAINT  
OPT\_DBCONSTRAINT  
OPT\_USERCONSTRAINTCOLLECTION  
OPT\_USERCONSTRAINT  
OPT\_MERITFUNCTIONS  
OPT\_MERITFUNCTION  
OPT\_ABERRATIONVECTOR  
OPT\_MESHMERITFUNCTION  
OPT\_CIEMESHMERITFUNCTION  
OPT\_COLORDIFFMESHMERITFUNCTION  
OPT\_CCTMESHMERITFUNCTION  
OPT\_FOCUSCOLLIMATEMERITFUNCTION  
OPT\_FOCUSMERITFUNCTION  
OPT\_COLLIMATEMERITFUNCTION  
OPT\_SLICEMERITFUNCTION  
OPT\_POINTMERITFUNCTION  
OPT\_USERMERITFUNCTION  
OPT\_DBMERITFUNCTION  
OPT\_DBMFITEM  
OPT\_NSRAYMERITFUNCTION  
OPT\_NSRAYMFITEM  
OPT\_PENALTYCONSTRAINTMERITFUNCTION

**OPT\_RESULTS**

- OPT\_RESULTS\_VARIABLES
  - OPT\_RESULTS\_VARIABLE\_ITERATIONS
- OPT\_RESULTS\_CONSTRAINTS
  - OPT\_RESULTS\_CONSTRAINT\_ITERATIONS
- OPT\_RESULTS\_MERITFUNCTIONS
  - OPT\_RESULTS\_MERITFUNCTION\_ITERATIONS
  - OPT\_RESULTS\_MERITFUNCTION\_DATUM
  - OPT\_RESULTS\_MESHMERITFUNCTION\_DATUM
  - OPT\_RESULTS\_SLICEMERITFUNCTION\_DATUM
- OPT\_MESHMFRTARGETFUNCTION
- OPT\_HIRESMESHTARGETFUNCTION

**Filters for TOL\_MANAGER Database****TOL\_TOLERANCES**

- TOL\_DBTOLERANCE
- TOL\_POSITIONGROUPTOLERANCE
  - TOL\_POSITIONTOLERANCE
  - TOL\_POSITIONRADIALTOLERANCE
- TOL\_USERDEFINEDTOLERANCEGROUP
  - TOL\_USERDEFINEDTOLERANCE
  - TOL\_TOLERANCEINDIVIDUALSENSITIVITY
  - TOL\_TOLERANCEQFITSSENSITIVITY

**TOL\_PERFORMANCEMEASURES**

- TOL\_PERFORMANCEGROUP
- TOL\_PERFORMANCEMEASURE
- TOL\_PERFORMANCEGROUPITEM
- TOL\_DBPERFORMANCEMeasure
- TOL\_DBPERFORMANCEGROUPITEM
- TOL\_MESHPERFORMANCEMEASURE
- TOL\_CIEMESHPERFORMANCEMEASURE
- TOL\_NSRAYPERFORMANCEMEASURE
- TOL\_USERDEFINEDPERFORMANCEGROUP
  - TOL\_USERDEFINEDPERFORMANCEMEASURE

**TOL\_INDIVIDUALSENSITIVITY****TOL\_QFITSSENSITIVITY****TOL\_SIMULATION\_MANAGER**

- TOL\_SIMULATION
  - TOL\_SIMULATION\_RESULTS

**UI\_MANAGER****Filters for ILLUM\_MANAGER Database****RAYPATH\_CONTROL****SOURCES**

- SOURCE
- POINT\_SOURCE
- SURFACE\_SOURCE
- OBJECT\_SOURCE

- CUBE\_SURFACE\_SOURCE
- CYLINDER\_SURFACE\_SOURCE
- SPHERE\_SURFACE\_SOURCE
- TOROID\_SURFACE\_SOURCE
- EMBEDDED\_GEOM\_SOURCE
- DISK\_SOURCE
- RECT\_SOURCE
- LUCIDSHAPE\_SOURCE
- LUCIDSHAPE\_POINT\_SOURCE
- SURFACE\_EMITTER
- EMITTER
  - NATIVE\_EMITTER
  - AREA\_EMITTER
    - DIRECTION\_GRID\_APODIZER
    - DIRECTION\_GAUSSIAN\_APODIZER
    - SURFACE\_GRID\_APODIZER
    - SURFACE\_GAUSSIAN\_APODIZER
- VOLUME\_SOURCE
- CUBE\_VOLUME\_SOURCE
- CYLINDER\_VOLUME\_SOURCE
- SPHERE\_VOLUME\_SOURCE
- TOROID\_VOLUME\_SOURCE
  - VOLUME\_GRID\_APODIZER
  - VOLUME\_CYLINDER\_APODIZER
- RAY\_DATA\_SOURCE
- RAY\_DATA\_SOURCE\_COLLECTION\_SURFACE
  - RAY\_DATA\_SOURCE\_NULL\_COLLECTION\_SURFACE
  - RAY\_DATA\_SOURCE\_PLANAR\_COLLECTION\_SURFACE
  - RAY\_DATA\_SOURCE\_SPHERE\_COLLECTION\_SURFACE
- LS\_RAY\_DATA\_SOURCE
- AIM
  - AIM\_SPHERE
  - AIM\_AREA
    - CIRCULAR\_AIM\_AREA
    - RECTANGULAR\_AIM\_AREA
    - POLYGON\_AIM\_AREA
      - CLOSED\_POLYGON
    - SURFACEBASED\_AIM\_AREA
  - AIM\_PATCH
  - AIM\_CONE
  - AIM\_TARGET\_SPHERE
- RECEIVERS
  - RECEIVER
  - SURFACE\_RECEIVER
  - SOLID\_RECEIVER
  - PRIMITIVE\_RECEIVER
  - FARFIELD\_RECEIVER
    - SPECTRAL\_DISTRIBUTION



```

FILTER_LIST
 RECEIVER_FILTER
 RAY_MAGNITUDE_FILTER
 ELEMENT_FILTER
 EXIT_ANGLE_FILTER
 HIT_NUMBER_FILTER
 INCIDENT_ANGLE_FILTER
 SOURCE_FILTER
 SURFACE_FILTER
 WAVELENGTH_FILTER
 ORDINAL_NUMBER_FILTER
 PROPERTY_ZONE_FILTER
 OPTICAL_PROPERTY_FILTER
 OPTICAL_PROPERTY_AFTER_FILTER
 PATH_TRANSMITTANCE_FILTER
 OPTICAL_PATH_LENGTH_FILTER
 POLARIZATION_FILTER
 LINEAR_POLARIZER_FILTER
 AFTER_SURFACE_FILTER
 AFTER_ELEMENT_FILTER
 AFTER_PROPERTY_ZONE_FILTER
 VOLUME_INTERFACE_FILTER
 AFTER_VOLUME_INTERFACE_FILTER
 RAYPATH_FILTER
 LIGHT_GUIDE_FILTER
 ANGULAR_CONE_FILTER
 NEAR_SPECULAR_COUNT_FILTER
 DIFFUSE_COUNT_FILTER
 SCATTER_COUNT_FILTER
 SPECULAR_COUNT_FILTER
 REFLECTANY_COUNT_FILTER
 REFLECTONLY_COUNT_FILTER
 REFRACT_COUNT_FILTER
 TIR_COUNT_FILTER
 FILTER_GROUP
SLICE
MESH
INTENSITY_MESH
 INTENSITY_APE_LIST
ILLUMINANCE_MESH
 ILLUMINANCE_APE_LIST
SPATIAL_LUMINANCE_MESH
ANGULAR_LUMINANCE_MESH
 ENCIRCLED_ENERGY_APE
 CIRC_ENCIRCLED_ENERGY_APE
 SLIT_ENCIRCLED_ENERGY_APE
 OVAL_ENCIRCLED_ENERGY_APE
 RECT_ENCIRCLED_ENERGY_APE

```

SQUARE\_ENCIRCLED\_ENERGY\_APE  
 SPATIAL\_CIE\_MESH  
 ANGULAR\_CIE\_MESH  
 SPATIAL\_CCT\_MESH  
 ANGULAR\_CCT\_MESH  
 SPATIAL\_LUM\_CIE\_MESH  
 ANGULAR\_LUM\_CIE\_MESH  
 SPATIAL\_LUM\_CCT\_MESH  
 ANGULAR\_LUM\_CCT\_MESH  
 CANDELA\_MESH  
 CANDELA\_MESH\_LINECHART  
 DATA\_MESH\_CHART  
 ILLUMINANCE\_POINTS  
 INTENSITY\_POINTS  
 INTENSITY\_SCATTER  
 ILLUMINANCE\_SCATTER  
 LUM\_METER  
 ANGULAR\_LUM\_METER  
 SPATIAL\_LUM\_METER  
 ILLUMINANCE\_OPL\_MESH  
 INTENSITY\_OPL\_MESH  
 ANGULAR\_LUMINANCE\_OPL\_MESH  
 SPATIAL\_LUMINANCE\_OPL\_MESH  
 FORWARD\_SIM\_FUNCTION  
 BACKWARD\_RAYTRACE\_FUNCTION  
 BACKWARD\_SIM\_FUNCTION  
 BACKWARD\_ILLUMINANCE\_FUNCTION  
 BACKWARD  
 BACKWARD\_POINTS  
 BACKWARD\_SIMULATION  
 BACKWARD\_ILLUMINANCE  
 BACKWARD\_INTENSITY  
 HYBRID  
 HYBRID\_SIMULATION  
 HYBRID\_ILLUMINANCE  
 HYBRID\_INTENSITY  
 BACKWARD\_ILLUMINANCE\_MESH  
 BACKWARD\_ILLUMINANCE\_POINTS  
 BACKWARD\_INTENSITY\_MESH  
 BACKWARD\_INTENSITY\_POINTS  
 HYBRID\_ILLUMINANCE\_MESH  
 HYBRID\_ILLUMINANCE\_POINTS  
 HYBRID\_INTENSITY\_MESH  
 HYBRID\_INTENSITY\_POINTS  
 ILLUMMAP\_FUNCTION  
 ILLUMMAP\_SOURCELIST  
 ILLUMMAP\_SOURCEPOINT  
 BACKWARD\_ANGULAR\_LUMINANCE

BACKWARD\_SPATIAL\_LUMINANCE  
 HYBRID\_ANGULAR\_LUMINANCE  
 HYBRID\_SPATIAL\_LUMINANCE  
 SIMULATIONS  
 RECEIVER\_DEFAULTS

#### Filters for PATH\_MANAGER Database

PATH  
     PATH\_RAY  
         PATH\_SEGMENT  
             CURVATURE\_SOLVE  
             THICKNESS\_SOLVE  
 PARAXIAL\_RAYS  
     PARAXIAL\_DATA  
     PARAXIAL\_RAY  
         PARAXIAL\_SEGMENT  
         PARAXIAL\_COORD  
     REFERENCE\_RAY  
         PATH\_RAY\_SEGMENT  
 FIELDS  
     FIELD\_POINT  
 PATH\_RAYS

#### Filters for USER\_COATINGS Database

COATING  
     COATING\_DATA\_SUBSET

#### Filters for PARAMETRIC\_CONTROLS Database

PARAMETRIC\_CONTROL\_GROUP  
     ALIAS\_GROUP  
         ALIAS  
     PARAMETRIC\_EXPRESSION\_GROUP  
         PARAMETRIC\_EXPRESSION  
         EXPRESSION  
         PARAMETER  
         NUMERIC\_PARAMETER  
         GRID\_PARAMETER  
         GRID\_EXPRESSION  
     USER\_DATA\_GROUP  
         STRING\_PARAMETER  
     PICKUP\_GROUP  
         PICKUP

#### Filters for CONFIG\_MANAGER Database

CONFIG\_ITEM

#### Filters for STUDIO\_MANAGER Database

FINISH\_LIST

FINISH

**Filters for access to MODEL data**

SELECTION-LIST  
MPI\_CLUSTER

**Filters for access to VIEW data**

ABSTRACT\_GRAPHICS\_VIEW  
NEW\_GRAPHICS\_VIEW  
NEW2D\_GRAPHICS\_VIEW  
2D\_CAD\_VIEW  
NEW3D\_GRAPHICS\_VIEW  
3D\_CAD\_VIEW  
GRAPHICS\_VIEW  
2D\_GRAPHICS\_VIEW  
LUM\_VIEW  
    PREF  
    TILER  
        MANAGER  
            AXIS  
            CHART\_SLICE  
                STYLE  
            SHADE  
            CHART\_MESH  
            HISTOGRAM  
            LEVEL  
            CONTOUR  
    DESIGN\_FEATURE\_MANAGER  
        DESIGN\_FEATURE  
        FREEFORM\_DESIGN\_FEATURE

## Appendix B General Data Access Filters

This table contains the filters that are used to access general object types in the LightTools database. The filters are used to construct object keys or in conjunction with object keys for the data access techniques described in Chapter 3, *LightTools Extensions to the BASIC Interpreter*. These general filters are not listed in the filterName column of a Table view. However, specific object types can be accessed by a general filter that corresponds to the specific filters listed in the filterName column for each object. These general filters and their associated specific filters are as follows:

### General

DATABASE

MATERIAL

MATERIAL\_ABSORPTION

MATERIAL\_INDEX

DEFAULT\_PREFERENCES

ENTITY

### Specific

LENS\_MANAGER

USER\_MATERIAL

OPTICAL\_DENSITY\_ABSORPTION  
TRANSMISSION\_ABSORPTION  
EXTINCTION\_COEFFICIENT

CAUCHY\_INDEX  
LAURENT\_INDEX  
GMLAURENT\_INDEX  
GM\_SELLMEIER  
HARTMANN\_INDEX  
STANDARD\_SELLMEIER  
INTERPOLATED\_INDEX  
CONSTANT\_INDEX  
SELFOC\_INDEX  
USER\_GRIN  
ANISOTROPIC\_INTERPOLATED\_INDEX

MATERIAL\_PREFERENCES  
REPAIR\_GEOMETRY

RAYTRACEABLE  
SOLID  
SW\_PART\_SOLID  
SURFACE\_SET  
CADFILE\_SOLID  
GENERIC\_SOLID  
SWEPT\_SOLID  
CEMENTED\_INTERFACE  
POLYLINE  
WIREBODY  
NURBS\_CURVE

## General

RAYTRACEABLE

SOLID

PRIMITIVE

## Specific

INTERPOLATED\_CURVE  
 COORDINATE\_SYSTEM  
 POINT  
 STROKE\_TEXT  
 GROUP  
 SW\_ASSEMBLY  
 DUMMY\_SURFACE  
 PLANE\_DUMMY\_SURFACE  
 SPHERE\_DUMMY\_SURFACE

SOLID  
 SW\_PART\_SOLID  
 SURFACE\_SET  
 CADFILE\_SOLID  
 GENERIC\_SOLID  
 SWEPT\_SOLID

SW\_PART\_SOLID  
 SURFACE\_SET  
 CADFILE\_SOLID  
 GENERIC\_SOLID  
 SWEPT\_SOLID

LENS\_PRIMITIVE  
 CIRC\_LENS\_PRIMITIVE  
 RECT\_LENS\_PRIMITIVE  
 CUBE\_PRIMITIVE  
 SPHERE\_PRIMITIVE  
 ELLIPSE\_PRIMITIVE  
 ELLIPTICAL\_FIBER\_PRIMITIVE  
 TOROID\_PRIMITIVE  
 REVOLUTION\_PRIMITIVE  
 EXTRUSION\_PRIMITIVE  
 REGULAR\_POLYGON\_EXTRUSION\_PRIMITIVE  
 DOVE\_PRISM\_EXTRUSION\_PRIMITIVE  
 PENTA\_PRISM\_EXTRUSION\_PRIMITIVE  
 PORRO\_PRISM\_EXTRUSION\_PRIMITIVE  
 RTANGLE\_PRISM\_EXTRUSION\_PRIMITIVE  
 HALF\_PLANE\_PRIMITIVE  
 CYLINDER\_PRIMITIVE  
 SKIN\_PRIMITIVE  
 CPC\_PRIMITIVE  
 CPCINVOLUTE\_PRIMITIVE  
 FREEFORM\_PRIMITIVE  
 SWEPT\_PRIMITIVE  
 GENERIC\_PRIMITIVE

**General**

LENS\_PRIMITIVE

EXTRUSION\_PRIMITIVE

PROFILE\_CURVE

PROFILE\_CURVE\_LIST

CURVE

SWEEP\_PATH

GENERIC\_PRIMITIVE

ANISOTROPIC\_MATERIAL\_ORIENTATION

CS\_ATTRIBUTE

**Specific**

SW\_PART\_PRIMITIVE

CADFILE\_PRIMITIVE

FUNGEO\_PRIMITIVE

CIRC\_LENS\_PRIMITIVE

RECT\_LENS\_PRIMITIVE

REGULAR\_POLYGON\_EXTRUSION\_PRIMITIVE

DOVE\_PRISM\_EXTRUSION\_PRIMITIVE

PENTA\_PRISM\_EXTRUSION\_PRIMITIVE

PORRO\_PRISM\_EXTRUSION\_PRIMITIVE

RTANGLE\_PRISM\_EXTRUSION\_PRIMITIVE

POLAR\_PROFILE\_CURVE

CIRCULAR\_PROFILE\_CURVE

ELLIPTICAL\_PROFILE\_CURVE

SQUARE\_PROFILE\_CURVE

RECT\_PROFILE\_CURVE

POLAR\_PROFILE\_CURVE\_LIST

CIRCULAR\_PROFILE\_CURVE\_LIST

ELLIPTICAL\_PROFILE\_CURVE\_LIST

SQUARE\_PROFILE\_CURVE\_LIST

RECT\_PROFILE\_CURVE\_LIST

POLYLINE\_CURVE

FITTED\_CURVE

BEZIER\_CURVE

CIRCULAR\_ARC

INTERPOLATED\_PATH

WIREFRAME\_PATH

SW\_PART\_PRIMITIVE

CADFILE\_PRIMITIVE

VOLUME\_SCATTER\_LOG

POINT\_ATTRIBUTE

POLYLINE\_ATTRIBUTE

SW\_FEATURE

SW\_COORDINATESYSTEM

SW\_POINT

SW\_REFAXIS

SW\_WIREBODY

SW\_PARAMETER

## General

SW\_FEATURE

SURFACE

LENS\_SURFACE

## Specific

SW\_COORDINATESYSTEM

SW\_POINT

SW\_REFAXIS

SW\_WIREBODY

SW\_PARAMETER

SURFACE\_EVALUATOR

PLANAR\_SURFACE

LENS\_SURFACE

SPHERICAL\_LENS\_SURFACE

CONIC\_LENS\_SURFACE

ASPHERE\_LENS\_SURFACE

POLYNOMIAL\_LENS\_SURFACE

ODDPOLYNOMIAL\_LENS\_SURFACE

ANAMORPHIC\_LENS\_SURFACE

XYPOLYNOMIAL\_LENS\_SURFACE

ZERNIKE\_LENS\_SURFACE

SUPERCONIC\_LENS\_SURFACE

XTOROID\_LENS\_SURFACE

YTOROID\_LENS\_SURFACE

XCYLINDER\_LENS\_SURFACE

YCYLINDER\_LENS\_SURFACE

SPLINESWEEP\_LENS\_SURFACE

SPLINEPATCH\_LENS\_SURFACE

CYLINDER\_SURFACE

ELLIPTICAL\_FIBER\_SURFACE

FULLSPHERE\_SURFACE

FULLELLIPSOID\_SURFACE

FULLTOROID\_SURFACE

GENERIC\_SURFACE

MESH\_SURFACE

SKIN\_SURFACE

SWEPT\_SURFACE

SWEPT\_FILLET\_SURFACE

FREEFORM\_SURFACE

SPHERICAL\_LENS\_SURFACE

CONIC\_LENS\_SURFACE

ASPHERE\_LENS\_SURFACE

POLYNOMIAL\_LENS\_SURFACE

ODDPOLYNOMIAL\_LENS\_SURFACE

ANAMORPHIC\_LENS\_SURFACE

XYPOLYNOMIAL\_LENS\_SURFACE

ZERNIKE\_LENS\_SURFACE

SUPERCONIC\_LENS\_SURFACE



**General**

SPHERICAL\_LENS\_SURFACE

CONIC\_LENS\_SURFACE

GENERIC\_SURFACE

DIRECTION\_ZONE

DOE\_DIRECTION\_ZONE

SPECULAR\_DIRECTION\_ZONE

**Specific**

XTOROID\_LENS\_SURFACE  
 YTOROID\_LENS\_SURFACE  
 XCYLINDER\_LENS\_SURFACE  
 YCYLINDER\_LENS\_SURFACE  
 SPLINESWEEP\_LENS\_SURFACE  
 SPLINEPATCH\_LENS\_SURFACE

CONIC\_LENS\_SURFACE  
 ASPHERE\_LENS\_SURFACE  
 POLYNOMIAL\_LENS\_SURFACE  
 ODDPOLYNOMIAL\_LENS\_SURFACE  
 ANAMORPHIC\_LENS\_SURFACE  
 XYPOLYNOMIAL\_LENS\_SURFACE  
 ZERNIKE\_LENS\_SURFACE  
 SUPERCONIC\_LENS\_SURFACE  
 XTOROID\_LENS\_SURFACE  
 YTOROID\_LENS\_SURFACE

ASPHERE\_LENS\_SURFACE  
 POLYNOMIAL\_LENS\_SURFACE  
 ODDPOLYNOMIAL\_LENS\_SURFACE  
 ANAMORPHIC\_LENS\_SURFACE  
 XYPOLYNOMIAL\_LENS\_SURFACE  
 ZERNIKE\_LENS\_SURFACE  
 SUPERCONIC\_LENS\_SURFACE  
 XTOROID\_LENS\_SURFACE  
 YTOROID\_LENS\_SURFACE

MESH\_SURFACE

DOE\_DIRECTION\_ZONE  
 RADPOLY\_DOE\_DIRECTION\_ZONE  
 RULED\_DOE\_DIRECTION\_ZONE  
 XYPOLY\_DOE\_DIRECTION\_ZONE  
 SPECULAR\_DIRECTION\_ZONE  
 DOMINANT\_DIRECTION\_ZONE  
 FRESNEL\_DIRECTION\_ZONE  
 RADIAL\_FRESNEL\_DIRECTION\_ZONE  
 CYLINDRICAL\_FRESNEL\_DIRECTION\_ZONE

RADPOLY\_DOE\_DIRECTION\_ZONE  
 RULED\_DOE\_DIRECTION\_ZONE  
 XYPOLY\_DOE\_DIRECTION\_ZONE

DOMINANT\_DIRECTION\_ZONE  
 FRESNEL\_DIRECTION\_ZONE

## General

FRESNEL\_DIRECTION\_ZONE

AMPLITUDE\_ZONE

SCATTER\_AMPLITUDE\_ZONE

## Specific

RADIAL\_FRESNEL\_DIRECTION\_ZONE  
CYLINDRICAL\_FRESNEL\_DIRECTION\_ZONE

RADIAL\_FRESNEL\_DIRECTION\_ZONE  
CYLINDRICAL\_FRESNEL\_DIRECTION\_ZONE

FRESNEL\_AMPLITUDE\_ZONE  
RT\_AMPLITUDE\_ZONE  
QWAR\_COATING\_AMPLITUDE\_ZONE  
THIN\_FILM\_STACK\_COATING\_AMPLITUDE\_ZONE  
USER\_COATING\_AMPLITUDE\_ZONE  
IDEAL\_POLARIZER\_AMPLITUDE\_ZONE  
IDEAL\_RETARDER\_AMPLITUDE\_ZONE  
JONES\_MATRIX\_AMPLITUDE\_ZONE  
MUELLER\_MATRIX\_AMPLITUDE\_ZONE  
SCATTER\_AMPLITUDE\_ZONE  
COSN\_SCATTER\_AMPLITUDE\_ZONE  
GAUSSIAN\_SCATTER\_AMPLITUDE\_ZONE  
LAMBERTIAN\_SCATTER\_AMPLITUDE\_ZONE  
AOI\_SCATTER\_AMPLITUDE\_ZONE  
USER\_SCATTER\_AMPLITUDE\_ZONE  
ELLIPTICAL\_GAUSSIAN\_SCATTER\_AMPLITUDE\_ZONE  
MIXED\_SCATTER\_AMPLITUDE\_ZONE  
USER\_DEFINED\_AMPLITUDE\_ZONE  
DIFFRACTIVE\_OPTICAL\_PROPERTIES  
DIFFRACTION\_EFFICIENCY\_ZONE  
PARTICLE  
LUMINESCENCE  
MIE\_SCATTERER  
HG\_SCATTERER  
GEGENBAUER\_SCATTERER  
USER\_DEFINED\_VOLUME\_SCATTERER  
USER\_DEFINED\_VOLUME\_SCATTERER\_SERVER  
INERT\_PARTICLE

COSN\_SCATTER\_AMPLITUDE\_ZONE  
GAUSSIAN\_SCATTER\_AMPLITUDE\_ZONE  
LAMBERTIAN\_SCATTER\_AMPLITUDE\_ZONE  
AOI\_SCATTER\_AMPLITUDE\_ZONE  
USER\_SCATTER\_AMPLITUDE\_ZONE  
ELLIPTICAL\_GAUSSIAN\_SCATTER\_AMPLITUDE\_ZONE  
MIXED\_SCATTER\_AMPLITUDE\_ZONE  
USER\_DEFINED\_AMPLITUDE\_ZONE

**General**

DIFFRACTIVE\_OPTICAL\_PROPERTIES

PARTICLE

ZONE\_EXTENT

2D\_RECT\_ARRAY\_ZONE\_EXTENT

3D\_ARRAY\_ZONE\_EXTENT

COORDINATE\_SYSTEM

GROUP

DUMMY\_SURFACE

OPT\_VARIABLE

OPT\_CONSTRAINT

**Specific**

DIFFRACTIVE\_OPTICAL\_PROPERTIES

DIFFRACTION\_EFFICIENCY\_ZONE

DIFFRACTION\_EFFICIENCY\_ZONE

LUMINESCENCE

MIE\_SCATTERER

HG\_SCATTERER

GEGENBAUER\_SCATTERER

USER\_DEFINED\_VOLUME\_SCATTERER

USER\_DEFINED\_VOLUME\_SCATTERER\_SERVER

INERT\_PARTICLE

CIRC\_ZONE\_EXTENT

RECT\_ZONE\_EXTENT

2D\_RECT\_ARRAY\_ZONE\_EXTENT

2D\_RECT\_ARRAY\_ARCS\_ZONE\_EXTENT

2D\_RECT\_ARRAY\_CIRCLES\_ZONE\_EXTENT

2D\_RECT\_ARRAY\_ELLIPSES\_ZONE\_EXTENT

2D\_RECT\_ARRAY\_RECTS\_ZONE\_EXTENT

3D\_ARRAY\_ZONE\_EXTENT

3D\_HEX\_ARRAY\_ZONE\_EXTENT

3D\_RECT\_ARRAY\_ZONE\_EXTENT

3D\_SHIFT\_RECT\_ARRAY\_ZONE\_EXTENT

TEXTURE\_ZONE\_EXTENT

2D\_RECT\_ARRAY\_ARCS\_ZONE\_EXTENT

2D\_RECT\_ARRAY\_CIRCLES\_ZONE\_EXTENT

2D\_RECT\_ARRAY\_ELLIPSES\_ZONE\_EXTENT

2D\_RECT\_ARRAY\_RECTS\_ZONE\_EXTENT

3D\_HEX\_ARRAY\_ZONE\_EXTENT

3D\_RECT\_ARRAY\_ZONE\_EXTENT

3D\_SHIFT\_RECT\_ARRAY\_ZONE\_EXTENT

POINT

SW\_ASSEMBLY

PLANE\_DUMMY\_SURFACE

SPHERE\_DUMMY\_SURFACE

OPT\_DBVARIABLE

OPT\_DBCONSTRAINT

**General**

OPT\_MERITFUNCTION

OPT\_MESHMERITFUNCTION

OPT\_CIEMESHMERITFUNCTION

OPT\_FOCUSCOLLIMATEMERITFUNCTION

OPT\_MESHMTARGETFUNCTION

TOL\_PERFORMANCEGROUPITEM

TOL\_MESHPERFORMANCEMEASURE

SOURCE

**Specific**

OPT\_MESHMERITFUNCTION

OPT\_CIEMESHMERITFUNCTION

OPT\_COLORDIFFMESHMERITFUNCTION

OPT\_CCTMESHMERITFUNCTION

OPT\_FOCUSCOLLIMATEMERITFUNCTION

OPT\_FOCUSMERITFUNCTION

OPT\_COLLIMATEMERITFUNCTION

OPT\_SLICEMERITFUNCTION

OPT\_POINTMERITFUNCTION

OPT\_USERMERITFUNCTION

OPT\_DBMERITFUNCTION

OPT\_NSRAYMERITFUNCTION

OPT\_PENALTYCONSTRAINTMERITFUNCTION

OPT\_CIEMESHMERITFUNCTION

OPT\_COLORDIFFMESHMERITFUNCTION

OPT\_CCTMESHMERITFUNCTION

OPT\_COLORDIFFMESHMERITFUNCTION

OPT\_CCTMESHMERITFUNCTION

OPT\_FOCUSMERITFUNCTION

OPT\_COLLIMATEMERITFUNCTION

OPT\_HIRESMESHTARGETFUNCTION

TOL\_DBPERFORMANCEMeasure

TOL\_DBPERFORMANCEGROUPITEM

TOL\_CIEMESHPERFORMANCEMEASURE

POINT\_SOURCE

SURFACE\_SOURCE

OBJECT\_SOURCE

CUBE\_SURFACE\_SOURCE

CYLINDER\_SURFACE\_SOURCE

SPHERE\_SURFACE\_SOURCE

TOROID\_SURFACE\_SOURCE

EMBEDDED\_GEOM\_SOURCE

DISK\_SOURCE

RECT\_SOURCE

LUCIDSHAPE\_SOURCE

LUCIDSHAPE\_POINT\_SOURCE

SURFACE\_EMITTER

EMITTER

VOLUME\_SOURCE

**General****Specific**

|                                         |                                           |
|-----------------------------------------|-------------------------------------------|
|                                         | CUBE_VOLUME_SOURCE                        |
|                                         | CYLINDER_VOLUME_SOURCE                    |
|                                         | SPHERE_VOLUME_SOURCE                      |
|                                         | TOROID_VOLUME_SOURCE                      |
|                                         | RAY_DATA_SOURCE                           |
|                                         | RAY_DATA_SOURCE_COLLECTION_SURFACE        |
|                                         | LS_RAY_DATA_SOURCE                        |
| OBJECT_SOURCE                           | CUBE_SURFACE_SOURCE                       |
|                                         | CYLINDER_SURFACE_SOURCE                   |
|                                         | SPHERE_SURFACE_SOURCE                     |
|                                         | TOROID_SURFACE_SOURCE                     |
| EMBEDDED_GEOM_SOURCE                    | DISK_SOURCE                               |
|                                         | RECT_SOURCE                               |
|                                         | LUCIDSHAPE_SOURCE                         |
|                                         | LUCIDSHAPE_POINT_SOURCE                   |
|                                         | SURFACE_EMITTER                           |
|                                         | EMITTER                                   |
| VOLUME_SOURCE                           | CUBE_VOLUME_SOURCE                        |
|                                         | CYLINDER_VOLUME_SOURCE                    |
|                                         | SPHERE_VOLUME_SOURCE                      |
|                                         | TOROID_VOLUME_SOURCE                      |
| RAY_DATA_SOURCE                         | RAY_DATA_SOURCE_COLLECTION_SURFACE        |
|                                         | LS_RAY_DATA_SOURCE                        |
| RAY_DATA_SOURCE_NULL_COLLECTION_SURFACE | RAY_DATA_SOURCE_PLANAR_COLLECTION_SURFACE |
|                                         | RAY_DATA_SOURCE_SPHERE_COLLECTION_SURFACE |
| AIM                                     | AIM_SPHERE                                |
|                                         | AIM_AREA                                  |
|                                         | AIM_PATCH                                 |
|                                         | AIM_CONE                                  |
|                                         | AIM_TARGET_SPHERE                         |
| RECEIVER                                | SURFACE_RECEIVER                          |
|                                         | SOLID_RECEIVER                            |
|                                         | PRIMITIVE_RECEIVER                        |
|                                         | FARFIELD_RECEIVER                         |
| RECEIVER_FILTER                         | RAY_MAGNITUDE_FILTER                      |
|                                         | ELEMENT_FILTER                            |
|                                         | EXIT_ANGLE_FILTER                         |

## General

## Specific

HIT\_NUMBER\_FILTER  
 INCIDENT\_ANGLE\_FILTER  
 SOURCE\_FILTER  
 SURFACE\_FILTER  
 WAVELENGTH\_FILTER  
 ORDINAL\_NUMBER\_FILTER  
 PROPERTY\_ZONE\_FILTER  
 OPTICAL\_PROPERTY\_FILTER  
 OPTICAL\_PROPERTY\_AFTER\_FILTER  
 PATH\_TRANSMITTANCE\_FILTER  
 OPTICAL\_PATH\_LENGTH\_FILTER  
 POLARIZATION\_FILTER  
 LINEAR\_POLARIZER\_FILTER  
 AFTER\_SURFACE\_FILTER  
 AFTER\_ELEMENT\_FILTER  
 AFTER\_PROPERTY\_ZONE\_FILTER  
 VOLUME\_INTERFACE\_FILTER  
 AFTER\_VOLUME\_INTERFACE\_FILTER  
 RAYPATH\_FILTER  
 LIGHT\_GUIDE\_FILTER  
 ANGULAR\_CONE\_FILTER  
 NEAR\_SPECULAR\_COUNT\_FILTER  
 DIFFUSE\_COUNT\_FILTER  
 SCATTER\_COUNT\_FILTER  
 SPECULAR\_COUNT\_FILTER  
 REFLECTANY\_COUNT\_FILTER  
 REFLECTONLY\_COUNT\_FILTER  
 REFRACT\_COUNT\_FILTER  
 TIR\_COUNT\_FILTER  
 FILTER\_GROUP

## MESH

INTENSITY\_MESH  
 ILLUMINANCE\_MESH  
 SPATIAL\_LUMINANCE\_MESH  
 ANGULAR\_LUMINANCE\_MESH  
 SPATIAL\_CIE\_MESH  
 ANGULAR\_CIE\_MESH  
 SPATIAL\_CCT\_MESH  
 ANGULAR\_CCT\_MESH  
 SPATIAL\_LUM\_CIE\_MESH  
 ANGULAR\_LUM\_CIE\_MESH  
 SPATIAL\_LUM\_CCT\_MESH  
 ANGULAR\_LUM\_CCT\_MESH  
 CANDELA\_MESH  
 ILLUMINANCE\_POINTS  
 INTENSITY\_POINTS

**General**

ENCIRCLED\_ENERGY\_APE

LUM\_METER

BACKWARD\_RAYTRACE\_FUNCTION

VIEW

ABSTRACT\_GRAPHICS\_VIEW

**Specific**

CIRC\_ENCIRCLED\_ENERGY\_APE  
 SLIT\_ENCIRCLED\_ENERGY\_APE  
 OVAL\_ENCIRCLED\_ENERGY\_APE  
 RECT\_ENCIRCLED\_ENERGY\_APE  
 SQUARE\_ENCIRCLED\_ENERGY\_APE

ANGULAR\_LUM\_METER  
 SPATIAL\_LUM\_METER  
 ILLUMINANCE\_OPL\_MESH  
 INTENSITY\_OPL\_MESH  
 ANGULAR\_LUMINANCE\_OPL\_MESH  
 SPATIAL\_LUMINANCE\_OPL\_MESH

BACKWARD\_SIM\_FUNCTION  
 BACKWARD\_ILLUMINANCE\_FUNCTION  
 BACKWARD\_ILLUMINANCE\_MESH  
 BACKWARD\_ILLUMINANCE\_POINTS  
 BACKWARD\_INTENSITY\_MESH  
 BACKWARD\_INTENSITY\_POINTS  
 HYBRID\_ILLUMINANCE\_MESH  
 HYBRID\_ILLUMINANCE\_POINTS  
 HYBRID\_INTENSITY\_MESH  
 HYBRID\_INTENSITY\_POINTS  
 ILLUMMAP\_FUNCTION  
 BACKWARD\_ANGULAR\_LUMINANCE  
 BACKWARD\_SPATIAL\_LUMINANCE  
 HYBRID\_ANGULAR\_LUMINANCE  
 HYBRID\_SPATIAL\_LUMINANCE

ABSTRACT\_GRAPHICS\_VIEW  
 NEW\_GRAPHICS\_VIEW  
 NEW2D\_GRAPHICS\_VIEW  
 2D\_CAD\_VIEW  
 NEW3D\_GRAPHICS\_VIEW  
 3D\_CAD\_VIEW  
 GRAPHICS\_VIEW  
 2D\_GRAPHICS\_VIEW  
 LUM\_VIEW

NEW\_GRAPHICS\_VIEW  
 NEW2D\_GRAPHICS\_VIEW  
 2D\_CAD\_VIEW  
 NEW3D\_GRAPHICS\_VIEW  
 3D\_CAD\_VIEW  
 GRAPHICS\_VIEW

**General****Specific**

NEW\_GRAPHICS\_VIEW

2D\_GRAPHICS\_VIEW

NEW2D\_GRAPHICS\_VIEW

2D\_CAD\_VIEW

NEW3D\_GRAPHICS\_VIEW

3D\_CAD\_VIEW

NEW2D\_GRAPHICS\_VIEW

2D\_CAD\_VIEW

NEW3D\_GRAPHICS\_VIEW

3D\_CAD\_VIEW

NEW3D\_GRAPHICS\_VIEW

3D\_CAD\_VIEW

GRAPHICS\_VIEW

2D\_GRAPHICS\_VIEW

DESIGN\_FEATURE

FREEFORM\_DESIGN\_FEATURE

There are exceptions to the rules for specific and general filters. The MATERIAL, VOLUME\_INTERFACE, and CONIC\_LENS\_SURFACE filters can be used as both specific and general filters. As specific filters, the MATERIAL filter is used to access user materials without refractive index and optical density specifications, and the CONIC\_LENS\_SURFACE filter is used to access surfaces that are specifically the conic type. In these instances, they allow access to the data elements associated with the MATERIAL and CONIC\_LENS\_SURFACE filter. As general filters, the MATERIAL filter is used to access any user material, and the CONIC\_LENS\_SURFACE filter is used to access any surface that has a conic constant. In these instances, they allow access to data elements associated with additional user material and surface filters. These additional user material and surface filters are classified as specific filters and are listed below.

|                    |                            |
|--------------------|----------------------------|
| MATERIAL           | USER_MATERIAL              |
|                    | GLASS_MATERIAL             |
|                    |                            |
| VOLUME_INTERFACE   |                            |
|                    |                            |
| CONIC_LENS_SURFACE | ASPHERE_SURFACE            |
|                    | POLYNOMIAL_LENS_SURFACE    |
|                    | ODDPOLYNOMIAL_LENS_SURFACE |
|                    | ANAMORPHIC_LENS_SURFACE    |
|                    | XYPOLYNOMIAL_LENS_SURFACE  |
|                    | ZERNIKE_LENS_SURFACE       |
|                    | SUPERCONIC_LENS_SURFACE    |
|                    | XTOROID_LENS_SURFACE       |
|                    | YTOROID_LENS_SURFACE       |