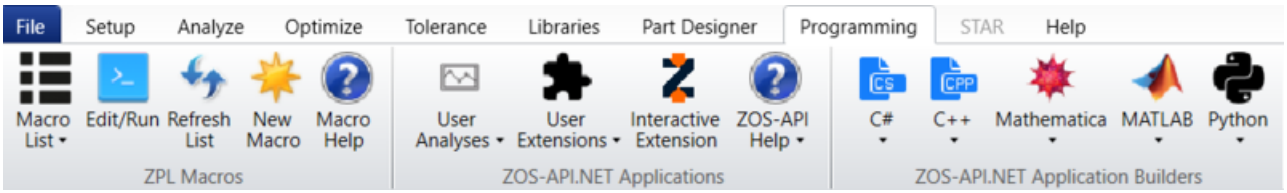
10: The Programming Tab

This tab provides access to OpticStudio's Programming features.



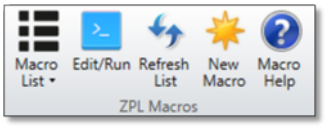
Although OpticStudio has a huge range of features and analysis options, specific applications may need some special feature or requirement. That's why OpticStudio has programming interfaces built right in. The first is the **Zemax Programming Language (ZPL)**, which is a very easy to learn scripting language similar to Basic. With ZPL, it is easy to perform special calculations, display data in different ways and automate repetitive keyboard tasks, among many others. Tools for creating, editing, and running ZPL Macros are located in the **ZPL Macros** group.

Ansys Zemax OpticStudio also has a **ZOS-API** programming interface which can be used in a .NET environment, using C#, or any other .NET-capable language. In addition, ZOS-API can also be used in a .COM environment, using C++, or any other .COM-capable language. Tools for using the ZOS-API are in the **ZOS-API.NET Applications** and **ZOS-API.NET Application Builders** groups.

- About the ZOS-API

10.1. ZPL Macros Group

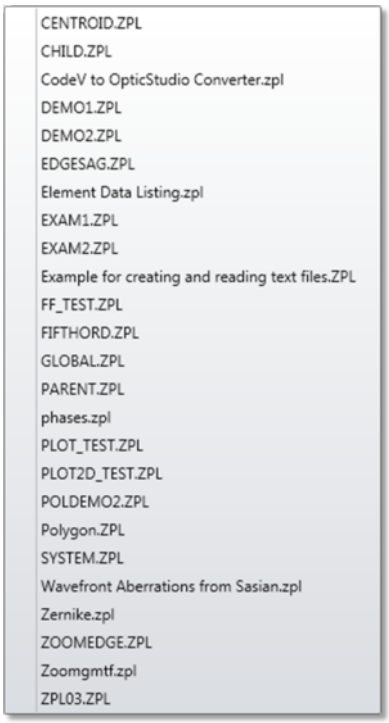
ZPL Macros is a section of the Programming Tab.

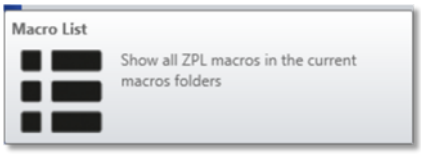


For more information, please see the section About the ZPL.

10.1.1. Macro List

The Macro List button is found in the ZPL Macros section of the Programming Tab.

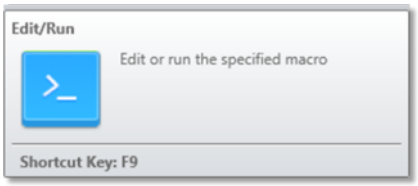




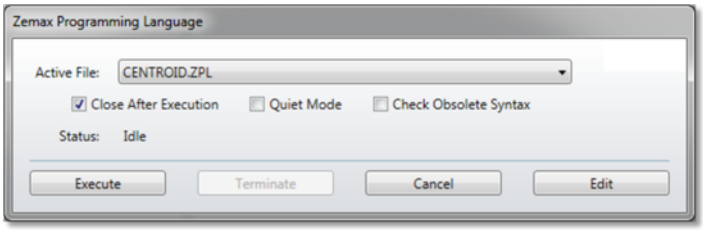
This feature shows a drop-down list of all ZPL Macros in the current macros folder (see " Folders ").

By clicking on the macro name, it immediately executes.

10.1.2. Edit/Run



The Edit/Run macro tool is found in the ZPL Macros section of the Programming Tab.



This tool edits or runs the specified macro.

To run a ZPL macro, select the Edit/Run button in the ZPL Macros section of the Programming Tab. Select the macro to run from the "Active File" list, and then click on Execute.

OpticStudio will begin running the macro. The entire macro will run, and any text output from PRINT commands or error messages will be displayed in a window. The display of this output window can be suppressed using the CLOSEWINDOW keyword.

This ZPL control dialog has the following options:

Active File A drop-down list of macros available. All the listed macros are text files ending in the extension .ZPL. The files must be in the folder for ZPL Macros; see " Folders ". Macros may also be placed in any subfolder within the macros folder.

Close After Execution If checked, the ZPL control dialog box will automatically close after the macro execution.

Quiet Mode If checked, the default output text window will not be shown. This is useful for graphics macros that do not generate useful text.

Check Obsolete Syntax If checked, OpticStudio will test the macro for use of obsolete syntax.

Status During execution of the macro, OpticStudio will use this area to print a status message stating the line number of the macro being executed. The status message is updated every quarter second.

Execute Click to begin execution of the selected macro.

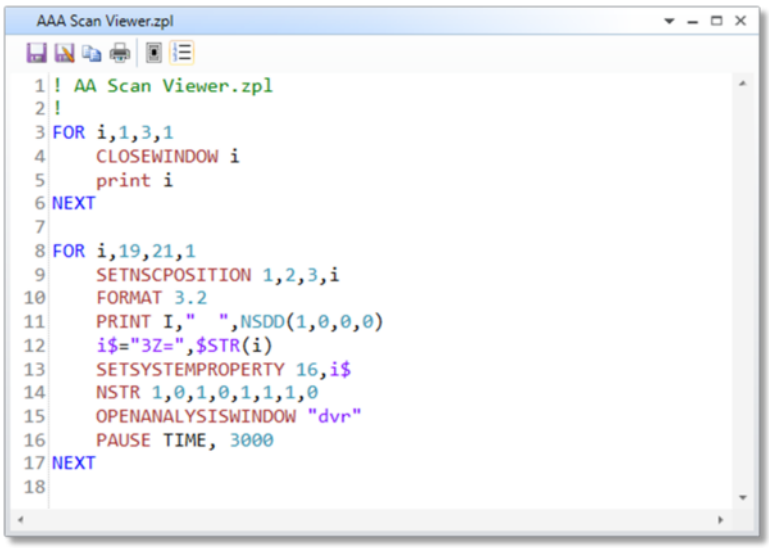
Terminate The terminate button will stop execution of the macro currently running.

Cancel The cancel button terminates the current macro if one is running. If no macro is running, cancel closes the ZPL control dialog box.

Edit The edit button invokes OpticStudio's text editor which can be used to modify or rename the macro. This editor colors the macro text using the following ZPL syntax:

Syntax Category	Color
Comment	Green
String	Violet
Number	Light Blue
ControlKeyword	Blue
FunctionKeyword	Brown
Identifier	Black

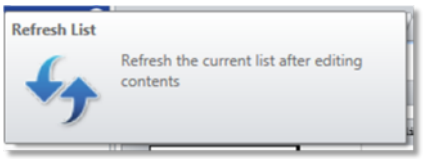
For example, see the following screenshot of a sample macro:



The text editor also contains a search bar that can be opened by clicking the "Find text" button in the tool bar.

10.1.3. Refresh List (zpl macros group)

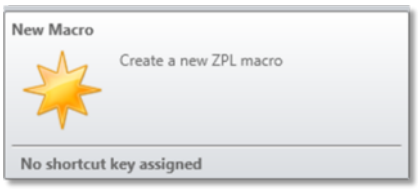
The Refresh List button is found in the ZPL Macros section of the Programming Tab.



This tool updates the current macro list; which may be required if any macros have been added or deleted since OpticStudio was started or the last time the list was refreshed.

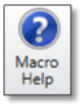
10.1.4. New Macro

The New Macro button is found in the ZPL Macros section of the Programming Tab.

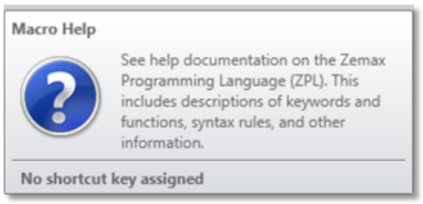


This button opens an OpticStudio text editor to create a new ZPL macro. Macros created with this text editor are automatically saved with the *.ZPL extension and in the Macros user data folder.

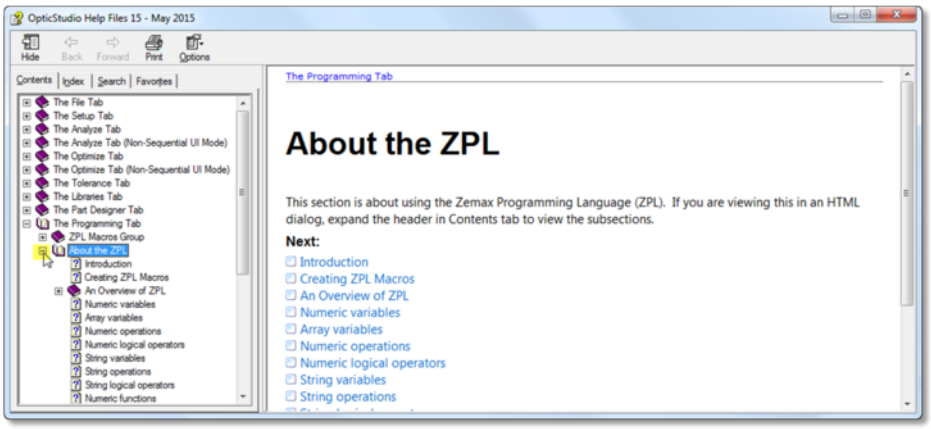
10.1.5. Macro Help



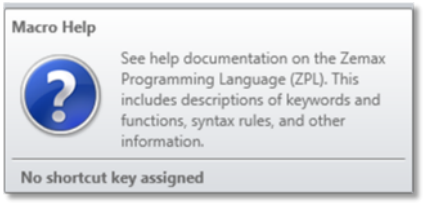
The Macro Help button is found in the ZPL Macros section of the Programming Tab.



This opens the HTML OpticStudio Help Files to the section About the ZPL. If you are viewing this as an HTML window, please click the "+" next to the section header in the Contents pane to expand the subtopics.



10.2. About the ZPL



This section is about using the Zemax Programming Language (ZPL). If you are viewing this in an HTML dialog, expand the header in Contents tab to view the subsections.

10.2.1. Introduction (about the zpl)

The Zemax Programming Language (ZPL) is a macro language specifically designed for use with OpticStudio. ZPL offers the power of user-extensibility. This means that if you need a particular calculation or graphical display which is not "built in", you can write your own ZPL macro to do the job.

ZPL is similar to the BASIC programming language, except not all BASIC constructs and keywords are supported, and capabilities and functions unique to ray tracing have been added. ZPL is easy to use, and this section will give you instructions and examples to get you started.

ZPL is a powerful programming language. While it is easy to use, ZPL has no built-in debugging or variable shadowing capabilities, so the user is responsible for error-checking, debugging code logic and for good programming practice. For this reason, technical support on macro writing is restricted to ensuring that all ZPL functions and keywords work as documented: we cannot advise on how to perform detailed calculations as part of technical support. If you need a OpticStudio macro, and do not possess the desire or ability to write it yourself, please feel free to contact OpticStudio Technical Support for a quote on developing a custom program to meet your requirements. We have considerable experience in developing these types of programs, and can generally write macros at very competitive rates on short notice.

10.2.2. Creating ZPL Macros

A ZPL macro consists of a series of commands which are stored in a text file. The commands are either assignments, keywords, or comments. See "OPTICSTUDIO PROGRAMMING LANGUAGE" for more information.

To create a ZPL macro, it is probably easiest to start with an existing macro that performs a task similar to the one you want to achieve. If you are attempting to write your first ZPL macro, you may want to read the example sections at the end of this help file. Some example ZPL macros can also be found in the <data>\Macros folder installed with the OpticStudio program.

Use any text editor to create the ZPL file (such as the NOTEPAD editor). The file may have any name but must end in the .ZPL extension. The file must be placed in the ZPL Folder, which by default is <data>\Macros. To change this folder see " Folders ".

There is a limit to the allowed complexity of any single line in a ZPL macro. If the "line too long" error occurs, try breaking the line into several smaller lines.

10.2.3. An Overview of ZPL

A ZPL macro consists of a series of commands which are stored in a text file. The commands are either assignments, keywords, or comments. Assignments may be for either numeric or string (textual) data. Both assignments and keywords may accept expressions as arguments, although the syntax is slightly different as described below.

10.2.3.1. Assignments

The general syntax for an assignment is

```
variable = (expression)
```

The (expression) may consist of an explicit value, such as 5, or a variable name containing some preassigned value, or a complex mathematical expression involving functions, constants, and variables. In all cases, the expression on the right side of the equal sign is evaluated, and the result is assigned to the variable designated on the left.

The simplest form of an assignment is where the expression is a fixed value, such as:

```
x = 5
```

There are several important things to note in this command. First, no declaration of variables is required. This means "x", which is called a variable, did not need to exist before the value of 5 was assigned to it. If "x" had already been assigned a value, it would now be reassigned. Second, no special symbol is required to terminate a command, such as ";" in C. Because of this, each ZPL command must be on a line by itself.

Here are some examples of assignments with expressions:

```
x = SQRT(5)
y = SINE(x)
z = SQRT(x+5*(7-x))
```

The functions SQRT (square root) and SINE (sine) are built in to ZPL. There are many of these functions, all of which are defined in "Numeric functions". Note that ZPL is not case-sensitive; SQRT() and sqrt() are the same function. This documentation will use the convention of capital letters for functions and keywords, and lower case for everything else.

There are also string assignments as described in "String variables".

10.2.3.2. Keywords (an overview of zpl)

The general syntax for a keyword is

```
KEYWORD argument1, argument2, argument3...
```

Some keywords have no arguments, others have many. Arguments may be either numeric expressions or string constants or string variables. Some keywords accept a mixture of numeric and string arguments.

One example of a keyword is PRINT. The PRINT keyword is followed by a list of items, separated by commas, to be printed. For example, the ZPL commands

```
x = 3 y = 4
z = SQRT( x * x + y * y ) PRINT "The hypotenuse is ",z
```

will print the following:

```
The hypotenuse is 5.0000
```

Note that ZPL enforces operator precedence. ZPL uses the following precedence from highest to lowest: Parentheses, functions (such as SQRT), logical operators (such as ==), multiplication and division, and then addition and subtraction.

There are many keywords, all are described in "Keywords".

10.2.3.3. Comments

There are 3 ways to add comments to a ZPL macro: by starting a line with the REM keyword, by starting a line with the "!" symbol, or by placing the "#" symbol anywhere on the line as long as the # symbol is not inside a string. Blank lines are also allowed anywhere in the macro. Here are 3 examples of comments:

```
REM this is a remark
! This is also a remark
x = 5 # The # symbol allows comments on the same line as a valid command
```

Comments make macros easier to understand and modify, and have no effect on performance.

10.2.3.4. Creating Graphics

There are a number of low-level functions for generating graphics, including GRAPHICS, GTEXT, GLEN- SNAME, and others. For details see "GRAPHICS". An easier method of generating typical OpticStudio style line graphs and 2D graphs graphics is available using the PLOT and PLOT2D keywords; see "PLOT" and "PLOT2D".

10.2.4. Numeric Variables

Variables provide temporary storage for numerical quantities whose exact value may not be known when the macro is written, but is defined when the macro is run. OpticStudio performs most of the work for you when you need a new variable. For example, the command

```
x = 5
```

will cause OpticStudio to allocate memory for the new variable "x" and keep track of the value associated with it. Once the variable is defined, it may be used in any subsequent expression. There are a few rules regarding ZPL variables:

Variable names must not contain any "special" characters that ZPL uses for logical operations or delimiting such as (,), =, +, -, *, /, !, >, <, ^, &, |, #, ", and the space character.

A variable cannot take on the same name as a keyword or function, such as THIC or RAYX. Since ZPL is not case-sensitive, you cannot use rayX or Thic to avoid this rule.

Each variable name is limited to 28 characters.

All ZPL numeric variables are stored internally as 64-bit double precision numbers.

10.2.5. Array Variables

Array variables are single- or multi-dimensioned arrays of double precision or integer values. Unlike (scalar) numeric variables, array variables must be declared prior to their use. The declaration syntax is

```
DECLARE name, type, num_dimensions, dimension1 [, dimension 2 [, dimension 3 [,
 dimension 4] etc...]]
```

The name may be any legal variable name as described in the previous section. The type must be either DOUBLE or INTEGER; this value indicates the type of array variable. The integer value num_dimensions defines the number of dimensions of the array (not the size), and must be between 1 and 4, inclusive. The integers dimension1, dimension2, etc., define the size of the array in that dimension. Note that array variables start at index 1, and thus an array of size 10 has valid indices from 1 to 10.

Array variables may be defined anywhere inside the macro, they need not be declared at the beginning of the macro. To release the memory associated with an array variable, use the RELEASE keyword. The syntax is

```
RELEASE name
```

The RELEASE keyword is optional, as the memory associated with the declared variable is automatically released when the macro terminates. However the RELEASE keyword is useful for conserving memory if large arrays are only needed during a portion of the macro execution.

Array variables are assigned values using the following syntax:

```
name ( index1, index2, ... ) = value
```

The values stored in the array may be retrieved with the same basic syntax:

```
value = name ( index1, index2, ...)
```

The following sample code declares a two-dimensional array variable, assigns a value to each element, prints the values, and then releases the memory for the array:

```
DECLARE Z, DOUBLE, 2, 5, 5
FOR i, 1, 5, 1
FOR j, 1, 5, 1
Z(i, j) = i + j
NEXT j
NEXT i

k = 0
FORMAT 8.0
FOR i, 1, 5, 1
FOR j, 1, 5, 1
PRINT k, i, j, Z(i,j)
k = k + 1
NEXT j
NEXT i

RELEASE Z
```

10.2.6. Numeric Operations

ZPL macros support basic numeric operations such as add, subtract, multiply, and divide. The syntax for each is shown below.

```
x = y + z
x = y - z
x = y * z
x = y / z
```

All other operations are supported only through the use of numeric functions or numeric logical operators, both described in subsequent sections.

10.2.7. Numeric Logical Operators

Logical operators are used to construct complex commands which ultimately evaluate to one or zero. Most logical operations take the form (left_expression) (operator) (right_expression), similar to mathematical expres- sions such as 1 + 2. The exception is the not operator "!" which takes only a single argument, of the form ! (right_expression). The logical operators use the convention that zero is "false" and any non-zero value is "true". The not operator returns 1 (true) if the (right_expression) is 0 (false) and returns 0 (false) if (right_expression) is non-zero (true). One common use of the not operator is in IF commands such as:

```
IF !x THEN PRINT "x is zero."
```

The other logical operators can be also be used as part of the argument in IF commands. For example, an IF command may contain two conditions which must both be true for the THEN command to be executed:

```
IF ( x > 1 ) & ( y < 2 ) THEN PRINT "Both conditions are true."
```

These two conditions are related by an "and" expression denoted by &. Note the parentheses are used to force precedence. ZPL supports the logical operators described in the following table.

NUMERIC LOGICAL OPERATORS

Logical	Description

&	And, returns 1 only if both expressions are non-zero.
|	Or, returns 1 if at least one expression is non-zero.
^	Xor, returns 1 if only one expression is non-zero.
!	Not, returns 0 if (right_expression) is non-zero, else returns 1.
==	Equality, returns 1 if expressions are equal.
>	Greater than, returns 1 if left_expression is greater than right_expression.
<	Less than, returns 1 if left_expression is less than right_expression.
>=	Greater than or equal to, returns 1 if left_expression is greater than or equal to right_expression.
<=	Less than or equal to, returns 1 if left_expression is less than or equal to right_expression.
!=	Inequality, returns 1 if expressions are unequal.

10.2.8. String Variables

ZPL supports string variables and basic string operations. String variables can hold a maximum of 360 characters. String variables do not need to be declared, but can be created at any time using a defining assignment command such as:

```
newstring$ = "Here is the new string"
```

Note that string variables are distinguished from numeric variables by the presence of the $ character at the end of the string.

There are also string functions which can be used to extract text data, such as

```
title$ = $LENSNAME()
```

Note the function $LENSTITLE() starts with the $ character. This identifies the function as returning a string result.

10.2.9. String Operations

String variables can be concatenated using the + operator. The syntax is:

```
C$ = A$ + B$
```

Concatenation can also include constant strings:

```
total$ = "A$ is " + A$ + " and B$ is " + B$
```

The string functions can be used in a defining command such as

```
this$ = "Here is the lens title: " + $LENSNAME()
```

String variables are printed just like other strings:

```
PRINT "Here is A$: ", A$
```

Note that the PRINT function can only print single string variables; there is no support for the concatenation operand or string functions inside print commands. The correct procedure is to concatenate the strings into a new string and then print the new string:

```
A$ = B$ + C$
PRINT A$
```

Alternatively, the comma acts as a concatenation operand:

```
PRINT A$, B$, C$
```

String functions cannot be printed directly like this:

```
PRINT $LENSNAME() ! NOT CORRECT !!!
```

Instead, the correct procedure is to assign the function result to a variable and then print the variable:

```
Z$ = $LENSNAME()
PRINT Z$
```

10.2.10. String Logical Operators

String logical operators are very similar to the numeric logical operators defined in a previous section. The key difference is that the expressions being compared are strings rather than numbers. The supported string logical operators are defined in the following table.

STRING LOGICAL OPERATORS

Logical	Description
$==	Equality, returns 1 if left_string and right_string are identical.
$>	Greater than, returns 1 if left_string is greater than right_string.
$<	Less than, returns 1 if left_string is less than right_string.
$>=	Greater than or equal to, returns 1 if left_string is greater than or identical to right_string.
$<=	Less than or equal to, returns 1 if left_string is less than or identical to right_string.
$!=	Inequality, returns 1 if left_string and right_string are not identical.

For example, an IF command may compare strings as follows:

```
A$ = "TEST"
BS = "TEST"
IF (A$ $== B$) THEN PRINT "Strings are identical."
```

10.2.11. Numeric Functions

Numeric functions can be used on the right hand side of a numeric variable assignment, and in expressions which are arguments to keywords. These functions may require no arguments, one argument, or multiple arguments. All functions return a single value. Certain functions, such as PWAV() (primary wavelength) , return a value which does not depend upon the argument, and therefore it is not required to provide one. The parentheses however, are still required.

In the following table, all ZPL functions are listed. If the syntax is given as FUNC(), then no arguments are required. FUNC(x) indicates one argument is required, FUNC(x,y) indicates two arguments, etc.

ZPL FUNCTION LISTING

Function	Argument	Return value
ABSO(x)	Numeric expression	The absolute value of the expression.
ACOS(x)	Numeric expression	Arc cosine in radians. This function can be used to define the constant pi: π = 2*ACOS(0).
APMN(x)	Valid surface number	The minimum radius value. For spider apertures, this is the width of the arms. For rectangular and elliptical apertures, it is the x-half width of the aperture.
APMX(x)	Valid surface number	The maximum radius value. For spider apertures, this is the number of arms. For rectangular and elliptical apertures, it is the y-half width of the aperture.
APOI(px, py)	Normalized pupil coordinates	Ray intensity apodization factor.
APXD(x)	Valid surface number	The aperture x-decenter value.
APYD(x)	Valid surface number	The aperture y-decenter value.
APTP(x)	Valid surface number	An integer code describing the aperture type on the specified surface.
ASIN(x)	Numeric expression	Arc sine in radians.
ASPR()	(null)	Aspect ratio of the current graphics device.
ATAN(x)	Numeric expression	Arc tangent in radians.
ATYP()	(null)	System aperture type code: 0 for EPD, 1 for F/#, 2 for NA, and 3 for float by stop size.
AVAL()	(null)	System aperture value.
CALD(i)	index	Returns a numeric value from the CALLMACRO buffer at the specified index. See " Calling a Macro from within a Macro "
CHZN(x)	Valid Surface Number	Chip Zone of surface.
CONF()	(null)	Returns the current configuration number, which is between 1 and the number of configurations, inclusive.
CONI(x)	Valid surface number	Conic constant of the surface.
COSI(x)	Numeric expression in radians	Cosine of the expression.
CURV(x)	Valid surface number	Curvature of the surface.
EDGE(x)	Valid surface number	Edge thickness at the clear semi-diameter or semi-diameter of that surface.

EOFF()	(null)	End of file flag. Returns 1 if the end of file has been reached, otherwise returns 0. Only valid after execution of READ keyword.
ETIM()	(null)	The elapsed time in seconds since last TIMER.
EXPE(x)	Numeric expression	e to the power of the expression.
EXPT(x)	Numeric expression	10 to the power of the expression.
FICL(vec#)	Vector #, between 1 and 4 inclusive..	The fiber coupling efficiency. See "Using the FICL() function".
FLDX(x)	Valid field number	X-angle or height of specified field.
FLDY(x)	Valid field number	Y-angle or height of specified field.
FTAN()	Valid field number	Tangential Angle of specified field.
FTYP()	(null)	Returns 0 if the current field type is angles in degrees, 1 if object height, 2 if paraxial image height, 3 if real image height, and 4 if theodolite angles. Heights are always in lens units .
FVAN(x)	Valid field number	Vignetting angle of specified field. This command is obsolete. See FTAN().
FVCX(x), FVCY(x)	Valid field number	Vignetting compression x or y of specified field.
FVDX(x), FVDY(x)	Valid field number	Vignetting decenter x or y of specified field.
FWGT(x)	Valid field number	Field weighting of specified field.
GABB(x)	Valid surface number	The glass catalog Abbe number of the glass for the specified surface.
GAUS(x)	Standard deviation	Returns a random value with a Gaussian distribution, zero mean, and the specified standard deviation.
GETT(window, line, column)	The window number, line number, and column number of the numerical value to extract. Columns are delimited by spaces and tab characters.	Extracts a numerical value from any open text window. This feature allows computations using any value OpticStudio can display in a window. This function assumes that a period is the decimal separator. To use the character currently selected in Windows settings, use GETL().

GETL(window, line, column)	The window number, line number, and column number of the numerical value to extract. Columns are delimited by spaces.	Extracts a numerical value from any open text window. This feature allows computations using any value OpticStudio can display in a window. This function uses the decimal separator defined in the Windows "Region and Language" settings . Also see GETT().
GIND(x)	Valid surface number	The glass catalog d-light index of the glass for the specified surface.
GLCA(x)	Valid surface number	Global vertex x vector of the specified surface.
GLCB(x)	Valid surface number	Global vertex y vector of the specified surface.
GLCC(x)	Valid surface number	Global vertex z vector of the specified surface.
GLCM(surf, item)	Surf must be a valid surface number, item is an integer between 1 and 12.	For item equal to 1-9, the return value is R11, R12, R13, R21, R22, R23, R31, R32, or R33. For item equal to 10-12, the return value is the x, y, or z component of the global offset vector. See "Global Coordinate Reference Surface".
GLCX(x)	Valid surface number	Global vertex x-coordinate of the specified surface.
GLCY(x)	Valid surface number	Global vertex y-coordinate of the specified surface.
GLCZ(x)	Valid surface number	Global vertex z-coordinate of the specified surface.
GNUM(A$)	Any string variable name	If the string A$ is the name of a valid glass, such as BK7, then GNUM returns the number of the glass in the glass catalog. The glass number can subsequent l y be used by SETSURFACEPROPERTY to set the glass type on a surface. If A$ does not correspond to any glass in the catalog, GNUM returns 0. GNUM returns -1 if the string is "MIRROR".
GPAR(x)	Valid surface number	The glass catalog partial dispersion coefficient of the glass for the specified surface.
GRIN(s, w, x, y, z)	Surface #, wavelength #, x, y, and z coordinates	Returns the index of refraction at the specified x, y, z position on surfaces at wavelength number w. Works for gradient and non-gradient media.

GTEM(code)	For checkbox values, code is 0 for Use Glass Substitution Template, 1 for Exclude Glasses With Incomplete Data, 11 for Standard, 12 for Preferred, 13 for Obsolete, 14 for Special, and 21-26 for Use Relative Cost, CR, FR, SR, AR, and PR, respectively. For numerical values, code is 31-36 for Relative Cost, CR, FR, SR, AR, and PR, respectively.	Returns data from the Glass Substitution Template. Checkboxes return zero for false (unchecked) and any non-zero value for true (checked).
IMAE(seed)	The seed value	Returns the geometric image analysis efficiency. See the discussion "Geometric Image Analysis". If seed is zero, the same random numbers will be used on every call to IMAE. If seed is not zero, then every call to IMAE will use unique random numbers.
INDX(surface)	Valid surface number	Index of refraction at the primary wavelength. See ISMS.
INTE(x)	Numeric expression	Returns the largest integer not greater than the argument.
ISCS(surface)	Valid surface number	Returns zero if the surface is not composite and one of the surfaces is complete.
ISMS(surface)	Valid surface number	If the surface is an odd mirror, or follows an odd mirror but is not a mirror, the return value will be one, otherwise the return value is 0.
LOGE(x)	Positive numeric expression	Log base e of the expression.
LOGT(x)	Positive numeric expression	Log base ten of the expression.
LOST(code)	Code is 1 to return lost energy due to errors, or 2 for lost energy due to thresholds.	The lost energy following the most recent NSTR trace.
LVAL(A$)	Any string variable name	String value. Returns a floating point value of the string A$, in which the current user language settings are employed for interpreting the decimal delimiter in the string. Also see SVAL.
MAGN(x,y)	x and y are any real numbers	Computes the square root of x squared plus y squared.

MAXF()	(null)	The maximum radial angle in degrees, radial object height in lens units, or radial image height in lens units. Which value is returned depends upon if the fields are defined by angle, object height, or image height. Note for field angles: See " Field Type " for more information about normalization and the difference between the max field angle and the max radial angle.
MAXG()	(null)	The number of glasses currently loaded.

MCON(row, config, data)	Row (operand number) ,Configuration number, and data value to extract from the Multi-Configuration Editor.	Extracts data from any row and configuration of the Multi-Configuration Editor. This function is similar to MCOP with extended capabilities for extracting data. If the row and config number are both zero, MCON returns either the number of operands, the number of configurations, or the active configuration number for data = 0, 1, and 2, respectively. If the row number is between 1 and the number of multi-config operands, and the config number is zero, MCON returns the operand type, integer 1, integer 2, integer 3, and string flag for that specified row, for data = 0 through 4 respectively. The 3 integer values are used for various purposes for different operands, such as surface and wavelength numbers. The string flag is 1 if the operand data is a string value, such as a glass name, or 0 for numerical data. If the row number is between 1 and the number of multi-config operands, and the config number is valid, MCON returns either the numerical value or the string data for that operand. Note that all string data returned by MCON must be extracted with the $buffer command after the call to MCON. For example, the following code sample will place the name of the operand on row 1 in a$: dummy = MCON(1, 0, 0) a$ = $buffer() See also keyword SETMCOPERAND .
MCOP(row, config)	Row (operand number) and configuration of the Multi -Configuration Editor.	Extracts data from any row and configuration of the Multi-Configuration Editor. A configuration number of zero indicates the current configuration. See also keyword SETMCOPERAND .
MCSD(x)	Valid Surface number	Mechanical Semi-Diameter of surface.

MFCN()	(null)	MFCN updates the lens, validates the merit function, updates the merit function, then returns the current merit function value. See OPER.
NCON()	(null)	Returns the number of configurations.
NFLD()	(null)	The number of defined fields.
NOBJ(surface)	Valid non-sequential surface number	The number of defined objects in the non- sequential surface.
NPAR(surf, object, param)	surf is the surface number, object is the object number, and param is the parameter number of the value to return.	Returns a value from the parameter columns of the non-sequential component editor. See " SETNSCPARAMETER ".
NPOS(surf, object, code)	surf is the surface number, object is the object number, and code is 1-6 for reference object, x, y, z, tilt x, tilt y, and tilt z position to return, respectively.	Returns a value from the position columns of the non-sequential component editor. See " SETNSCPOSITION ".
NPRO(surf, object, code, face)	surf is the surface number, object is the object number, code values are as def ined for the SETNSCPROPERTY keyword, and face is the face number	Returns a numeric or string value from the property pages of objects defined in the non- sequential components editor. For information on the code values, see " SETNSCPROPERTY ". String values may be extracted using the $buffer() function after calling this function with a code that returns string data. For example, to extract the comment column of object 5 from NSC surface 2 use these two commands: dummy = NPRO(2, 5, 1, 0) a$ = $buffer() The value dummy may be ignored. The string variable a$ will contain the object comment.
NSDC(surf, object, pixel, data)	The surf value indicates the surface number of non-sequential group. The object number indicates the desired detector. Pixel indicates which pixel value is returned. Data is 0 for real, 1 for imaginary, 2 for the amplitude, and 3 for the coherent intensity. See also "NSTR"	If the object number corresponds to a detector rectangle object, then the coherent intensity data from the specified pixel is returned. Coherent intensity consists of 4 numbers which are each summed over all incident rays: the real part, the imaginary part, the amplitude, and the coherent intensity. If the pixel number is zero, then the sum of the data for all pixels for that detector object is returned.

NSDD(surf, object, pixel, data)	The surf value indicates the surface number of non-sequential group. The object number indicates the desired detector. See the discussion to the right for details on the pixel and data arguments. See also "NSTR".	If the object number is zero, then all detectors are cleared and the function returns zero. If the object number is less than zero, then the detector defined by the absolute value of the object number is cleared and the function returns zero. If the object number corresponds to a detector rectangle, surface, or volume object, then the incoherent intensity data from the specified pixel is returned. For a full discussion of the pixel and data arguments, see "NSDD". This ZPL function supports the same pixel and data arguments as the NSDD optimization operand. This function does not support specifying a value for the # Ignored argument provided by the NSDD optimization operand, and assumes that the value for # Ignored is always zero. Note that, not like the merit function operand NSDD, this ZPL numeric function NSDD() doesn't support pixel = -14/-15 as there is no way to pass the spatial frequency. For calculating geometric MTF in ZPL, please see GETNSCMTF .

NSDE(surf, object, pixel, angle, data, wavelength)	The surf value indicates the surface number of non-sequential group. The object number indicates the desired detector. See the discussion to the right for details on the pixel, angle, data, and wavelength arguments. See also "NSTR".	If the object number is zero, then all detectors are cleared and the function returns zero. If the object number is less than zero, then the detector defined by the absolute value of the object number is cleared and the function returns zero. If the object number corresponds to a detector color object, then the data from the specified pixel is returned. For a full discussion of the pixel, angle, data and wavelength arguments, see "NSDE" under the optimization operand. This ZPL function supports the same pixel, angle, data, and wavelength arguments as the NSDE optimization operand. This function does not support specifying a value for the # Ignored argument provided by the NSDD optimization operand, and assumes that the value for # Ignored is always zero.
NSDP(surf, object, pixel, data)	The surf value indicates the surface number of non-sequential group. The object number indicates the desired detector. See the discussion to the right for details on the pixel and data arguments. See also "NSTR".	If the object number is zero, then all detectors are cleared and the function returns zero. If the object number is less than zero, then the detector defined by the absolute value of the object number is cleared and the function returns zero. If the object number corresponds to a detector polar object, then the data from the specified pixel is returned. For a full discussion of the pixel and data arguments, see "NSDP" This ZPL function supports the same pixel and data arguments as the NSDP optimization operand.
NSUR()	(null)	The number of defined surfaces.
NWAV()	(null)	The number of defined wavelengths.
OBJC(A$)	Any string variable name	Object with comment. Returns the first non- sequential object number where the comment matches the string A$. The comparison is case insensitive. If no NSC object has the matching comment the function returns -1.

OCOD(A$)	Any string variable or literal that is the name of a Optic Studio optimization operand.	The optimization operand code number used by the OPEV function.
ONUM(A$)	Any string variable name	If the string A$ is the name of a valid optimization operand, such as EFFL, then ONUM returns the id number of the operand. The operand id number can subsequently be used by SETOPERAND to set the operand type in the merit function editor. If A$ does not correspond to any operand name, ONUM returns 0.
OPDC()	(null)	The optical path difference. Valid only after a RAYTRACE call. OPDC will not return valid data if the chief ray cannot be traced. RAYTRACEX does not support OPDC.
OPER(row, col)	Row (operand number) and column (data type) of the Merit Function Editor.	Extracts data from any row and column of the Merit Function Editor. The row is the same as the operand number; the column is 1 for type, 2 for int1, 3 for int2, 4-7 for data1-data4, 8 for target, 9 for weight, 10 for value, 11 for percent contribution, and 12-13 for data5-data6. To obtain the comment from a BLNK operand, use column number 2. The comment is then placed in the string buffer. For example, to obtain the comment from row 1: dummy = OPER(1,2) a$ = $buffer() The value in the dummy variable may be ignored, and a$ will then contain the comment string. OPER does not update the lens or the merit function, it returns the current data. See also OPEV, MFCN and keyword SETOPERAND .

OPEV(code, int1, int2, data1, data2, data3, data4)	Code is the optimization operand code (see function OCOD), and int1-int2, and data1-data4 are the defining values for the operand. See Optimization Operands.	Computes the same value as any optimization operand would, without the need to add the operand to the merit function. This is useful for computing numbers already available from the optimization operands. For example, to compute the EFL from the EFFL operand use this code: C = OCOD("EFFL") E = OPEV(C, 0, 1, 0, 0, 0, 0) Make sure to fill all 7 inputs argument for OPEV. If the optimization operand requires less than 6 parameters use additional 0s as in the example above. See also OPER, OCOD, MFCN and keyword SETOPERAND. Some optimization operands, such as OPGT, only make sense when taken in the context of other operands. OPEV will only work for operands that do not depend upon the presence of prior operands. See also OPEW below.
OPEW(code, int1, int2, data1, data2, data3, data4, data5, data6)	Code is the optimization operand code (see function OCOD), and int1-int2, and data1-data6 are the defining values for the operand. See Optimization Operands.	This function is very similar to OPEV, the key difference being that OPEW supports two additional arguments. Some optimization operands use up to 6 data values rather than 4.
OPTH(x)	Valid surface number	The optical path length along the ray to the specified surface. Unlike RAYT and RAYO, OPTH considers the phase added by diffractive surfaces such as gratings, holograms, and binary optics. Valid only after a RAYTRACE call. OPTH will not return valid data if the chief ray cannot be traced. RAYTRACEX does not support OPTH.
PARM(n,s)	Valid parameter number and surface number	Parameter "n" of surface "s".
PIXX(A$)	A string variable containing the full path to a BMP, JPG or PNG graphic file.	The number of X-direction pixels in the graphic.
PIXY(A$)	A string variable containing the full path to a BMP, JPG or PNG graphic file.	The number of Y-direction pixels in the graphic.
PMOD()	(null)	0 if paraxial mode is off, else 1.

POWR(x,y)	x and y are any numbers	Computes the absolute value of x to the power of y.
PVEX()	(null)	Data5 parameter from the ZPLM optimization operand.
PVEY()	(null)	Data6 parameter from the ZPLM optimization operand.
PVHX()	(null)	Data1 parameter from the ZPLM optimization operand.
PVHY()	(null)	Data2 parameter from the ZPLM optimization operand.
PVPX()	(null)	Data3 parameter from the ZPLM optimization operand.
PVPY()	(null)	Data4 parameter from the ZPLM optimization operand.
PWAV()	(null)	The primary wavelength number.
RADI(x)	Valid surface number	Radius of curvature of surface. If the surface has an infinite radius, RADI returns 0.0. This possibility must be considered to avoid potential divide by zero errors.
RAGL(x)	Valid surface number	The global X-direction cosine of the ray following the surface. Valid only after a RAYTRACE or RAYTRACEX call.
RAGM(x)	Valid surface number	The global Y-direction cosine of the ray following the surface. Valid only after a RAYTRACE or RAYTRACEX call.
RAGN(x)	Valid surface number	The global Z-direction cosine of the ray following the surface. Valid only after a RAYTRACE or RAYTRACEX call.
RAGX(x)	Valid surface number	The global x coordinate of the ray intercept. Valid only after a RAYTRACE or RAYTRACEX call.
RAGY(x)	Valid surface number	The global y coordinate of the ray intercept. Valid only after a RAYTRACE or RAYTRACEX call.
RAGZ(x)	Valid surface number	The global z coordinate of the ray intercept. Valid only after a RAYTRACE or RAYTRACEX call.

RAND(x)	Numeric expression	Random floating point number uniformly distributed between 0 and the expression. It is recommend that the RANDOMIZE keyword be called before this function to change the random number seed. In particular, if NSC raytrace operands (NSTR) are being used, RANDOMIZE is required to re-seed the random number generator.
RANX(x)	Valid surface number	The X-direction cosine of the surface normal. Valid only after a RAYTRACE or RAYTRACEX call.
RANY(x)	Valid surface number	The Y-direction cosine of the surface normal. Valid only after a RAYTRACE or RAYTRACEX call.
RANZ(x)	Valid surface number	The Z-direction cosine of the surface normal. Valid only after a RAYTRACE or RAYTRACEX call.
RAYE()	(null)	The ray-trace error flag, 0 if no error. Valid only after a RAYTRACE or RAYTRACEX call. See the RAYTRACE keyword for details.
RAYL(x)	Valid surface number	The X-direction cosine of the ray following the surface. Valid only after a RAYTRACE or RAYTRACEX call.
RAYM(x)	Valid surface number	The Y-direction cosine of the ray following the surface. Valid only after a RAYTRACE or RAYTRACEX call.
RAYN(x)	Valid surface number	The Z-direction cosine of the ray following the surface. Valid only after a RAYTRACE or RAYTRACEX call.
RAYO(x)	Valid surface number	The ray optical path length from the previous surface to the specified surface. The optical path length is the path length times the index of refraction, either or both of which may be negative. For rays inside a non-sequential surface, RAYO returns the sum of the path length times the index of refraction of all objects the ray passed through.Valid only after a RAYTRACE or RAYTRACEX call. See also OPTH and RAYT.

RAYT(x)	Valid surface number	The ray path length from the previous surface to the specified surface. The path length may be negative. Valid only after a RAYTRACE or RAYTRACEX call. See also OPTH and RAYO. For rays inside a non-sequential surface, RAYT returns the sum of the path lengths along all segments. See also RAYO.
RAYV()	(null)	0 if ray was not vignetted, else vignetted surface number. Valid only after a RAYTRACE or RAYTRACEX call.
RAYX(x)	Valid surface number	The X-coordinate of the ray intercept. Valid only after a RAYTRACE or RAYTRACEX call.
RAYY(x)	Valid surface number	The Y-coordinate of the ray intercept. Valid only after a RAYTRACE or RAYTRACEX call.
RAYZ(x)	Valid surface number	The Z-coordinate of the ray intercept. Valid only after a RAYTRACE or RAYTRACEX call.
RELI(f)	Field number	RI for the specified field position.
SAGG(x,y,z)	x,y are the coordinates in lens units at the surface number z	Computes the sag in lens units at the specified point on the surface.
SCOM(A$, B$)	Any two string variable names	If the strings are equal, SCOM returns 0. If A$ is less than B$, then SCOM returns a value less than 0; otherwise, a value greater than 0.
SDIA(x)	Valid surface number	Clear semi-diameter or Semi-diameter of surface.
SIGN(x)	Numeric expression	Returns -1 if the argument is less than zero, 0 if the argument is zero, and +1 if the argument is positive.
SINE(x)	Numeric expression in radians	Sine of the expression.
SLEN(A$)	Any string variable name	Returns the length of the string variable A$. The length is defined as the number of characters in the string variable.

SOLM(operand, config, code)	Operand is the multi-configuration operand number. Config is the configuration number. Code is 0 for the status, 1 for the pickup configuration number, 2 for the pickup operand row, 3 for the pickup scale, 4 for the pickup offset, 5 for the current numerical or string value of the operand, 6 for the ZPL macro solve name, and 7 for the glass catalog name. Not all codes are used by all solve status types.	The return value is data about the solve type and data for the specified multi configuration operand and configuration. If the code is 0 for status, the returned value is 0 for fixed, 1 for variable, 2 for pick-up, 3 for thermal pick-up, 4 for ZPL Macro, and 5 for glass substitution solve. String values may be extracted using the $buffer() function after calling this function with a code that returns string data. For more information on the solve data and the meaning of the solve parameters, see Solve Types (multiple configuration editor).
SOLV(surf, code, param)	Surf is the surface number. Code is 0 for curvature, 1 for thickness, 2 for glass, 3 for conic, 4 for semidiameter, and 5 for TCE. For parameter data, the code is 100 plus the parameter number. For extra data, the code is 300 plus the extra data number. Param is an integer between 0 and 4, inclusive.	The return value is data about the solve type for the specified surface and data. If param is zero, then an integer corresponding to the solve type is returned. For param between 1 and 3, the data is the solve parameters. For param 4, the data is the pickup column number. String values may be extracted using the $buffer() function after calling this function with a code that returns string data. For more information on the solve data and the meaning of the solve parameters, see Solve Types (lens data editor).
SOSO(code)	Code is 0 to return the surface number o f the macro solve currently executing, or 1 to return the object number.	This function returns the surface number and/or object number of the macro currently being executed as a solve. If the macro is not being used as a solve, these functions return meaningless data.

SPRO(surf, code)	Surf is the surface number. The code values are as defined for the SETSURFACEPROPERTY keyword. Only integer code values, and not mnemonics are supported.	Returns a numeric or string value from the property pages or editors of surfaces defined in the lens data editor. For information on the code values, see " SETSURFACEPROPERTY, SURP ". String values may be extracted using the $buffer() function after calling this function with a code that returns string data. For example, to extract the comment column of surface 4 use these two commands: dummy = SPRO(4, 1) a$ = $buffer() The value dummy may be ignored. The string variable a$ will contain the surface comment. Properties where SETSURFACEPROPERTY requires the use of value2 are not supported by SPRO, use SPRX instead. These properties include parameter data, extra data, and coating layer multiplier and index offset values and status.
SPRX(surf, code, value2)	Surf is the surface number. The code and value2 values are as defined for the SETSURFACEPROPERTY keyword. Only integer code values, and not mnemonics are supported.	This function is similar to SPRO. The key difference is that SPRX supports one additional argument. Only parameters set by SETSURFACEPROPERTY that use the "value2" argument should use SPRX. See " SETSURFACEPROPERTY, SURP "
SRCN(A$, n)	A$ is any string variable name, n is the n*th* matching surface number to return.	Surface matching comment #n. Returns the n*th* surface number where the comment matches the string A$. The comparison is case insensitive. When n is positive, the search is in the forward direction (0 to # of surfaces), and when n is negative the search direction is backwards (# of surfaces to 0). If there are not n matching comments the function returns -1. When n is zero, returns the count of matching surface comments.
SQRT(x)	Positive numeric expression	Square root of the expression.
STDD() - obsolete	This function is obsolete, use SRPO instead.	

SURC(A$)	Any string variable name	Surface with comment. Returns the first surface number where the comment matches the string A$. The comparison is case insensitive. If no surface has the matching comment the function returns -1. When tolerancing, the SURC function will keep returning the original surface number even if additional surfaces are added automatically. To avoid this behaviour, use the numeric function SRCN instead.
SVAL(A$)	Any string variable name	String value. Returns a floating point value of the string A$, assuming that the period symbol is the decimal delimiter in the string. Also see LVAL.
SYPR(code)	The code values are as defined for the SETSYSTMPROPERTY keyword. Only integer code values, and not mnemonics are supported.	Returns a numeric or string value for the corresponding system data. For information on the code values , see " SETSYSTEMPROPERTY, SYSP ". String values may be extracted using the $buffer() function after calling this function with a code that returns string data. For example, to extract the lens title use these two commands: dummy = SYPR(16) a$ = $buffer() The value dummy may be ignored. The string variable a$ will contain the lens title. Properties where SETSYSTEMPROPERTY requires the use of value2 are not supported by SYPR.
TANG(x)	Numeric expression in radians	Tangent of the expression.
Function	Argument	Return value
TMAS()	(null)	The total mass in grams of the lens from surface 1 to the image surface.
THIC(x)	Valid surface number	Thickness of the surface.

TOLV(op, col)	The integer op is the tolerance operand number, col is the column number in the Tolerance Data Editor. See discussion.	Returns a numeric or string value from the Tolerance Data Editor. If col is 1, 2, or 3, the return value is the integer 1, 2, or 3 value as used by that operand type. If col is 4 or 5, the min or max value of the tolerance is returned. If col is 6, the "Do Not Adjust During Inverse Tolerancing" flag is returned (1 for checked, 0 for unchecked). If col is 7, the "Ignore this Operand During Tolerancing" flag is returned (1 for checked, 0 for unchecked). If col is 90, the nominal value of the tolerance is returned. If the col value is 0 or 99, the operand type name or operand comment is placed in the buffer string, respectively. For example, to extract the operand name of operand 7 use these two commands: dummy = TOLV(7, 0) a$ = $buffer() The value dummy may be ignored. The string variable a$ will contain the operand name. See also the string functions $TOLOPERAND and $TOLCOMMENT which perform a similar function.
UNIT()	(null)	Returns 0, 1, 2, or 3, if the current unit type is millimeters, centimeters, inches, or meters, respectively.
VEC1(x)	Positive subscript value	Returns the value of the array variable at the specified subscript.
VEC2(x)	Positive subscript value	Returns the value of the array variable at the specified subscript.
VEC3(x)	Positive subscript value	Returns the value of the array variable at the specified subscript.
VEC4(x)	Positive subscript value	Returns the value of the array variable at the specified subscript.
VERS()	(null)	Returns the version of the OpticStudio program. The version is of the form yymmdd. For example, the version for the June 15, 2005 release of OpticStudio will be 050615.
WAVL(x)	Valid wavelength number	Wavelength in micrometers.

WINL()	(null)	Returns the number of the analysis window most recent ly opened by the keyword OPENANALYSISWINDOW.
WINN()	(null)	Returns the number of open analysis windows.
WWGT(x)	Valid wavelength number	Wavelength weighting.
XMIN()	(null)	The minimum x-coordinate in the graphics window.
XMAX()	(null)	The maximum x-coordinate in the graphics window.
Function	Argument	Return value
YMIN()	(null)	The minimum y-coordinate in the graphics window.
YMAX()	(null)	The maximum y-coordinate in the graphics window.

10.2.12. Using the FICL() Function

The FICL function computes fiber coupling. Because the function takes so many arguments, the arguments are placed in a vector array and then the function is called. The vector array must be one of the four VEC arrays already defined. The values placed in the vector array are defined as follows:

```
0 = Sampling (1 for 32 x 32, 2 for 64 x 64, 3 for 128 x 128, etc..)
1 = Wavelength #
2 = Field #
3 = Ignore source flag (0 for false, 1 for true)
4 = Source NA x-direction
5 = Receiver NA x-direction
6 = Source x angle (deg)
7 = Source y angle (deg)
8 = Receiver tilt about x (deg
9 = Receiver tilt about y (deg)
10 = Receiver decenter x
11 = Receiver decenter y
12 = Receiver decenter z
13 = Align source to chief ray flag (0 for false, 1 for true)
14 = Align receiver to chief ray flag (0 for false, 1 for true)
15 = Use polarization flag (0 for false, 1 for true)
16 = Source NA y-direction (if zero then NAy = NAx)
17 = Receiver NA y-direction (if zero then NAy = NAx)
18 = Use Huygens integral (0 for false, 1 for true)
```

The fiber coupling is computed by calling FICL(n) where n is the vector number containing the argument list.

10.2.13. String Functions

The following table lists the available string functions.

ZPL STRING FUNCTIONS

Function	**Description**

$BUFFER()	Returns the current string in the lens buffer. This function is used to extract string data from various ZPL keywords and functions.
$CALLSTR(i)	Returns the string from the CALLMACRO string buffer at index i. See "Calling a Macro from within a Macro".
$COAT(i)	Returns the coating name for the ith surface.
$COATINGPATH()	Returns the path name for coating files.
$COMMENT(i)	Returns the comment string for the ith surface.
$DATAPATH()	Returns the path name for data files.
$DATE()	Returns the current date and time string. The formatting is specified by the Date/Time control settings in the General section of the OpticStudio Preferences.
$EXTENSIONPATH()	Returns the path name for OpticStudio extensions.
$FILENAME()	Returns the current lens file name, without the path.
$FILEPATH()	Returns the current lens file name, with the complete path.
$GETSTRING(A$, n)	Returns the nth sub-string for the string A$ using spaces for delimiters. For example, if A$ = "one two three", then $GETSTRING(A$, 2) returns "two". Input A$ must be a string variable defined prior to calling this function.
$GETSTRINGC(A$, n)	Returns the nth sub-string for the string A$ using commas for delimiters. For example, if A$ = "one,two,three", then $GETSTRING(A$, 2) returns "two". Input A$ must be a string variable defined prior to calling this function.
$GLASS(i)	Returns the glass name of surface number i.
$GLASSCATALOG(i)	Returns the name of the ith loaded glass catalog for the current lens. If i is less than 1, then the names of all the loaded catalogs separated by spaces are returned in a single string.
$GLASSPATH()	Returns the path name for glass catalog files.
$LEFTSTRING(A$, n)	Returns the left most n characters in the string A$. If A$ has fewer than n characters, the remaining spaces will be padded with blanks. This allows formatting of strings with a fixed length. Input A$ must be a string variable defined prior to calling this function.
$LENSNAME()	Returns the lens title defined in the System Explorer.
$MACROPATH()	Returns the path name for macro files.
$NEWLINE()	Returns the tab character (\n). This function should primarily be used to update the Title/Notes section in the System Explorer.

$NOTE(line#)	Returns the notes information defined in the System Explorer. Because the notes may be very long, $NOTE returns the characters from the notes in groups called lines. A line ends when a newline (carriage return) character is found, or when the total number of consecutive characters on the line reaches 100, whichever comes first. The line# indicates which line in the notes is to be returned. For example, $NOTE(1) returns the first line in the notes, up to the first newline character. If there are more than 100 consecutive characters before the first newline is found, then line 1 will be the first 100 characters, and line 2 will be the remainder of the line up to the first newline, or the next 100 characters, whichever comes first. The maximum number of characters in the notes is currently 4000 (because of space limitations, it may not be possible to edit this many characters in the Title/Notes section of the System Explorer however). $NOTE will return a null (empty) string if there are no defined characters in the notes for the specified line. The function SLEN can be used to determine the actual number of characters returned by $NOTE using this syntax: A$ = $NOTE(1) N = SLEN(A$) The value N will be the integer number of characters in the string A$.
$OBJECTPATH()	Returns the path name for NSC object files.
$PATHNAME()	Returns the path name only for the current lens file. This is useful for determining the folder where the lens file is stored.
$PROGRAMPATH()	Returns the path name for program files.
$QUOTE()	Returns the double quote character (").
$RIGHTSTRING(A$, n)	Returns the right most n characters in the string A$. If A$ has fewer than n characters, the remaining spaces will be padded with blanks. This allows formatting of strings with a fixed length. Input A$ must be a string variable defined prior to calling this function.
$STR(expression)	Returns a string formatted using the format defined by the FORMAT keyword. The numeric expression may be any equation, including combinations of constants, variables, and functions. See function SVAL(A$) to convert strings to numbers.
$TAB()	Returns the tab character (\t).

$TEMPFILENAME()	Returns the name of a temporary file, with complete path, suitable for temporary storage of text or binary data. See keyword GETTEXTFILE. Use the DELETEFILE keyword to delete temporary files as they are not automatically deleted.
$TOLCOMMENT(operand)	Returns the comment for the specified tolerance operand.
$TOLOPERAND(operand)	Returns the operand name for the specified tolerance operand.
$UNITS()	Returns either MM, CM, IN, or M, depending upon the current lens units.

10.2.14. KEYWORDS (about the zpl)

Keywords provide the capability to direct program flow, to generate output, and to perform certain crucial tasks, such as ray tracing and modifying the lens prescription. Each keyword is described in detail in the following sections.

10.2.14.1. APMN, APMX, APTP, APXD, APYD

These commands are obsolete. See " SETSURFACEPROPERTY, SURP " .

10.2.14.2. ATYP, AVAL

These commands are obsolete. See " SETSYSTEMPROPERTY, SYSP ".

10.2.14.3. BEEP

Makes an audible beep sound.

Syntax:

```
BEEP
```

Discussion:

This command can be used to alert the user when a calculation is finished or input is required.

10.2.14.4. BROWSE

BROWSE displays a window that prompts the user to select or enter a file name and populates a string variable with the chosen file path.

Syntax:

```
BROWSE "prompt", variable, mode
BROWSE "prompt", variable, mode, "extension"
```

Discussion:

The variable may be any valid string variable name. The mode argument is an integer that can be either 0 for 'load mode' which prompts the user for an existing file or 1 for 'save mode' which prompts the user for a file name to save as. The string "extension" provides the option to filter the files shown by the designated extension (e.g. ".CFG").

Note that the BROWSE function is meant to allow the user to select a file and populate a string variable with the file path and will not automatically open or save files.

Example:

```
! Load
BROWSE "File:",A$,0,".txt"
OPEN A$
PRINT "Reading file:"
PRINT A$
READ x
READ y
READ z
CLOSE

! Save
BROWSE "Save Merit Function as: ",B$,1,".mf"
SAVEMERIT B$
PRINT "Saving file as: ",B$
```

Related keywords:

INPUT

10.2.14.5. CALLMACRO

Calls another ZPL macro.

Syntax:

```
CALLMACRO filename.zpl
CALLMACRO name$
```

Discussion:

This command is used to call another ZPL macro. The filename should be the full macro name without the path. The file must exist in the folder for ZPL macro files. When the macro returns, any text output generated by the called macro will be copied into the calling macro's text output window. See "Calling a Macro from within a Macro".

10.2.14.6. CALLSETDBL

Sets a double precision numeric value in the master macro's buffer.

Syntax:

```
CALLSETDBL index, value
```

Discussion:

This command is used to set a numeric value into the buffer maintained by the master macro. The index can be any value between 0 and 50, inclusive. The numeric value may be retrieved using the CALD function. See "Calling a Macro from within a Macro".

10.2.14.7. CALLSETSTR

Sets a string value in the master macro's buffer.

Syntax:

```
CALLSETSTR index, text$
```

Discussion:

This command is used to set a string value into the buffer maintained by the master macro. The index can be any value between 0 and 50, inclusive. The string value may then be retrieved using the $CALLSTRING function. See " Calling a Macro from within a Macro ".

10.2.14.8. COAT

For setting, coating surface properties use the topic " SETSURFACEPROPERTY, SURP " keyword, and for getting coating surface properties use the $COAT(i) String Function as discussed under String Functions.

10.2.14.9. COMPOSITEOFFAXISAPERTUREON (keywords)

Set the surface to be an Add-on Composite

Syntax:

```
COMPOSITEON surf
```

Discussion:

Checks the Composite property option "Composite Surface: Add sag to the next surface" to make the surface an Add-on composite.

10.2.14.10. COMPOSITEOFF (keywords)

Avoid the surface to be an Add-on Composite

Syntax:

```
COMPOSITEOFF surf
```

Discussion:

Unchecks the Composite property option "Composite Surface: Add sag to the next surface" to avoid the surface being an Add-on composite.

10.2.14.11. COMPOSITEON (keywords)

Set the surface to be an Add-on Composite

Syntax:

```
COMPOSITEON surf
```

Discussion:

Checks the Composite property option "Composite Surface: Add sag to the next surface" to make the surface an Add-on composite.

10.2.14.12. CLOSE

Closes a file previously opened by the OPEN command.

Syntax:

```
CLOSE
```

Discussion:

See the description for OPEN .

10.2.14.13. CLOSEWINDOW

Suppresses the display of the default output window. This keyword can also be used to close any open analysis windows.

Syntax:

```
CLOSEWINDOW
CLOSEWINDOW n
```

Discussion:

CLOSEWINDOW (with no argument "n" provided) is used to run the ZPL macro in "quiet" mode. The text window normally displayed at the end of the macro execution will not be displayed if the CLOSEWINDOW keyword is included at any line in the macro. CLOSEWINDOW has no other effect on macro execution.

CLOSEWINDOW (with an integer argument "n" provided) will close analysis window number n.

10.2.14.14. COLOR (keywords, about the zpl)

Sets the current pen color to use for graphics line and text functions.

Syntax:

```
COLOR n
```

Discussion:

The value n must evaluate to an integer corresponding to the desired color. The new pen color will be used for all subsequent line and text commands in graphics mode. Black is color 0, the other colors are as defined in "Colors 1-12, Colors 13-24". There are 24 colors other than black available.

10.2.14.15. COMMAND

Executes a shell command.

Syntax:

```
COMMAND executable, arguments
```

Discussion:

The values executable and arguments are either string literals or string variables. The value of executable is the name (including folder path) to the executable to run. The value of arguments is optional, and includes any command line arguments to the executable.

Once the command is launched, the executable runs in its own thread. This command does not cause the ZPL macro or OpticStudio to "wait" until the executable completes its task. No error codes are returned, and OpticStudio has no way of knowing if the shell command was launched successfully.

10.2.14.16. COMMENT (keywords, about the zpl)

This command is obsolete. See " SETSURFACEPROPERTY, SURP ".

10.2.14.17. CONI

For setting surface properties use the " SETSURFACEPROPERTY, SURP " keyword, and for getting surface properties use the SPRO(surf, code) Numeric Function as discussed under Numeric Functions.

10.2.14.18. CONVERTFILEFORMAT

Converts a text file from ANSI to Unicode format or from Unicode to ANSI format.

Syntax:

```
CONVERTFILEFORMAT filename, encoding
```

Discussion:

Filename is the name of the text file to convert. Encoding is 1 to convert the file from Unicode to ANSI, or 2 to convert ANSI to Unicode. Note converting a Unicode file into ANSI may cause the loss of data, if any of the characters have no ANSI equivalent.

To convert the file, first close the file (for example using OUTPUT SCREEN) and then use the CONVERTFILEFORMAT keyword.

10.2.14.19. CONVERTIMAGETOGRID

CONVERTIMAGETOGRID Converts monochromatic image files (.BMP, .JPG, .TIFF, .PNG, etc) to .DAT files that can be used with the Grid Phase surface. For RGB images, the R channel is used for the conversion. Performs the Bitmap Image to OpticStudio DAT feature from within a macro.

Syntax:

```
CONVERTIMAGETOGRID filename, delta x, delta y, unit flag, offset x, offset y
```

Discussion:

The filename is the full path of the file to be converted. Delta x and delta y are the width and height between pixels in units specified by the unit flag. The unit flag is 0 for millimeters, 1 for centimeters, 2 for inches, and 3 for meters. Offset x and offset y may be used to decenter the image.

Example:

```
A$ = "C:\pictures\opticstudio.bmp"
CONVERTIMAGETOGRID A$, 0.1, 0.1, 0, 0, 0
```

10.2.14.20. COPYFILE

COPYFILE is used to make a copy of a file.

Syntax:

```
COPYFILE sourcefilename, newfilename
```

Discussion:

This keyword requires two file names, defined as literal string expressions in quotes or as string variables. The file sourcefilename is copied into the new file newfilename. If newfilename already exists it is overwritten without warning.

Example:

```
COPYFILE "C:\source.dat", "C:\copy.dat"
```

Related Keywords:

DELETEFILE

RENAMEFILE

10.2.14.21. CURV

For setting surface properties use the " SETSURFACEPROPERTY, SURP " keyword, and for getting surface properties use the SPRO(surf, code) Numeric Function as discussed under Numeric Functions.

10.2.14.22. DECLARE

See " Array variables ".

10.2.14.23. DEFAULTMERIT

Generates a default merit function.

Syntax:

```
DEFAULTMERIT type, data, reference, method, rings, arms, grid, delete, axial,
lateral, start, xweight, oweight, pup_obsc
```

Discussion:

This keyword generates a default merit function in the Merit Function Editor. Any existing default merit function will be deleted. For details see "Modifying the merit function". The values are as follows:

type: use 0 for RMS, 1, for PTV.

data: use 0 for wavefront, 1 for spot radius, 2 for spot x, 3 for spot y, 4 for spot x + y. reference: use 0 for centroid, 1 for chief, 2 for unreferenced.

method: use 1 for Gaussian quadrature, 2 for rectangular array.

rings: the number of annular rings (Gaussian quadrature only).

arms: the number of radial arms (Gaussian quadrature only). The number of arms must be even and no less than 6.

grid: the size of the grid. Use an integer, such as 8, for an 8 x 8 grid. n must be even and no less than 4. delete: use 0 to not delete vignetted rays, 1 to delete vignetted rays.

axial: use -1 for automatic, which will use symmetry only if the system is axial symmetric. Use 1 to assume axial symmetry, 0 to not assume axial symmetry.

lateral: use 1 to ignore lateral color, 0 otherwise.

start: use -1 for automatic, which will add the default merit function after any existing DMFS operand. Otherwise use the operand number at which to add the default merit function. Any existing operands above the specified operand number will be retained.

xweight, oweight: the x direction weigh and overall weight for the merit function. Only the data "spot x + y" uses the xweight value.

pup_obsc: the pupil obscuration ratio.

10.2.14.24. DELETE

Deletes a surface from the spreadsheet.

Syntax:

```
DELETE n
```

Discussion:

The value n must evaluate to an integer expression. See also INSERT .

Example:

```
DELETE 5
DELETE i+2*j
```

10.2.14.25. DELETECONFIG (keywords)

DELETECONFIG deletes an existing configuration in the multi-configuration editor.

Syntax:

```
DELETECONFIG config
```

Discussion:

The value config must evaluate to an integer expression greater than 0 and less than or equal to the current number of configurations. See also INSERTCONFIG and DELETEMCO .

10.2.14.26. DELETEFILE

DELETEFILE is used to delete a file.

Syntax:

```
DELETEFILE filename
```

Discussion:

This keyword requires a file names, defined as literal string expression in quotes or as a string variable.

Example:

```
DELETEFILE XFILE$
```

Related Keywords:

COPYFILE RENAMEFILE

10.2.14.27. DELETEMCO (keywords)

DELETEMCO deletes an existing operand in the multi-configuration editor.

Syntax:

```
DELETEMCO row
```

Discussion:

The value row must evaluate to an integer expression greater than 0 and less than or equal to the current number of operands. See also INSERTMCO and DELETECONFIG.

10.2.14.28. DELETEMFO (keywords)

DELETEMFO deletes an existing operand in the merit function editor.

Syntax:

```
DELETEMFO row
DELETEMFO ALL
```

Discussion:

The value row must evaluate to an integer expression greater than 0 and less than or equal to the current number of operands. If the ALL argument is used all operands are deleted. See also INSERTMFO.

10.2.14.29. DELETEOBJECT (keywords)

Deletes an existing NSC object.

Syntax:

```
DELETEOBJECT surf, object
```

Discussion:

The value surf must evaluate to an integer expression, and the specified surface must be a non-sequential component surface. Use a surf value of 1 if the program mode is non-sequential. The value object must evaluate to an integer that is between 1 and the current number of objects plus 1, inclusive. The specified object will be deleted, renumbering other objects as required. See also INSERTOBJECT and SETNSCPROPERTY.

Example:

```
DELETEOBJECT 1, 1
```

10.2.14.30. DELETETOL

DELETETOL deletes an existing operand in the tolerance data editor.

Syntax:

```
DELETETOL row
```

Discussion:

The value row must evaluate to an integer expression greater than 0 and less than or equal to the current number of operands. See also INSERTTOL and SETTOL.

10.2.14.31. EDVA

For setting surface extra data properties use the topicSETSURFACEPROPERTY, SURP keyword, and for getting surface extra data properties use the topic"GETEXTRADATA.

10.2.14.32. END

See GOSUB .

10.2.14.33. EXPORTBMP

Exports any graphic window as a BMP file.

Syntax:

```
EXPORTBMP winnum, filename, timeout
```

Discussion:

The winnum value may be either an integer or an expression that evaluates to an integer. The integer winnum corresponds to the graphic window number that should be saved to a file. OpticStudio numbers windows sequentially as they are opened, starting with 1. Any closed windows are deleted from the window list, without renumbering the windows which remain. Any windows opened after another window has been closed will use the lowest window number available. The filename should be the full file name including the path, but with no extension. OpticStudio will automatically add the BMP extension. The optional timeout parameter specifies a time timeout in milliseconds. For some complex graphics, a timeout is required to allowed the graphic to be completely redrawn and the screen capture to complete. If the BMP files appear incomplete, try a timeout value of 500 - 2500 milliseconds. See also EXPORTJPG .

Example:

```
EXPORTBMP 1, "C:\TEMP\MYBMPFILE"
EXPORTBMP k, A$
```

10.2.14.34. EXPORTCAD (keywords)

Exports lens data as an IGES, STEP, SAT, or STL file for import into CAD programs.

Syntax:

```
EXPORTCAD filename
```

Discussion:

For a full description of CAD file export, see " CAD Files ".

That tool is described in The File Tab > Export Group > CAD Files.

The filename can be a literal string, such as "C:\DATA\FILE.IGS" or a string variable name, such as MYFILENAME$. There are many other parameters required to define the export. These other data are placed in the vector 1 variable as follows:

VEC1(1): The file type. Use 0 for IGES, 1 for STEP, 2 for SAT, 3 for STL.

VEC1(2): The number of spline points to use (if required on certain entity types). Use 16, 32, 64, 128, 256, or 512.

VEC1(3): The First surface to export. In NSC Mode, this is the first object to export.

VEC1(4): The last surface to export. In NSC Mode, this is the last object to export.

VEC1(5): The layer to place ray data on.

VEC1(6): The layer to place lens data on.

VEC1(7): Use 1 to export dummy surfaces, otherwise use 0.

VEC1(8): Use 1 to export surfaces as solids, otherwise use 0.

VEC1(9): Ray pattern. Use 0 for XY, 1 for X, 2 for Y, 3 for ring, 4 for list, 5 for none, 6 for grid, and 7 for solid beams.

VEC1(10): The number of rays.

VEC1(11): The wave number. Use 0 for all.

VEC1(12): The field number. Use 0 for all.

VEC1(13): Use 1 to delete vignetted rays, otherwise use 0.

VEC1(14): The dummy surface thickness in lens units.

VEC1(15): Use 1 to split rays from NSC sources, otherwise use 0.

VEC1(16): Use 1 to scatter rays from NSC sources, otherwise use 0.

VEC1(17): Use 1 to use polarization when tracing NSC rays, otherwise use 0. Polarization is automatically selected if splitting is specified.

VEC1(18): Use 0 for the current configuration, 1 to n for a specific configuration where n is the total number of configurations, n+1 to export "All By File", n+2 to export "All By Layer", and n+3 for "All At Once". For a more detailed explanation of the configuration setting, see "Export CAD File".

VEC1(19): Tolerance setting. Use 0 for 1.0E-4, 1 for 1.0E-5, 2 for 1.0E-6, and 3 for 1.0E-7. Note that if any other value is entered it will default to 1E-4.

VEC1(20), VEC1(21): If the program mode is a mixed sequential/non-sequential mode (meaning the range of surfaces includes a non-sequential component surface), these values allow a range of the non-sequential objects to be exported. VEC1(20) is the first object to export, and VEC1(21) is the last object to export. If both values are zero or out of range all non-sequential objects are exported.

VEC1(22)=1: Use 1 to export ray footprint data.

VEC1(23)=1: The layer to place ray footprint data on.

VEC1(24)=1: Use 1 to export Chief Ray.

In a NSC mode to export specific rays using filter strings, additional string parameter which is the "Filter String" needs to be given after the filename. example: `EXPORTCAD "C:\TEMP\MYCADFILE.STP", "FilterString"`

Example:

```
VEC1(1) = 0
VEC1(2) = 32
VEC1(3) = 1
etc...
VEC1(18) = 0
EXPORTCAD "C:\TEMP\MYCADFILE.IGS"
```

10.2.14.35. EXPORTJPG

Exports any graphic window as a JPG file.

Syntax:

```
EXPORTJPG winnum, filename, timeout
```

Discussion:

See " EXPORTBMP ".

Example:

```
EXPORTJPG 3, "C:\TEMP\MYJPGFILE"
EXPORTJPG k, FILENAME$, 500
```

10.2.14.36. FINDFILE

Used to find the names of files.

Syntax:

```
FINDFILE TEMPNAME$, FILTER$
```

Discussion:

This keyword requires two expressions, one to specify the string variable name to store the file name in, and another string variable which contains a "filter" string. The filter string usually specifies a path name and wildcards appropriate to the desired file type. See the example below.

FINDFILE is useful for listing all files of a certain type in a folder, or for analyzing large numbers of similar lens files. To reset FINDFILE back to the first file of any type, just call FINDFILE with a different filter, then call FINDFILE again with the original filter name. Each time FINDFILE is called with a new filter, it resets back to the first file that meets the filter specifications.

Example:

```
FILTER$ = "C:\Zemax\*.ZMX"
PRINT "Listing of all Zemax files in ", FILTER$
FINDFILE TEMPFILE$, FILTER$
LABEL 1
if (SLEN(TEMPFILE$))
PRINT TEMPFILE$
FINDFILE TEMPFILE$, FILTER$
GOTO 1
ENDIF
PRINT "No more files."
```

10.2.14.37. FLDX, FLDY, FWGT, FVDX, FVDY, FVCX, FVCY, FVAN

These commands are obsolete. See "SETSYSTEMPROPERTY, SYSP".

10.2.14.38. FOR, NEXT

The FOR and NEXT keywords define a program block which is executed a specific number of times in a loop.

Syntax:

```
FOR variable, start_value, stop_value, increment commands...
NEXT
```

Discussion:

The keyword FOR marks the beginning of a group of commands to be executed a multiple number of times. FOR requires a variable to be specified which acts as a counter (it need not be an integer), a starting value for the counter, a stop value, and an increment. The NEXT keyword marks the end of the group of commands. FOR- NEXT loops may be nested. The number of FOR and NEXT commands must be the same.

Upon reaching a FOR command, the expressions for the start, stop, and increment values are evaluated and saved. The stop and increment values are not evaluated again, even if the expressions defining the values consist of variables whose values change within the program block. Only the values valid at the beginning of the FOR loop are used.

If the start value and stop value are the same, the loop executes exactly once.

If the start value is less than the stop value, then the loop continues until the counter variable is greater than the stop value.

If the start value is greater than stop value, then the loop continues until the counter variable is less than the stop value.

Note that only integral values can be reliably represented exactly using computer floating point math, so comparing two floating point values for equality is not advised. Rather than using floating point ranges in loops, it is advised that the loop is conducted over an integer range and then converted from the index to the actual value of interest. More information on the accuracy of floating point arithmetic can be found in this helpful Wikipedia article: https://en.wikipedia.org/wiki/Floating-point_arithmetic#Accuracy_problems

Example:

```
FOR a, 0.2, 2, 0.2
   PRINT a
NEXT

j = 5
k = 0

FOR i, j, j + 5, 2
   k = i + j + k
NEXT
```

10.2.14.39. FORMAT

Specifies the numerical precision format for subsequent PRINT and $STR commands.

Syntax:

```
FORMAT m.n
FORMAT m.n EXP
FORMAT m [INT]
FORMAT "C_format_string" [LIT]
```

Discussion:

The integers m and n are separated by a decimal point. The values for m and n used must be explicit, i.e. values stored as variables cannot be used. The value m refers to the total number of characters to be printed, even though some of them may be blank. The value n refers to the number of places to display after the decimal point. Therefore, FORMAT 8.4 will cause subsequent PRINT commands to print 8 characters, with 4 numbers after the decimal point. FORMAT .5 will cause PRINT to show 5 decimal places, and as many total places as needed. FORMAT only affects numeric output from PRINT. If a number is too large to fit within the m decimal places, then the m portion of the FORMAT command will be ignored.

The optional keyword EXP after the m.n expression indicates exponential notation should be used.

The optional keyword INT indicates the value should be first converted to an integer and printed in integer format using the number of places specified by m.

The optional keyword LIT (for literal) indicates the value should be printed according to the "C" language format specifier. The C format specification can be found in any programming reference for the C language.

Example:

The macro:

```
X = 5
FORMAT 5.3
PRINT "FORMAT 5.3         :", X
FORMAT 12.2 EXP
PRINT "FORMAT 12.2 EXP    :", X
FORMAT 6 INT
PRINT "FORMAT 6 INT       :", X
FORMAT "%#06i" LIT
PRINT "FORMAT %#06i LIT   :", X
```

will produce this output:

FORMAT 5.3 :5.000

FORMAT 12.2 EXP : 5.00E+000

FORMAT 6 INT : 5

FORMAT %#06i LIT :000005

10.2.14.40. FTYP

For setting the field type use the SETSYSTEMPROPERTY, SYSP keyword, and for getting the data use the the SYPR(code) Numeric Function as discussed under "Numeric Functions.

10.2.14.41. GCRS

For setting the system property use the SETSYSTEMPROPERTY, SYSP keyword, and for getting the data use the the SYPR(code) Numeric Function as discussed under Numeric Functions .

10.2.14.42. GDATE

GDATE will place the current date under the lens name in the text box on user defined graphics screens.

Syntax:

```
GDATE
```

Discussion:

GDATE is primarily used for making graphics generated by ZPL macros look like other OpticStudio graphics.

Example:

See " GRAPHICS ".

10.2.14.43. GETDENCUSER1D

Calculates diffraction encircled energy data from a Huygens PSF text file and places the data in one of the vector arrays (either VEC1, VEC2, VEC3, or VEC4).

Syntax:

```
GETDENCUSER1D wave, pupil_samp, PSFFile$, vector
```

Discussion:

Wave is an integer corresponding to the wavelength number to use for the calculation. A value of zero indicates a polychromatic calculation. Pupil_samp may be 1 (32 x 32), 2 (64 x 64), 3 (128 x 128), etc... up to 2048 x 2048. PSFFile$

is a string containing the file name and path of the text file containing the Huygens PSF data to use. The vector argument must be an integer value between 1 and 4, and specifies which vector array the data should be placed in. If any of the arguments fall outside the valid ranges, then the nearest acceptable value is used instead. This calculation uses the Huygens MTF method.

*Note that TXT File Encoding (set under OpticStudio Preferences > General) must be set to ANSI to use this keyword.

Example:

```
PSFFile$ = "C:\Users\Name\Documents\Zemax\PSF.txt"
PSFFile$ = "C:\Users\Name\Documents\Zemax\PSF.txt"

SettingsFile$ = "C:\Users\Name\Documents\Zemax\Configs\HPS.CFG"

wave = 3
pupil_samp = 4
image_samp = 3

MODIFYSETTINGS SettingsFile$, HPS_PUPILSAMP, pupil_samp
MODIFYSETTINGS SettingsFile$, HPS_IMAGESAMP, image_samp
MODIFYSETTINGS SettingsFile$, HPS_WAVE, wave
MODIFYSETTINGS SettingsFile$, HPS_FIELD, 1
MODIFYSETTINGS SettingsFile$, HPS_CENTROID, 0

GETTEXTFILE PSFFile$, Hps

vector = 4

GETDENCUSER1D wave, pupil_samp, PSFFile$, vector
N_BINS = vec4(0)
OFFSET = vec4(1)

OUTPUT SCREEN

FORMAT 15.0
PRINT "Number of Bins = ", N_BINS
FORMAT 15.0
PRINT "Offset = ", OFFSET

OFF1 = OFFSET
OFF2 = OFF1 + N_BINS

MAXI = N_BINS-1
FORMAT 16.6
PRINT
PRINT " Radial Distance Energy"
PRINT

FOR i, 0, MAXI, 1
PRINT vec4(OFF1 + i),
PRINT vec4(OFF2 + i)
NEXT i
```

10.2.14.44. GETEXTRADATA

Retrieves the extra data values loaded into the lens data editor. The data is placed in one of the vector array variables (either VEC1, VEC2, VEC3, or VEC4).

Syntax:

```
GETEXTRADATA vector_expression, surface_expression
```

Discussion:

The data is stored in the specified VECn array variable. For example, if the command GETEXTRADATA 1, 5 is issued, the extra data for surface 5 will be placed in VEC1. The data is stored in the following format, where the first number in each line refers to the array position:

```
0:    The number of extra data values in the vector
1:    The first extra data value
n:    The nth extra data value
```

See "Extra data" for descriptions of the extra data values.

10.2.14.45. GETGLASSDATA

Retrieves the data for any glass in the current catalogs. The data is placed in one of the vector array variables (either VEC1, VEC2, VEC3, or VEC4).

Syntax:

```
GETGLASSDATA vector_expression, glass_number
```

Discussion:

The data is stored in the specified VECn array variable. For example, if the command GETGLASSDATA 1, 32 is issued, the data for glass number 32 will be placed in VEC1. The glass number is returned by the GNUM function. The data is stored in the following format, where the first number in each line refers to the array position:

0: The number of data values in the vector

1: Formula number: The number indicates the formula as follows:

1 = Schott, 2 = Sellmeier 1, 3 = Herzberger, 4 = Sellmeier 2

5 = Conrady,6 = Sellmeier 3, 7 = Handbook of Optics 1, 8 = Handbook of Optics 2

9 = Sellmeier 4,10 = Extended, 11 = Sellmeier 5, 12 = Extended 2

2: Reference temperature in degrees c

3: Nd

4: Vd

5: Thermal coefficient of expansion -30 to +70 c

6: Thermal coefficient of expansion +20 to 300 c

7: Density in g/cm^3

8: Deviation from normal line P gf

9: Lambda min

10: Lambda max

11-16:Constants of dispersion A0-A5 (meaning depends upon formula)

17-22:Thermal constants of dispersion

23-(22 + #waves): Internal transmission coefficient (per mm) alpha, T = exp(- alpha * path). The alpha for wavelength 1 is stored in 23, wavelength 2 is in 24, etc., up to the number of wavelengths used by the system.

(23 + #waves) - (32 + #waves): Constants of dispersion A0-A9 (meaning depends upon formula)

Related Functions:

GNUM

10.2.14.46. GETLSF

Calculates the geometric edge and line response functions, similar to the "Geometric Line/Edge Spread".

Syntax:

```
GETLSF wave, field, sampling, vector, maxradius, use_polarization
```

Discussion:

Wave is an integer corresponding to the wavelength number to use for the calculation. A value of zero indicates a polychromatic calculation. Field must be an integer between 1 and the maximum number of fields. The value indicates which field position to use. Sampling may be 1 (32 x 32), 2 (64 x 64), 3 (128 x 128), etc... up to 2048 x 2048. The vector argument must be an integer value between 1 and 4, and specifies which vector array the data should be placed in. The maxradius argument is the maximum radial coordinate of the edge and line spread functions; this is the half-width of the data range. Use 0 for a default width. If any of the arguments fall outside the valid ranges, then the nearest acceptable value is used instead.

The data is returned as an array of values in the specified vector. Vector position 0-3 will hold the number of points "N", the starting x coordinate (this is the negative of the half width of the data range), the delta coordinate, and the offset (defined below), respectively. The offset is the first position in the vector that holds the edge or line spread data. Starting at the offset, the first N value are the tangential LSF response. The next N values are the sagittal LSF response. The tangential and sagittal ERF values are in the next two groups of N data values.

If the current vector size is not large enough, OpticStudio will automatically increase the size of the vectors to hold the LSF data in the manner described in SETVECSIZE.

Example:

```
! Macro computes and prints the LSF and ERF for polychromatic light at field 1.
!
! Syntax is GETLSF wave, field, samp, vector, maxradius, usepol
!
GETLSF 0, 1, 3, 1, 0, 0

N_BINS = vec1(0)
STARTX = vec1(1)
DELTAX = vec1(2)
OFFSET = vec1(3)

FORMAT 15.0
PRINT "Number of Bins     = ", N_BINS
FORMAT 15.3 EXP
PRINT "Starting Coordinate = ", STARTX
PRINT "Delta Coordinate   = ", DELTAX
FORMAT 15.0
PRINT "Offset             = ", OFFSET

OFF1 = OFFSET
OFF2 = OFF1 + N_BINS
OFF3 = OFF2 + N_BINS
OFF4 = OFF3 + N_BINS
MAXI = N_BINS-1
FORMAT 16.3 EXP
PRINT
PRINT "             X          TLSF          SLSF          TERF          SERF"
PRINT

FOR i, 0, MAXI, 1
PRINT STARTX + DELTAX*i,
PRINT vec1(OFF1 + i),
```

```
PRINT vec1(OFF2 + i),
PRINT vec1(OFF3 + i),
PRINT vec1(OFF4 + i)

NEXT i
```

10.2.14.47. GETMTF

Calculates tangential and sagittal MTF, real part, imaginary part, phase, or square wave response data for the currently loaded lens file, and places the data in one of the vector arrays (either VEC1, VEC2, VEC3, or VEC4).

Syntax:

```
GETMTF freq, wave, field, sampling, vector, type
```

Discussion:

- The freq argument is the desired spatial frequency in MTF Units (see "MTF Units"). If the frequency is less than zero, or greater than the cutoff frequency, GETMTF returns zero.
- Wave is an integer corresponding to the wavelength number to use for the calculation. A value of zero indicates a polychromatic calculation.
- Field must be an integer between 1 and the maximum number of fields. The value indicates which field position to use.
- Sampling may be 1 (32 x 32), 2 (64 x 64), 3 (128 x 128), etc... up to 2048 x 2048.
- The vector argument must be an integer value between 1 and 4, and specifies which vector array the data should be placed in.
- The type argument refers to the data type:
- 1 for MTF,
- 2 for real part,
- 3 for imaginary part,
- 4 for phase in radians,
- 5 for square wave MTF.

If any of the arguments fall outside the valid ranges, then the nearest acceptable value is used instead.

This calculation uses a fast, sparse sampling integration method to compute the MTF, i.e. at the given point the MTF value is calculated as the autocorrelation of the complex pupil function (the complex wavefront in the exit pupil). The fast sampling method used by GETMTF is not directly related to the MTF Analysis feature. Because only a single spatial frequency is required, the method of computation used by GETMTF is different, and generally much faster, than the algorithm used by the analysis feature (see also MTFA operand Grid parameter).

The data is returned in one of the vector arrays with the following format:

- Vector position 0: tangential response;
- Vector position 1: sagittal response.

Example:

```
! This macro computes the T & S response at 30 lp/mm
! for the currently loaded lens, polychromatic,
! at the maximum defined field,
! and a 32x32 grid density (sampling = 1).
! Data will be placed in vector 1.
! This is all it takes to get the data:
GETMTF 30, 0, NFLD(), 1, 1, 1
PRINT "Tangential response:", vec1(0)
PRINT "Sagittal response :", vec1(1)
```

10.2.14.48. GETMTFUSER1D

Calculates tangential and sagittal MTF data from a Huygens PSF text file and places the data in one of the vector arrays (either VEC1, VEC2, VEC3, or VEC4).

Syntax:

```
GETMTFUSER1D wave, pupil_samp, idelta, PSFFile$, vector
```

Discussion:

Wave is an integer corresponding to the wavelength number to use for the calculation. A value of zero indicates a polychromatic calculation. Pupil_samp may be 1 (32 x 32), 2 (64 x 64), 3 (128 x 128), etc... up to 2048 x 2048. Idelta is the distance in micrometers between points in the image grid. Use zero for the default grid spacing. PSFFile$ is a string containing the file name and path of the text file containing the Huygens PSF data to use. The vector argument must be an integer value between 1 and 4, and specifies which vector array the data should be placed in. If any of the arguments fall outside the valid ranges, then the nearest acceptable value is used instead. This calculation uses the Huygens MTF method.

Note that TXT File Encoding (set under OpticStudio Preferences > General) must be set to ANSI to use this keyword. To convert existing text files to the right format, use the keyword CONVERTFILEFORMAT.

Note that you should close any active Huygens PSF files created with the OUTPUT keyword by using OUTPUT SCREEN prior to calling GETMTFUSER1D.

Example:

```
PSFFile$ = "C:\Users\Name\Documents\Zemax\PSF.txt"

SettingsFile$ = "C:\Users\Name\Documents\Zemax\Configs\HPS.CFG"

wave = 3
pupil_samp = 4
idelta = 0.008366


MODIFYSETTINGS SettingsFile$, HPS_PUPILSAMP, pupil_samp
MODIFYSETTINGS SettingsFile$, HPS_IMAGEDELTA, idelta
MODIFYSETTINGS SettingsFile$, HPS_IMAGESAMP, 2
MODIFYSETTINGS SettingsFile$, HPS_WAVE, wave
MODIFYSETTINGS SettingsFile$, HPS_FIELD, 1
MODIFYSETTINGS SettingsFile$, HPS_CENTROID, 1

GETTEXTFILE PSFFile$, Hps

vector = 4

GETMTFUSER1D wave, pupil_samp, PSFFile$, vector
N_BINS = vec4(0)
OFFSET = vec4(1)

OUTPUT SCREEN

FORMAT 15.0
PRINT "Number of Bins = ", N_BINS
FORMAT 15.0
PRINT "Offset = ", OFFSET

OFF1 = OFFSET
OFF2 = OFF1 + N_BINS
```

```
OFF3 = OFF2 + N_BINS

MAXI = N_BINS-1
FORMAT 16.6
PRINT
PRINT "             X          Tan            Sag"
PRINT

FOR i, 0, MAXI, 1
PRINT vec4(OFF1 + i),
PRINT vec4(OFF2 + i),
PRINT vec4(OFF3 + i)
NEXT i
```

10.2.14.49. GETNSCMTF

Calculate the X and Y direction geometric MTF in the non-sequential mode based on the spot diagram on Detector Rectangular.

Syntax:

```
GETNSCMTF freq, surface, object
```

Discussion:

The freq argument is designed for spatial frequency in MTF Units (see "MTF Units"). If the frequency is less than zero, GETNSCMTF uses its absolute value. The Surface argument allows the feature to be used in a sequential and non-sequential mixed mode. For a pure non-sequential mode system, the Surface number should be always set to 1. The Object argument points to the detector on which the MTF is calculated. It should be set to a Detector Rectangle only. This calculation uses the Geometric MTF method, which does a Fourier transform of the Spot Diagram on the Detector Rectangle.

The data is returned in first vector arrays, vec1(), with the following format: Vector position 0: X direction; Vector position 1: Y direction. See VEC1, VEC2, VEC3, VEC4 for more information on how to use vector arrays.

Example:

```
! This macro computes the X & Y direction NSC MTF at 50 lp/mm
! at object 10 - Detector Rectangular

PRINT "Resetting detectors..."
y = NSDD(0, 0, 0, 0)
PRINT "Tracing rays..."
NSTR 1, 0, 0, 0, 0, 1, 0

PRINT "Calculating MTF X and Y..."
GETNSCMTF 50.0, 1, 10
PRINT "mtf X = " +  $STR(vec1(0))
PRINT "mtf Y = " +  $STR(vec1(1))
```

10.2.14.50. GETPSF

Calculates the diffraction point spread function (PSF) using the FFT algorithm and places the data in one of the vector arrays (either VEC1, VEC2, VEC3, or VEC4).

Syntax:

```
GETPSF wave, field, sampling, vector, unnormalized, phaseflag, imagedelta
```

Discussion:

- Wave is an integer corresponding to the wavelength number to use for the calculation. A value of zero indicates a polychromatic calculation.
- Field must be an integer between 1 and the maximum number of fields. The value indicates which field position to use.
- Sampling may be 1 (32 x 32), 2 (64 x 64), 3 (128 x 128), etc... up to 2048 x 2048.
- The vector argument must be an integer value between 1 and 4, and specifies which vector array the data should be placed in.
- The unnormalized flag is zero if the data should be normalized to a peak of 1.0, if the unnormalized value is 1, then the data is returned unnormalized.
- If phase flag is zero, the data returned is intensity.If phase flag is 1, then the phase in degrees is returned.If phase flag is 2, the data returned is the real part of the PSF.If phase flag is 3, the data returned is the imaginary part of the PSF.
- The imagedelta value is the spacing between PSF points in micrometers; use zero for the default spacing. The wavelength must be monochromatic to compute phase data. If any of the arguments fall outside the valid ranges, then the nearest acceptable value is used instead.

The data is returned in one of the vector arrays with the following format:

Vector position 0: the total number of PSF data points in the vector array. Usually, this number will be 4*n*n where n is the sampling size (32, 64, etc.). For example, if the sampling density is 2, the pupil sampling will be 64 x 64, and there will be 128 x 128 or 16,384 values in the array. This will require 8 bytes per number, or a total of 131 kb. A sampling density of 1024 will require at least 8 Mb just for the array; another 64 Mb or more to compute the PSF. Position 0 also returns other values as error codes. If position 0 is zero, then the computation was aborted. If -1, then the vector array is not large enough to hold all the data. Use SETVECSIZE to make the array bigger. If -2, then there is not enough system RAM to compute the PSF data. If -3, a general error occurred while computing the PSF.

Vector position 1 through 4*n*n holds the PSF data intensity. The first 2n values are the first row, going left to right from -x to +x, then each subsequent block of 2n values is another row, going from -y to +y. Vector position 4*n*n+1 holds the spacing between data values in micrometers.

Example:

```
! This macro computes the PSF
! for the currently loaded lens, polychromatic,
! at the first field,
! and a 32x32 grid density (sampling = 1),
! data will be placed in vector 1,
! normalized to 1,
! no phase data,
! default image delta.

SETVECSIZE 4500
GETPSF 0, 1, 1, 1, 0, 0, 0
np = vec1(0)
IF (np == 0)
PRINT "PSF Computation aborted."
GOTO 1
ENDIF
IF (np == -1)
PRINT "SETVECSIZE too small for PSF data."
GOTO 1
ENDIF
IF (np == -2)
PRINT "Not enough system RAM for PSF data."
GOTO 1
ENDIF
PRINT "There are ", np, " data points, spaced ", vec1(np+1), " micrometers apart".
LABEL 1
```

10.2.14.51. GETSYSTEMDATA

Retrieves most system specific data, such as effective focal length, working F/#, apodization factors, and other data not associated with any particular surface. The data is placed in one of the vector array variables (either VEC1, VEC2, VEC3, or VEC4).

Syntax:

```
GETSYSTEMDATA vector_expression
```

Discussion:

The data is stored in the specified VECn array variable. For example, if the command GETSYSTEMDATA 1 is issued, the system data will be placed in VEC1. See also SETSYSTEMPROPERTY.

The data is stored in the following format, where the first number in each line refers to the array position:

0: The number of system data values in the vector

1: Aperture Value

2: Apodization Factor

3: Apodization Type (0:none, 1:gaussian, 2:tangent)

4: Adjust Index Data To Environment setting (1 if true, 0 if false)

5: Temperature in degrees c (valid only if Use Env Data true)

6: Pressure in ATM (valid only if Use Env Data true)

7: Effective Focal Length

8: Image Space F/#

9: Object Space Numerical Aperture

10: Working F/#

11: Entrance Pupil Diameter

12: Entrance Pupil Position

13: Exit Pupil Diameter

14: Exit Pupil Position

15: Paraxial Image Height

16: Paraxial Magnification

17: Angular Magnification

18: Total Track

19: Ray Aiming (0 for off, 1 for paraxial, and 2 for real)

20: X Pupil Shift

21: Y Pupil Shift

22: Z Pupil Shift

23: Stop Surface Number

24: Global Coordinate Reference Surface Number

25: Telecentric object space (0 for off, 1 for on)

26: The number of configurations

27: The number of multi-configuration operands

28: The number of merit function operands

29: The number of tolerance operands

30: Afocal image space (0 for off, 1 for on)

31: X Pupil Compress

32: Y Pupil Compress

33: Axial symmetry of the system (0 for axial, 1 for non-axial)

10.2.14.52. GETTEXTFILE (keywords)

Creates a text file from any OpticStudio analysis window that supports text.

Syntax:

```
GETTEXTFILE textfilename, type, settingsfilename, flag
```

Discussion:

The textfilename must be in quotes, or be a string variable name, and include the full path, name, and extension for the file to be created. The string function $TEMPFILENAME can be used to define a suitable temporary file name. Temporary files created with the $TEMPFILENAME() function are not automatically deleted. Use the DELETEFILE keyword to delete these temporary files.

The type argument is a 3 character string code that indicates the type of analysis to be performed. For the full list of string codes, see the "String Codes" section of the The Programming Tab. If no type is provided or recognized, a standard ray trace will be generated.

If a valid file name in quotes or a string variable name is used for the settingsfilename argument, OpticStudio will use or save the settings used to compute the text file, depending upon the value of the flag parameter.

If the flag value is 0, then the default settings will be used. If the lens file has it's own default settings, then those will be used; these are the settings stored in the "lensfilename.cfg" file. If no lens specific default settings exist, then the default settings for all OpticStudio files, stored in the file "OpticStudio.CFG" will be used, if any. If no previous settings have been saved for this or any other lens, then the default settings used are the "factory" defaults used by OpticStudio.

If the flag value is 1, then the settings provided in the settings file, if valid, will be used to generate the file. If the data in the settings file is in anyway invalid, then the default settings will be used to generate the file. The only valid settings files are those generated by OpticStudio, then renamed and saved to a new user defined file name. For example, on the Spot Diagram settings dialog, pressing "Save" will generate a "SPT.CFG" file with the saved settings. If this file is renamed to "MySPT.CFG", then the file, with a full path, may be used as the settingsfilename argument to GetTextFile.

No matter what the flag value is, if a valid file name is provided for the settingsfilename, the settings used will be written to the settings file, overwriting any data in the file. To modify the settings defined within an existing settings file, use MODIFYSETTINGS.

Only text, and not graphic files, are supported by GETTEXTFILE.

See also OPEN , CLOSE , READ , READNEXT , READSTRING , and MODIFYSETTINGS .

Example:

```
! Macro sample to generate cardinal points in a file.
! Get a temporary file name
A$ = $TEMPFILENAME()
! Compute the data and place in the temp file
```

```
GETTEXTFILE A$, Car
! Open the new file and print it out
OPEN A$
LABEL 1
READSTRING B$
IF (EOFF()) THEN GOTO 2
PRINT B$
GOTO 1
LABEL 2
DELETEFILE A$
CLOSE
```

10.2.14.53. GETVARDATA

Retrieves the current number, type, and value of all optimization variables. The data is placed in one of the vector array variables (either VEC1, VEC2, VEC3, or VEC4).

Syntax:

```
GETVARDATA vector
```

Discussion:

The data is stored in the specified VECn array variable. For example, if the command GETVARDATA 1 is issued, the data will be stored in VEC1. The data is stored in the following format, where the first number in each line refers to the array position:

0: n, the number of variables

1: The type code for the first variable

2: Surface number for the first variable

3: Parameter number for the first variable

4: Object number for the first variable

5: The value of the first variable

5*q-4: The type code for the qth variable

5*q-3: Surface number for the qth variable

5*q-2: Parameter number for the qth variable

5*q-1: Object number for the qth variable

5*q: The value of the qth variable

etc...

The integer q goes from 1 to n, where n is the number of variables. If n is zero, then no valid data is returned. The value of the number in array position zero, n, is always valid. The type codes for variables is as described in the following table. The surface number, parameter number, and object number may or may not have meaning depending upon the type of variable.

GETVARDATA TYPE AND ID CODES

Variable type	Type Code	Surface	Parameter	Object
Curvature	1	surface #	-	-
Thickness	2	surface #	-	-
Conic	3	surface #	-	-

Index Nd	4	surface #	-	-
Abbe Vd	5	surface #	-	-
Partial Dispersion ΔP_{gF}	6	surface #	-	-
TCE	7	surface #	-	-
Parameter Values	8	surface #	parameter #	-
Extra Data Values	9	surface #	extra data #	-
Multi-configuration Operand Values	10	oper #	config #	-
Non-sequential Object Position X	11	surface #	-	object #
Non-sequential Object Position Y	12	surface #	-	object #
Non-sequential Object Position Z	13	surface #	-	object #
Non-sequential Object Tilt X	14	surface #	-	object #
Non-sequential Object Tilt Y	15	surface #	-	object #
Non-sequential Object Tilt Z	16	surface #	-	object #
Non-sequential Object Parameters	17	surface #	parameter #	object #

10.2.14.54. GETZERNIKE

Calculates Zernike Fringe, Standard, or Annular coefficients in units of waves for the currently loaded lens file, and places them in one of the vector arrays (either VEC1, VEC2, VEC3, or VEC4).

Syntax:

```
GETZERNIKE maxorder, wave, field, sampling, vector, zerntype, epsilon, reference
```

Discussion:

The maxorder argument is any number between 1 and 37 for Fringe or between 1 and 231 for Standard or Annular coefficients (see the discussion of zerntype below), and corresponds to the highest Zernike term desired. Wave and field are the integer values for the wavelength and field number respectively. The value for sampling determines the size of the grid used to fit the coefficients. Sampling may be 1 (32 x 32), 2 (64 x 64), etc.... up to 2048 x 2048. The vector argument must be an integer value between 1 and 4, and specifies which vector array the data should be placed in. The zerntype is 0 for "fringe" Zernike terms, 1 for "Standard" Zernike terms, and 2 for "Annular" Zernike terms. See "Zernike Fringe Coefficients", "Zernike Standard Coefficients", and "Zernike Annular Coefficients" for descriptions of these different types of Zernike terms. For Annular Zernike Coefficients epsilon is the annular ratio; this value is ignored for other Zernike types. To reference the OPD to the chief ray, the reference value should be zero or omitted; use 1 to reference to the surface vertex. If any of the arguments fall outside the valid ranges, then the nearest acceptable value is used instead.

The data is returned in one of the vector arrays with the following format: Vector position 1: Peak to valley to the chief ray in waves; Vector position 2: RMS to the zero OPD line in waves (this value is not physically meaningful but is provided for reference); Vector position 3: RMS to the chief ray in waves; Vector position 4: RMS to the image centroid in waves (this is the most physically meaningful number related to image quality); Vector position 5: Variance in waves; Vector position 6: Strehl ratio; Vector position 7: RMS fit error in waves; Vector position 8: Maximum fit error (at any one point) in waves. The remaining vector positions contain the actual Zernike coefficient data. For example, Zernike term number 1 is in vector position 9, Zernike term 2 is in position 10, and so on.

Example:

```
! This macro computes the first 37 Zernike Fringe coefficients
! for the currently loaded lens, at wave 1, field 1
! and a 32x32 grid density (sampling = 1). The coefficients
! will be placed in vector 1. First get the data:
GETZERNIKE 37,1,1,1,1,0
! Now print it out:
FORMAT 16.6
PRINT "Peak to Valley   : ", vec1(1)
PRINT "RMS to chief     : ", vec1(3)
PRINT "RMS to centroid  : ", vec1(4)
PRINT "Variance         : ", vec1(5)
PRINT "Strehl ratio     : ", vec1(6)
PRINT "RMS Fit Error    : ", vec1(7)
PRINT "Maximum Fit Error : ", vec1(8)
i = 1
label 1
FORMAT 2.0
PRINT "Zernike #", i, " = ",
FORMAT 16.6
PRINT vec1(8+i)
i = i + 1
if (i < 38) THEN GOTO 1
PRINT "All Done!"
```

10.2.14.55. GLAS

For setting glass properties use the " SETSURFACEPROPERTY, SURP " keyword, and for getting glass properties use the $GLASS(i) String Function as discussed under String Function.

10.2.14.56. GLASSTEMPLATE

GLASSTEMPLATE sets data on the Glass Substitution Template.

Syntax:

```
GLASSTEMPLATE code, data
```

Discussion:

For checkbox values, code is 0 for Use Glass Substitution Template, 1 for Exclude Glasses With Incomplete Data, 11 for Standard, 12 for Preferred, 13 for Obsolete, 14 for Special, and 21-26 for Use Relative Cost, CR, FR, SR, AR, and PR, respectively. The checkbox will be unchecked if the value of data is zero. Any non-zero value will check the checkbox.

For numerical values, code is 31-36 for Relative Cost, CR, FR, SR, AR, and PR, respectively.

Related Function:

GTEM

10.2.14.57. GLENSNAME

GLENSNAME will place the current lens name at the top left corner of the text box on user defined graphics screens.

Syntax:

```
GLENSNAME
```

Discussion:

GLENSNAME is primarily used for making your graphics look like other OpticStudio graphics.

Example:

See the section GRAPHICS .

10.2.14.58. GLOBALTOLOCAL (keywords)

Converts coordinate breaks from global to local references.

Syntax:

```
GLOBALTOLOCAL surf1, surf2, direction
```

Discussion:

GLOBALTOLOCAL concatenates all adjacent coordinate breaks into single coordinate breaks. Surf1 and surf2 define the surface range. Direction is 0 for forward, 1 for reverse coordinate breaks.

See " Global To Local ".

See also " LOCALTOGLOBAL ".

10.2.14.59. GOSUB, SUB, RETURN, and END

These four keywords are used together to define and call subprograms within the ZPL macro file. Each keyword has a special purpose. GOSUB is used to direct the program flow to a defined subroutine. SUB is used to define the subroutine name, as well as indicate the beginning of the subroutine body. RETURN indicates the macro execution should continue at the point where the most recent GOSUB call was placed. END indicates that the macro should terminate immediately.

Syntax:

See the Example section for sample syntax.

Discussion:

There can be no more than 100 defined subroutines per ZPL macro file. Each subroutine must be terminated by a RETURN command. More than one return command may be placed in the body of a subroutine. If subroutines are defined, at least one END command must be used to indicate the end of the main macro body. The main macro body must be at the top of the file.

There can be no more than 100 "nesting levels" used in the ZPL macro. For example, if subroutine ABC calls subroutine XYZ, then the nesting level is 2. If subroutine XYZ then calls subroutine DEF, the nesting level is 3.

All variables in ZPL are global. Any variables used or defined within a subroutine exist within the main macro as well.

Example:

```
x = 1 y = 2
GOSUB add
print "the sum of ", x, " and ", y, " is ", z
END
```

```
SUB add
z = x + y
RETURN
```

10.2.14.60. GOTO

Normally, each macro line is executed in turn. GOTO allows execution to resume at an arbitrary point in the macro. GOTO is always used in conjunction with the LABEL command.

Syntax:

```
GOTO label_number
GOTO text_label
```

Discussion:

There must be a LABEL command with the corresponding label_number or text_label somewhere in the macro, or an error will result.

Example:

```
LABEL 1
x = RAND(10)
if x <= 5 THEN GOTO 1
PRINT " X is greater than 5 "
```

10.2.14.61. GRAPHICS (keywords)

Creates a standard OpticStudio graphics frame with ruling lines for the plot title.

For an easy way of creating plots, see " PLOT " and " PLOT2D ".

Syntax:

```
GRAPHICS
GRAPHICS NOFRAME
GRAPHICS OFF
```

Discussion:

If GRAPHICS is specified alone, then a standard OpticStudio graphics window will be created. If the optional argument NOFRAME is supplied, then the standard frame for the graph title will be suppressed. All subsequent graphics commands will be sent to this newly created window. GRAPHICS OFF will close any existing open graphics windows, and then display the closed window.

Example:

```
Graphics

xmx = xmax()
xmn = xmin()
ymx = ymax()
ymn = ymin()

xwidth = xmx-xmn
ywidth = ymx-ymn

xleft = xmn + ( .1 * xwidth )
xrigh = xmn + ( .9 * xwidth )
ytopp = ymn + ( .1 * ywidth )
ybott = ymn + ( .7 * ywidth )
```

```
line xleft,ytopp,xrigh,ytopp
line xrigh,ytopp,xrigh,ybott
line xrigh,ybott,xleft,ybott
line xleft,ybott,xleft,ytopp

gtitle " the rain in spain falls mainly on the plain"
glensname
gdate
gtext xmx/2,ymx/2,0, " start this text in the center."
gtextcent ymx*.05, " center this text near the top."
gtext xmx*.05,ymx*.75,90, " place me vertically near left edge."
gtext xmx*.15,ymx*.68,30, " orient me at 30 degrees."
graphics off
```

Related Keywords:

LINE, GITITLE, GLENSNAME, GDATE, GTEXTCENT, GTEXT, PIXEL, PLOT

10.2.14.62. GTEXT

GTEXT is used for labeling graphics plots with user defined text.

Syntax:

```
GTEXT x, y, angle, user_text
GTEXT x, y, angle, A$
```

Discussion:

The coordinates x and y refer to the left edge of where the text string user_text will appear. "user_text" may be either a constant string in quotes or a string variable name. Angle specifies how the text is rotated with respect to the graphics frame, and defaults to 0 degrees (horizontal). See also SETTEXTSIZE .

Example:

See " GRAPHICS ".

10.2.14.63. GTEXTCENT

GTEXTCENT is used for centering labels on graphics plots with user defined text.

Syntax:

```
GTEXTCENT y, user_text
```

Discussion:

The coordinate y refers to the vertical position of the text string user_text. See also SETTEXTSIZE.

Example:

See " GRAPHICS ".

10.2.14.64. GTITLE

GTITLE is similar to GTEXT except only the text needs to be specified, and the text will appear centered in the title bar on the graphics display. GTITLE is useful for making ZPL graphics functions look like standard OpticStudio graphics.

Syntax:

```
GTITLE user_title
```

Example:

See " GRAPHICS ".

10.2.14.65. HAMMER (keywords)

Invokes the Hammer optimization algorithm to optimize the current lens with the current merit function.

Syntax:

```
HAMMER
HAMMER number_of_cycles
HAMMER number_of_cycles, algorithm
```

Discussion:

If no argument is provided, then the Hammer Optimization runs 1 cycle using Damped Least Squares. If an argument is provided, it must evaluate to an integer between 1 and 99, and the Hammer optimization algorithm will run the specified number of cycles. For the algorithm argument, use 0 for Damped Least Squares (the default) and 1 for Orthogonal Descent. For more information see "Performing an optimization".

Related Functions:

MFCN

Example:

```
PRINT "Starting merit function:", MFCN()
HAMMER 3
PRINT "Ending merit function :", MFCN()
```

10.2.14.66. IF-THEN-ELSE-ENDIF

IF provides conditional macro execution and branching.

Syntax:

```
IF (expression)
(commands)
ELSE (commands)
ENDIF
```

or

```
IF (expression) THEN (command)
```

Discussion:

The IF-ELSE-ENDIF construction is used for conditional execution of either the group of commands following the IF command or the commands following the ELSE command, but not both. The value of expression is considered false if it is zero, otherwise it is considered true. The expression can be any valid ZPL expression, composed of functions, variables, operands, and constants. The IF command must be paired with an ENDIF, although the ELSE is optional. IF-ENDIF pairs may be nested to any level.

The IF-THEN construction is handy for conditional execution of a single instruction. If the THEN keyword is specified, the IF command is terminated, and there is no need for an ENDIF. The ELSE keyword is not supported in the IF-THEN construction.

Example:

```
x = 1
y = 2
if (x < y)
PRINT "x is less than y"
ELSE
if (x == y) THEN PRINT "x equals y"
if (x > y) THEN PRINT "x is greater than y"
ENDIF
```

10.2.14.67. IMA

Computes the Geometric Image Analysis feature and saves the result to a BIM file. For a description of the BIM (Binary Image) format see "The BIM format". For a description of the Geometric Image Analysis feature see "Geometric Image Analysis".

Syntax:

```
IMA outfilename, infilename
```

Discussion:

This keyword requires the name of the output BIM file, and optionally, the name of the input IMA or BIM file. If the extension to the outfilename is not provided, the extension BIM will be appended. The extension must be provided on the infilename. The filenames must be enclosed in quotes if any blank or other special characters are used. The outfilename will be placed in the <data>\<images> folder. The infilename must also be placed in the <data>\<images> folder. No paths should be provided with the file names.

The settings for the Geometric Image Analysis feature will be those settings previously saved for the current lens. To make adjustments to the settings, open a Geometric Image Analysis window, choose the appropriate settings, then press "Save". Be certain to choose "Show As" as anything other than "spot diagram". All subsequent calls to IMA will use the saved settings. The exceptions are the output file name, which is specified as the first argument after the IMA keyword, and the input source file, which is optionally specified as the second argument after the IMA keyword.

Example:

```
IMA "output.BIM", "LETTERF.IMA"
```

Related Keywords:

IMASHOW, IMASUM

10.2.14.68. IMAGECOMBINE

Combines two graphic files and writes the combined graphic to a new file.

Syntax:

```
IMAGECOMBINE source1$, source2$, destination$, mode$
```

Discussion:

This keyword opens the two source graphic files, combines them to a single image, and then writes the resulting image to the destination file. The files may be in BMP, JPG or PNG format. Conversion between the formats is automatic and is based upon the extension provided. If the source images have the same number of rows, they can be combined left-to-right. If the source images have the same number of columns, they can be combined top-to-bottom. The combination used depends upon the mode$ value. Use either "LEFTRIGHT" or "TOPBOTTOM" for the value of mode$ to define how the images should be combined.

Example:

```
S1$ = "C:\TEMP\SOURCE1.JPG" S2$ = "C:\TEMP\SOURCE2.JPG"
D$ = "C:\TEMP\DESTINATION.JPG" M$ = "LEFTRIGHT"
IMAGECOMBINE S1$, S2$, D$, M$
```

Related Functions:

PIXX, PIXY

Related Keyword:

IMAGEEXTRACT

10.2.14.69. IMAGEEXTRACT

Extracts a rectangular region of a graphic file and writes the portion of the graphic to a new file.

Syntax:

```
IMAGEEXTRACT source$, destination$, startx, starty, width, height
```

Discussion:

This keyword opens the source graphic, and extracts a rectangular portion of the file, then writes the extracted portion to the destination file. The files may be in BMP, JPG or PNG format. Conversion between the formats is automatic and is based upon the extension provided. The startx and starty values are the offsets from the left side and top side respectively, in pixels. The top row and left column start at number 1. The width and height values are the size in pixels of the rectangular region to extract.

Example:

```
S$ = "C:\TEMP\SOURCE.JPG"
D$ = "C:\TEMP\DESTINATION.JPG"
IMAGEEXTRACT S$, D$, 1, 1, 20, 20
```

Related Functions:

PIXX, PIXY

Related Keywords:

IMAGECOMBINE

10.2.14.70. IMASHOW

Displays an IMA or BIM file in a viewer window. For a description of the IMA and BIM file formats, see "The IMA format" and "The BIM format".

Syntax:

```
IMASHOW filename.ima
```

Discussion:

This keyword requires the name of the IMA or BIM file. The extension must be included. The filename may be enclosed in quotes if any blank or other special characters are used. The file must be located in the <data>\<im- ages> folder. This command will open a new window to display the file.

Example:

```
IMASHOW "LETTERF.IMA"
```

Related Keywords:

IMA, IMASUM

10.2.14.71. IMASUM

Sums the intensity in a BIM file. Despite the name of this function, currently only BIM files, and not IMA files, may be summed. This limitation is due to the low number of grey scales that are supported in the IMA file format. For a description of the IMA and BIM file formats, see "The IMA format" and "The BIM format".

Syntax:

```
IMASUM filename1, filename2, outfilename
```

Discussion:

This keyword requires the names of three BIM files (which need not be distinct). The BIM extension must be provided on all three file names. The filenames may be enclosed in quotes if any blank or other special characters are used. The files must be located in the <data>\<images> folder, which is where the output file will be. The two source files must have the same number of pixels and color channels, or an error message will result.

Example:

```
IMASUM "A.BIM", "B.BIM", "sum of A and B.BIM"
```

Related Keywords:

IMA, IMASHOW

10.2.14.72. IMPORTEXTRADATA (keywords)

Imports data into the lens data editor from a file.

Syntax:

```
IMPORTEXTRADATA surface, filename
```

Discussion:

The surface may be any valid surface number that uses extra data values. The filename should include the full path. For the supported data formats,

see "Importing grid data" under the sequential surface " Grid Sag ".

10.2.14.73. INPUT

INPUT provides a means for prompting the user for numeric or text data when the macro is run.

Syntax:

```
INPUT "Prompt String" , variable
INPUT variable
INPUT "Prompt String" , string_variable$
INPUT string_variable$
```

Discussion:

The variable may be any valid variable name. If the variable name is a string variable, then the input will be interpreted as a literal string; otherwise, as a numeric. The INPUT command will use a "?" prompt if no prompt string is supplied.

The prompt is always displayed on the screen, and the input always is accepted from the keyboard only. The prompt string cannot exceed 200 characters in length.

Example:

```
INPUT "Enter value for x:", x
PRINT "X = ", x
INPUT "Enter a value for A$:", A$
PRINT A$
```

10.2.14.74. INSERT

INSERT inserts a new surface in the lens data editor.

Syntax:

```
INSERT surf
```

Discussion:

The value surf must evaluate to an integer expression. See also DELETE and SURFTYPE.

Example:

```
INSERT 5
INSERT i+2*j
```

10.2.14.75. INSERTCONFIG

INSERTCONFIG inserts a new configuration in the multi-configuration editor.

Syntax:

```
INSERTCONFIG config
```

Discussion:

The value config must evaluate to an integer expression greater than 0 and less than or equal to the current number of configurations plus 1. See also DELETECONFIG and INSERTMCO.

Example:

```
INSERTCONFIG 4
```

10.2.14.76. INSERTMCO (keywords)

INSERTMCO inserts a new multi-configuration operand in the multi-configuration editor.

Syntax:

```
INSERTMCO row
```

Discussion:

The value row must evaluate to an integer expression greater than 0 and less than or equal to the current number of operands plus 1. See also DELETEMCO and INSERTCONFIG.

Example:

```
INSERTMCO 7
```

10.2.14.77. INSERTMFO (keywords)

INSERTMFO inserts a new merit function operand in the merit function editor.

Syntax:

```
INSERTMFO row
```

Discussion:

The value row must evaluate to an integer expression greater than 0 and less than or equal to the current number of operands plus 1.

See also DELETEMFO and SETOPERAND.

Example:

```
INSERTMFO 23
```

10.2.14.78. INSERTOBJECT (keywords)

Inserts a new NSC object.

Syntax:

```
INSERTOBJECT surf, object
```

Discussion:

The value surf must evaluate to an integer expression, and the specified surface must be a non-sequential component surface. Use a surf value of 1 if the program mode is non-sequential. The value object must evaluate to an integer that is between 1 and the current number of objects plus 1, inclusive. The new object will be inserted at the specified object number, renumbering other objects as required.

See also DELETEOBJECT and SETNSCPROPERTY.

Example:

```
INSERTOBJECT 1, 1
```

10.2.14.79. INSERTTOL

INSERTTOL inserts a new operand in the tolerance data editor.

Syntax:

```
INSERTTOL row
```

Discussion:

The value row must evaluate to an integer expression greater than 0 and less than or equal to the current number of operands plus 1. See also DELETETOL and SETTOL.

Example:

```
INSERTTOL 3
```

10.2.14.80. LABEL

LABEL provides a destination for the LABEL command, see " GOTO " for details.

Syntax:

```
LABEL label_number
LABEL text_label
```

Discussion:

The label_number must be an integer value greater than zero, such as 1 or 7. If a text label is used, the text label must not contain spaces or other special characters that are used as delimiters. LABEL has no effect on program flow by itself. There is a limit of 300 label commands in a macro.

Example:

```
LABEL 7
LABEL startover
```

10.2.14.81. LINE (keywords)

LINE is the primitive line drawing function for graphical displays.

Syntax:

```
LINE oldx, oldy, newx, newy
```

Discussion:

LINE will evaluate the four expressions and draw a line connecting the points defined. The coordinates refer to the current graphics frame, and must be contained within the bounds defined by XMIN, YMIN, XMAX, and YMAX. Although only integer pixel values can actually be plotted, LINE will accept real values as arguments and round the coordinates to the nearest integer equivalents. LINE is only valid in graphics mode. The color of the line drawn is controlled by the current pen color, which is specified using the COLOR keyword.

Example:

See " GRAPHICS ".

10.2.14.82. LOADARCHIVE

Opens an existing Zemax archive (*.ZAR) file.

Syntax:

```
LOADARCHIVE filename, extractpath
```

Discussion:

The filename must include the name of the archive file including the file extension. If the filename does not include a complete path to the archive file, the default folder for lenses is assumed. If no extractpath is defined, the path is assumed to be the same as the filename. The lens and session files defined in the archive file are then opened. See "Restore From Archive File".

10.2.14.83. LOADCATALOG

Reloads glass and coating catalogs for the currently loaded lens.

Syntax:

```
LOADCATALOG
```

Discussion:

When lenses are loaded, any associated glass catalogs and data files, including the COATING.DAT file, are automatically loaded if they are not already loaded. However, if these catalogs have been modified, perhaps by the ZPL macro itself; then the LOADCATALOG keyword may be used to force a reload of the catalogs. Use of this keyword is not required unless the COATING.DAT or glass AGF catalog files have been modified since the start of the current Zemax session.

10.2.14.84. LOADDETECTOR (keywords)

Loads the data saved in a file to an NSC Detector Rectangle, Detector Color, Detector Polar, or Detector Volume object.

Syntax:

```
LOADDETECTOR surf, object, filename
```

Discussion:

This keyword requires numeric expressions that specify the surface and object, and a file name with or without a full path. Surf is the surface number of the non-sequential group; use 1 when using non-sequential mode. Object is the object number of the detector object. The filename may include the full path, if no path is provided the path of the current lens file is used. The extension should be DDR, DDC, DDP, or DDV for Detector Rectangle, Color, Polar, and Volume objects, respectively.

Related Keywords:

SAVEDETECTOR

10.2.14.85. LOADLENS

Loads a new lens file, replacing any lens file currently in memory.

Syntax:

```
LOADLENS filename, appendflag, session
```

Discussion:

LOADLENS will load a lens file. If the filename contains the complete path, such as C:\MYDIR\MYLENS.ZMX, then the specified file will be loaded. If the path is left off, then the default folder for lenses will be used (see " Folders ").

If the appendflag is zero or absent, then LOADLENS loads the file.

If the appendflag is greater than zero, then the file is appended to the current lens starting at the surface specified by the value of the appendflag. The object and image surfaces of the specified file will be ignored.

If the appendflag is less than zero, then the file is appended to the current lens starting at the surface specified by the absolute value of the appendflag, and the object surface of the specified file will be included. The image surface will be ignored.

The appendflag should only be used when appending one sequential system to another. Appending non-sequential systems isn't currently supported.

If the session flag is non-zero, any associated session file will be loaded with the lens and all windows will be updated otherwise the lens session file is ignored.

Example:

```
LOADLENS "COOKE.ZMX"
```

Related Keywords:

10.2.14.86. LOADMERIT (keywords)

Loads a merit function file, replacing the merit function in the current lens.

Syntax:

```
LOADMERIT "filename"
LOADMERIT file$
```

Discussion:

LOADMERIT will load a new merit function from a .MF file. If the filename contains the complete path, such as C:\MYDIR\MYLENS.MF, then the specified file will be loaded. If the path is left off, then the <data>\MeritFunc- tion folder will be used (see " Folders "). This keyword can also load just the merit function from a ZMX file using the same syntax.

See also SAVEMERIT .

10.2.14.87. LOADTOLERANCE (keywords)

Loads a tolerance file, replacing the tolerances in the current lens.

Syntax:

```
LOADTOLERANCE "filename"
LOADTOLERANCE file$
```

Discussion:

LOADTOLERANCE will load a new tolerance file. If the filename contains the complete path, such as C:\MYDIR\MYLENS.TOL, then the specified file will be loaded. If the path is left off, then the <data>\Tolerance folder will be used (see " Folders ").

See also SAVETOLERANCE .

10.2.14.88. LOCALTOGLOBAL (keywords)

Converts coordinate breaks from local to global references.

Syntax:

```
LOCALTOGLOBAL surf1, surf2, reference
```

Discussion:

LOCALTOGLOBAL modifies all coordinate breaks into groups of 3 coordinate breaks, providing a means to locate surfaces in global coordinates. Surf1 and surf2 define the surface range. Reference is the reference surface number, which must precede s1. See " Local To Global ". See also " GLOBALTOLOCAL ".

10.2.14.89. LOCKWINDOW

Locks any one or all open windows.

Syntax:

```
LOCKWINDOW winnum
```

Discussion:

See "Graphic windows operations". If the winnum is zero, then all open windows are locked. If the winnum argument is -1, then the currently executing window will lock at the end of the macro execution.

Example:

```
LOCKWINDOW 7
```

Related Keywords:

UNLOCKWINDOW

10.2.14.90. MAKEFACETLIST

Makes an ASCI listing of the vertices of a faceted description of any importable CAD file.

Syntax:

```
MAKEFACETLIST infile, outfile
```

Discussion:

The infile is a string variable or name with the full path to a CAD format file. The outfile is a string variable or name to the full path for the output file. The output file will be a text listing. There are 18 lines for each triangle in the faceted description of the object. The first 6 lines are data for vertex 1 of the first triangle in this format:

x coordinate

y coordinate

z coordinate

x normal vector

y normal vector

z normal vector

The next 6 values are for vertex 2, and then 6 more values for vertex 3. The 18 values together define the first triangle. The pattern repeats for the next triangle; each represented by 18 values, until all the triangles are written.

10.2.14.91. MAKEFOLDER

Makes a folder for files.

Syntax:

```
MAKEFOLDER "C:\TEMP\TEST_FOLDER"
MAKEFOLDER path$
```

Discussion:

The new folder will be created. If the folder already exists the keyword is ignored. If the path is invalid, an error message is issued and the macro execution will be terminated.

10.2.14.92. MODIFYSETTINGS (keywords)

Modifies data within the settings files used by GETTEXTFILE.

The settings file must be manually created prior to using MODIFYSETTING. To create a setting file, open the analysis in OpticStudio and click Save in the settings dropdown.

Syntax:

```
MODIFYSETTINGS settingsfilename, type, value, windowtype
```

Discussion:

The settingsfilename must be in quotes, or be a string variable name, and include the full path, name, and extension for the file to be modified.

The type argument is a text mnemonic that indicates which setting within the file is to be modified. The supported values for the type argument are listed in the table below.

The windowtype argument is a 3 character string code that indicates the type of analysis to look for in a lens-specific settings file. For the full list of string codes, see the "String Codes" section of the The Programming Tab. If windowtype is omitted, the first settings that match the type argument will be modified; note that some type arguments may match multiple different windows settings and, if no settings match the given type, no change will be made to the file.

Type codes supported by MODIFYSETTINGS

Feature	**Available type codes**
2D Layout	LAY_RAYS: The number of rays.
Detector Viewer	DVW_SURFACE: The surface number. Use 1 for Non-sequential mode. DVW_DETECTOR: The detector number. DVW_SHOW: The "Show As" setting. The meaning depends upon the type of window displayed (Graphic or Text) and the type of detector (Detector Rectangle, Detector Color, etc.). The value for DVW_SHOW corresponds to the position of the desired item in the UI of the Detector Viewer window. For an example, see Table 2: DVW_SHOW Example on page 1580. DVW_ROWCOL: The row or column number for cross section plots. DVW_ZPLANE: The Z-Plane number for detector volumes. DVW_SCALE: The scale mode. Use 0 for linear, 1 for Log -5, 2 for Log -10, and 3 for Log - 15. DVW_SMOOTHING: The integer smoothing value. DVW_DATA: The "Show Data" setting. Like DVW_SHOW, the meaning depends on other settings such as type of window displayed, type of detector, and source units. The value corresponds to the position of the desired item in the UI of the Detector viewer window. For an example, see Table 3: DVW_DATA Example on page 1581 DVW_ZRD: The ray data base name, or null for none. DVW_FILTER: The filter string. DVW_MAXPLOT: The maximum plot scale. DVW_MINPLOT: The minimum plot scale. DVW_OUTPUTFILE: The output file name.

Extended Diffraction Image Analysis	EXD_DISPLAYSIZE: The display size. EXD_FIELD: The field number. EXD_FILESIZE: The file size. EXD_WAVE: The wavelength number.
Extended Source Encircled Energy	XSE_FIELD: Field number (1, .., n) XSE_FIELDSIZE: Field Size XSE_WAVE: Wavelength (0 = "All", 1, ...n) XSE_KRAYS: Rays x 1000 XSE_IMANAME: *.IMA File name XSE_SURFACE: Surface number XSE_MAXRAD: Maximum Radial Distance XSE_TYPE: Type of the Data (1 = "Encircled", 2 = "X Only", 3 = "Y Only", etc.) XSE_REF: Reference to (0 = "Chief Ray", 1 = "Centroid", 2 = "Vertex") XSE_POLARIZATION: Use Polarization flag (0 for off, 1 for on) XSE_REMVIGNET: Remove Vignetting Factors flag XSE_MULDIFFLI: Multiply by Diffraction Limit flag
FFT Line/Edge Spread	LSF_COHERENT: Use 0 for incoherent, 1 for coherent LSF_TYPE: Use 0-9 for X-Linear, Y-Linear, X-Log, Y-Log, X-Phase, Y-Phase, X- Real, Y-Real, X-Imaginary, or Y-Imaginary, respectively. LSF_SAMP: The sampling, use 1 for 32 x 32, 2 for 64 x 64, etc. LSF_SPREAD: Use 0 for line, 1 for edge. LSF_WAVE: The wavelength number, use 0 for polychromatic (incoherent only). LSF_FIELD: The field number. LSF_POLARIZATION: Use 0 for unpolarized, 1 for polarized. LSF_PLOTSCALE: The plot scale.
FFT PSF	PSF_TYPE: Use 0-4 for Linear, Log, Phase, Real, or Imaginary, respectively. PSF_SAMP: The sampling, use 1 for 32 x 32, 2 for 64 x 64, etc. PSF_WAVE: The wavelength number, use 0 for polychromatic. PSF_FIELD: The field number. PSF_SURFACE: The surface number, use 0 for image. PSF_POLARIZATION: Use 0 for unpolarized, 1 for polarized. PSF_NORMALIZE: Use 0 for unnormalized, 1 for unity normalization. PSF_IMAGEDELTA: The image point spacing in micrometers.

FFT PSF Cross Section	PSF_TYPE: Use 0-9 for X-Linear, Y-Linear, X-Log, Y-Log, X-Phase, Y-Phase, X- Real, Y-Real, X-Imaginary, or Y-Imaginary, respectively. PSF_ROW: The row number (if doing an X scan) or column number (if doing a y scan). Use 0 for center. PSF_SAMP: The sampling, use 1 for 32 x 32, 2 for 64 x 64, etc. PSF_WAVE: The wavelength number, use 0 for polychromatic. PSF_FIELD: The field number. PSF_POLARIZATION: Use 0 for unpolarized, 1 for polarized. PSF_NORMALIZE: Use 0 for unnormalized, 1 for unity normalization. PSF_PLOTSCALE: The plot scale.
Footprint Diagram	FOO_RAYDENSITY: The ray density. Use 0 for ring, 1 for 10, 2 for 15, 3 for 20 etc. FOO_SURFACE: The surface number. FOO_FIELD: The field number. FOO_WAVELENGTH: The wavelength number. FOO_DELETEVIGNETTED: Delete vignetted, use 0 for no, 1 for yes.
Full-Field Aberration	FFA_FIELD: The field number. FFA_FIELDSHAPE: The field shape, use 0 for Rectangular and 1 for Elliptical. FFA_WAVELENGTH: The wavelength number. FFA_XFIELDWIDTH: The half size of the field sampling grid in X. FFA_YFIELDWIDTH: The half size of the field sampling grid in Y. FFA_XFIELDSAMPLING: The number of sampling field points in X, use 0 for 5, 1 for 6, etc. FFA_YFIELDSAMPLING: The number of sampling field points in Y, use 0 for 5, 1 for 6, etc. FFA_DECOMPOSITION: The decomposition option, use 0 for Zernike Terms. FFA_MAXIMUMTERM: The maximum term to be considered for the decomposition. Any value up to 231 may be specified. FFA_ABERRATION: The aberration to be plotted, use 0 for Defocus, 1 for Primary Astigmatism, etc. FFA_PUPILSAMPLING: The pupil density to use for coefficient fitting, use 1 for 32 x 32, 2 for 64 x 64, etc. FFA_DISPLAY: The display mode, use 0 for Absolute, 1 for Relative, and 2 for Average. FFA_SHOWAS: The "Show As" setting, use 0 for Grey Scale, 1 for Inverse Grey Scale, etc.

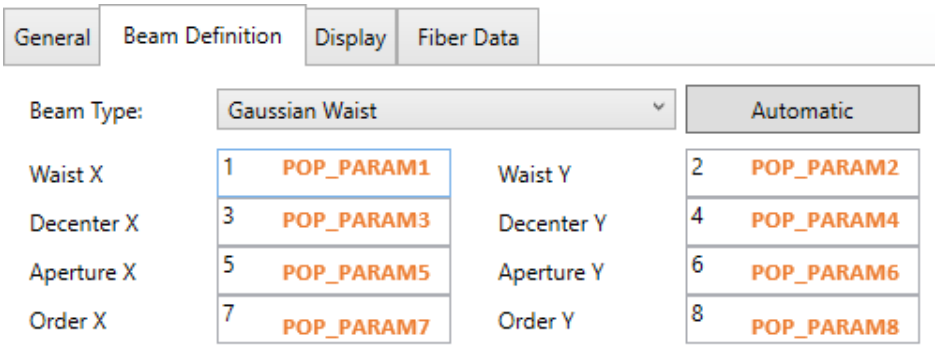
Geometric Bitmap Image Analysis	GBM_FIELDSIZE: The field Y size. GBM_RAYS: The number of rays per source pixel. GBM_XPIX: The number of X pixels. GBM_YPIX: The number of Y pixels. GBM_XSIZ: The X pixel size. GBM_YSIZ: The Y pixel size. GBM_INPUT: The input file name GBM_OUTPUT: The output file name GBM_SURFACE: The surface number GBM_ROTATION: The rotation setting
Geometric Image Analysis	IMA_FIELD: The field size. IMA_IMAGESIZE: The image size. IMA_IMANAME: The image file name. IMA_KRAYS: The number of rays x 1000. IMA_NA: The numerical aperture. IMA_OUTNAME: The output file name. IMA_SURFACE: The surface number. IMA_PIXELS: The number of pixels.
FFT Through Focus MTF	TFM_SAMP: The sampling. Use 1 for 32x32, 2 for 64x64, etc. TFM_DELTAFOC: The delta focus. TFM_FREQ: The spatial frequency for which the data is plotted. TFM_STEPS: The number of focal plane steps. TFM_WAVE: The wavelength number. Use 0 for all. TFM_FIELD: The field number. Use 0 for all. TFM_TYPE: The data type. Use 0 for modulation, 1 for real, 2 for imaginary, 3 for phase, or 4 for square wave. TFM_POLAR: Use polarization. Use 0 for no, 1 for yes. TFM_DASH: Use dashes. Use 0 for no, 1 for yes.

Huygens MTF	HMF_PUPILSAMP: The pupil sampling. Use 1 for 32x32, 2 for 64x64, etc. HMF_IMAGESAMP: The image sampling. Use 1 for 32x32, 2 for 64x64, etc. HMF_IMAGEDELTA: The image point spacing in micrometers. HMF_CONFIG: The configuration number. Use 0 for all, 1 for current, etc. HMF_WAVE: The wavelength number. Use 0 for polychromatic. HMF_FIELD: The field number. Use 0 for all. HMF_TYPE: The data type. Currently only modulation (0) is supported. HMF_MAXF: The maximum spatial frequency. HMF_POLAR: Use polarization. Use 0 for no, 1 for yes. HMF_DASH: Use dashes. Use 0 for no, 1 for yes.
Huygens Through Focus MTF	HTF_PUPILSAMP: The pupil sampling. Use 1 for 32x32, 2 for 64x64, etc. HTF_IMAGESAMP: The image sampling. Use 1 for 32x32, 2 for 64x64, etc. HTF_IMAGEDELTA: The image point spacing in micrometers. HTF_CONFIG: The configuration number. Use 0 for all, 1 for current, etc. HTF_FREQ: The spatial frequency for which data is plotted. HTF_WAVE: The wavelength number. Use 0 for all. HTF_FIELD: The field number. Use 0 for all. HTF_TYPE: The data type. Currently only modulation (0) is supported. HTF_DELTAFOC: The delta focus. HTF_STEPS: The number of focal plane steps. HTF_POLAR: Use polarization. Use 0 for no, 1 for yes. HTF_DASH: Use dashes. Use 0 for no, 1 for yes.
Huygens MTF vs. Field	HMH_SAMP: The sampling. Use 1 for 32x32, 2 for 64x64, etc. HMH_SCANTYPE: The field scan type. Use 0 for +Y, 1 for +X, etc. HMH_WAVE: The wavelength. Use 0 for all. HMH_FIELDDENSITY: The field density. HMH_FREQ1: Spatial frequency 1. HMH_FREQ2: Spatial frequency 2. HMH_FREQ3: Spatial frequency 3. HMH_FREQ4: Spatial frequency 4. HMH_FREQ5: Spatial frequency 5. HMH_FREQ6: Spatial frequency 6. HMH_POLAR: Use polarization. Use 0 for no, 1 for yes. HMH_DASH: Use dashes. Use 0 for no, 1 for yes. HMH_REMOVEVIGNETTING: Remove vignetting factors. Use 0 for no, 1 for yes.

Huygens PSF	HPS_CENTROID: Use Centroid flag (0 for off, 1 for on). HPS_PUPILSAMP: The pupil sampling, use 1 for 32 x 32, 2 for 64 x 64, etc. HPS_IMAGESAMP: The image sampling, use 1 for 32 x 32, 2 for 64 x 64, etc. HPS_WAVE: The wavelength number, use 0 for polychromatic. HPS_FIELD: The field number. HPS_IMAGEDELTA: The image point spacing in micrometers. HPS_TYPE: The data type. Use 0-8 for Linear, Log -1, Log -2, Log -3, Log -4, Log -5, Real, Imaginary, or Phase, respectively.
Huygens PSF Cross Section	HPC_PUPILSAMP: The pupil sampling, use 1 for 32 x 32, 2 for 64 x 64, etc. HPC_IMAGESAMP: The image sampling, use 1 for 32 x 32, 2 for 64 x 64, etc. HPC_WAVE: The wavelength number, use 0 for polychromatic. HPC_FIELD: The field number. HPC_IMAGEDELTA: The image point spacing in micrometers. HPC_TYPE: The data type. Use 0-9 for X-Linear, Y-Linear, X-Log, Y-Log, X-Real, Y-Real, X-Imaginary, Y-Imaginary, X-Phase, or Y-Phase, respectively.
Illumination XY Scan	ILL_SOURCE: The source size. ILL_SMOOTH: The smoothing value to use. ILL_DETSIZE: The detector size. ILL_SURFACE: The surface number.

Image Simulation	ISM_INPUTFILE: The input file name. This should be specified without a path. ISM_FIELDHEIGHT: The Y field height. ISM_OVERSAMPLING: Oversample value. Use 0 for none, 1 for 2X, 2 for 4x, etc. ISM_GUARDBAND: Guard band value. Use 0 for none, 1 for 2X, 2 for 4x, etc. ISM_FLIP: Flip Source. Use 0 for none, 1 for TB, 2 for LR, 3 for TB&LR. ISM_ROTATE: Rotate Source: Use 0 for none, 1 for 90, 2 for 180, 3 for 270. ISM_WAVE: Wavelength. Use 0 for RGB, 1 for 1+2+3, 2 for wave #1, 3 for wave #2, etc. ISM_FIELD: Field number. ISM_PSAMP: Pupil Sampling. Use 1 for 32x32, 2 for 64x64, etc. ISM_ISAMP: Image Sampling. Use 1 for 32x32, 2 for 64x64, etc. ISM_PSFX, ISM_PSFY: The number of PSF grid points. ISM_ABERRATIONS: Use 0 for none, 1 for geometric, 2 for diffraction. ISM_POLARIZATION: Use 0 for no, 1 for yes. ISM_FIXEDAPERTURES: Use 0 for no, 1 for yes. ISM_USERI: Use 0 for no, 1 for yes. ISM_SHOWAS: Use 0 for Simulated Image, 1 for Source Bitmap, and 2 for PSF Grid. ISM_REFERENCE: Use 0 for chief ray, 1 for vertex, 2 for primary chief ray. ISM_SUPPRESS: Use 0 for no, 1 for yes. ISM_PIXELSIZE: Use 0 for default or the size in lens units. ISM_XSIZE, ISM_YSIZE: Use 0 for default or the number of pixels. ISM_FLIPIMAGE: Use 0 for none, 1 for top-bottom, etc. ISM_OUTPUTFILE: The output file name or empty string for no output file.
MTF - FFT	MTF_SAMP: The pupil sampling, use 1 for 32, 2 for 64, etc. MTF_WAVE: The wavelength number, use 0 for all. MTF_FIELD: The field number, use 0 for all. MTF_TYPE: Use 0 for modulation, 1 for real, 2 for imaginary, 3 for phase, 4 for square wave. MTF_SURF: The surface number, use 0 for image. MTF_MAXF: The maximum frequency, use 0 for default. MTF_SDLI: Show diffraction limit, 0 for no, 1 for yes. MTF_POLAR: Polarization, 0 for no, 1 for yes. MTF_DASH: Use dashes, 0 for no, 1 for yes.
NSC Object Viewer	SHA_ROTX: The x rotation in degrees. SHA_ROTY: The y rotation in degrees. SHA_ROTZ: The z rotation in degrees.

NSC Shaded Model	SHA_ROTX: The x rotation in degrees. SHA_ROTY: The y rotation in degrees. SHA_ROTZ: The z rotation in degrees.
Partially Coherent Image Analysis	PCI_FIELD: The field number. PCI_FILESIZE: The file size. PCI_WAVE: The wavelength number. PCI_RESAMPLE: The resample image setting, 0 for no 1 for yes. PCI_RSNX: The resample number x PCI_RSNY: The resample number y PCI_RSDCX: The resample decenter x PCI_RSDCY: The resample decenter y PCI_RSDLX: The resample delta x PCI_RSDLY: The resample delta y
Polarization Pupil Map	PPM_SAMP: The sampling, use 0 for 3x3, 1 for 5x5, 2 for 7x7, etc. PPM_FIELD: The field number. PPM_WAVE: The wavelength number. PPM_SURFACE: The surface number. PPM_JX: The Jx amplitude. PPM_JY: The Jy amplitude. PPM_PX: The Px phase. PPM_PY: The Py phase. PPM_ADDCONFIG: The add configs string. PPM_SUBCONFIGS: The subtract configs string.
Physical Optics Propagation - General Tab	POP_END: The end surface. POP_FIELD: The field number. POP_START: The starting surface. POP_WAVE: The wavelength number.

Physical Optics Propagation - Beam Definition Tab	POP_AUTO: Simulates the pressing of the "auto" button which chooses appropriate X and Y beam widths based upon the sampling and other settings. POP_BEAMTYPE: Selects the beam type. Use 0 for Gaussian Waist, 1 for Gaussian Angle, 2 for Gaussian Size + Angle, 3 for Top Hat, 4 for File, 5 for DLL and 6 for Multimode. POP_PARAMn: Sets beam parameter n, for example, use POP_PARAM3 to set parameter3. For example if the Beam Type is Gaussian Waist, there will be 8 parameters listed in that order.  The parameters are read row by row. POP_PEAKIRRAD: Sets the normalization by peak irradiance. POP_POWER: Sets the normalization by total beam power. POP_SAMPX: The X direction sampling, use 1 for 32, 2 for 64, etc. POP_SAMPY: The Y direction sampling, use 1 for 32, 2 for 64, etc. POP_SOURCEFILE: The file name if the starting beam is defined by a ZBF file, DLL, or multimode file. POP_WIDEX: The X direction width. POP_WIDEY: The Y direction width.
Physical Optics Propagation - Fiber Data Tab	POP_COMPUTE: Use 1 to check the fiber coupling integral on, 0 to check it off. POP_FIBERFILE: The file name if the fiber mode is defined by a ZBF or DLL. POP_FIBERTYPE: Use the same values as POP_BEAMTYPE above, except for multimode which is not yet supported. POP_FPARAMn: Sets fiber parameter n, for example, use POP_FPARAM3 to set fiber parameter3. POP_IGNOREPOL: Use 1 to ignore polarization, 0 to consider polarization. POP_POSITION: Fiber position setting. Use 0 for chief ray, 1 for surface vertex. POP_TILTX: The X-Tilt. POP_TILTY: The Y-Tilt.

Relative Illumination	REL_RAYDENSITY: The number of rays. REL_FIELDDENSITY: The number of field points. REL_WAVE: The wavelength number, use 0 for all. REL_POLAR: Use 1 to use polarization, 0 to ignore polarization REL_LOG: Use 1 for a log scale, 0 for linear. REL_REMOVEVIGNETTING: Use 1 to remove vignetting factors, otherwise 0. REL_SCANTYPE: Use 0 for +y, 1 for +x, 2 for -y, or 3 for -x scan direction.
Shaded Model	SHA_ROTX: The x rotation in degrees. SHA_ROTY: The y rotation in degrees. SHA_ROTZ: The z rotation in degrees.
Spot Diagram	SPT_RAYS: The ray density.
Surface Sag	SRS_SAMP: The sampling. Use 1 for 33x33, 2 for 65x65, etc. SRS_SURF: The surface number.
Universal Plot 1D	UN1_CATEGORY: Use 0 for surface, 1 for system, 2 for config. UN1_PARAMETER: Use 0 for first option, 1 for second option, etc. UN1_SURFACE: The surface or configuration number. UN1_STARTVAL: The start value for the independent variable. UN1_STOPVAL: The stop value for the independent variable. UN1_STEPS: The number of steps between start and stop. UN1_OPERAND: The optimization operand name. UN1_MFLINE: The optimization operand line number. Use 0 for MF value. UN1_PAR1: Operand parameter 1. UN1_PAR2: Operand parameter 2. UN1_PAR3: Operand parameter 3. UN1_PAR4: Operand parameter 4. UN1_PAR5: Operand parameter 5. UN1_PAR6: Operand parameter 6. UN1_PAR7: Operand parameter 7. UN1_PAR8: Operand parameter 8. UN1_PLOTMIN: The minimum plot value for the dependent variable. UN1_PLOTMAX: The maximum plot value for the dependent variable. UN1_TITLE: The plot title.

Universal Plot 2D	UN2_CATEGORYX: Use 0 for surface, 1 for system, 2 for config. UN2_PARAMETERX: Use 0 for first option, 1 for second option, etc. UN2_SURFACEX: The surface or configuration number. UN2_STARTVALX: The start value for the independent variable. UN2_STOPVALX: The stop value for the independent variable. UN2_STEPSX: The number of steps between start and stop. UN2_CATEGORYY: Use 0 for surface, 1 for system, 2 for config. UN2_PARAMETERY: Use 0 for first option, 1 for second option, etc. UN2_SURFACEY: The surface or configuration number. UN2_STARTVALY: The start value for the independent variable. UN2_STOPVALY: The stop value for the independent variable. UN2_STEPSY: The number of steps between start and stop. UN2_OPERAND: The optimization operand name. UN2_MFLINE: The optimization operand line number. Use 0 for MF value. UN2_PAR1: Operand parameter 1. UN2_PAR2: Operand parameter 2. UN2_PAR3: Operand parameter 3. UN2_PAR4: Operand parameter 4. UN2_PAR5: Operand parameter 5. UN2_PAR6: Operand parameter 6. UN2_PAR7: Operand parameter 7. UN2_PAR8: Operand parameter 8. UN2_SHOWAS: Data display. Use 0 for surface, 1 for contour, etc. UN2_CONTOURFORMAT: Contour format string. UN2_PLOTMIN: The minimum plot value for the dependent variable. UN2_PLOTMAX: The maximum plot value for the dependent variable. UN2_TITLE: The plot title.
Wavefront Map	WFM_SAMP: The sampling, use 1 for 32, 2 for 64, etc. WFM_FIELD: The field number. WFM_WAVE: The wavelength number. WFM_SUBSR: The sub aperture radius. WFM_SUBSX: The sub aperture X decenter. WFM_SUBSY: The sub aperture Y decenter.

Table 2: DVW_SHOW Example

Value	Graphic Window, Detector Color	Text Window, Detector Polar

0	Gray Scale	Full Listing
1	Inverse False Color	Azimuth Cross Section
2	False Color	
3	Inverse False Color	
4	True Color	
5	Cross Section Row	
6	Cross Section Column	

Table 3: DVW_DATA Example

Value	Graphic Window, Detector Rectangle, Source Units: Lumens	Text Window, Summarize All
0	Incoherent Illuminance	Incoherent Illuminance
1	Coherent Illuminance	Coherent Illuminance
2	Coherent Phase	
3	Luminous Intensity	
4	Luminance (position space)	
5	Luminance (angle space)	

The value parameter is the new data for the specified setting. The modified settings file is written back to the original settings file name.

See also GETTEXTFILE .

Example:

```
MODIFYSETTINGS "C:\MySPT.CFG", SPT_RAYS, 24
MODIFYSETTINGS "C:\MyPOP.CFG", POP_SOURCEFILE, "MyStartBeam.ZBF"
```

10.2.14.93. NEXT

See " FOR, NEXT ".

10.2.14.94. NSLT

Initiates a Non-sequential LightningTrace.

Syntax:

```
NSLT surf, source, ray_sampling, edge_sampling
```

Discussion:

Surf is an integer value that indicates the number of the Non-sequential surface. If the program mode is set to Non-Sequential, use 1. Source refers to the object number of the desired source. If source is zero, all sources will be traced. Ray_sampling refers to the resolution of the LightningTrace mesh, with valid values being between 0 (corresponding to a mesh resolution of "Low (1X)") and 5 (corresponding to a mesh resolution of "1024X").

Edge_sampling refers to the resolution used in refining the LightningTrace mesh near the edges of objects, with valid values being between 0 (corresponding to a mesh resolution of "Low (1X)") and 4 (corresponding to a mesh resolution of "256X"). For more details on LightningTrace, see "The LightningTrace Control".

NSLT always calls UPDATE before executing a LightningTrace to make certain all objects are correctly loaded and updated.

Example:

```
NSLT 1, 0, 3, 2
```

10.2.14.95. NSTR

Initiates a Non-sequential trace with ability to save data to a ZRD file.

Syntax:

```
NSTR surf, source, split, scatter, usepolar, ignore_errors, random_seed, save,
savefilename, filter, zrd_format
```

Discussion:

Surf is an integer value that indicates the number of the Non-sequential surface. If the program mode is set to Non-Sequential, use 1. Source refers to the object number of the desired source. If source is zero, all sources will be traced. If Split is non-zero, then splitting is on, otherwise, ray splitting is off. If Scatter is non-zero, then scattering is on, otherwise scattering is off. If Usepolar is non-zero then polarization will be used, otherwise polarization is off. If splitting is on polarization is automatically selected. If ignore_errors is non-zero, then errors will be ignored, otherwise ray errors will terminate the non-sequential trace and macro execution and an error will be reported.

If random_seed is zero, then the random number generator will be seeded with a random value, and every call to NSTR will produce different random rays. If random_seed is any integer other than zero, then the random number generator will be seeded with the specified value, and every call to NSTR using the same seed will produce identical rays. When using NSTR for optimization, it is recommended that a non-zero value be used for random_seed.

If save is omitted or is zero, the arguments savefilename, filter, and zrd_format need not be supplied. If save is not zero, the rays will be saved to a file. The saved data file will have the name specified by the savefilename. The file naming convention and destination folder are the same as described in "Saving ray data to a file". The extension of savefilename should be provided, but no path should be specified. If save is not zero, then the optional filter name is either a string variable with the filter, or the literal filter in double quotes. If no filter is being used, enter an empty pair of double quotes like this: "". For information on filter strings see "The filter string". For ZRD files, the zrd_format can be 0, 1, or 2 for uncompressed full data, compressed basic data, or compressed full data, respectively. For more information on ZRD formats see " Ray database (ZRD) files ".

NSTR always calls UPDATE before tracing rays to make certain all objects are correctly loaded and updated.

Related Functions:

NSTR2

NSDD

Example:

```
NSTR 1, 0, 0, 0, 0, 1, 0, 1, "saverays.ZRD", "h2"
```

10.2.14.96. NSTR2

Initiates a Non-sequential trace with ability to save data to a ZRD and/or PAF file.

Syntax:

```
NSTR surf, source, split, scatter, usepolar, ignore_errors, random_seed, save,
savefilename, filter, zrd_format, savepaths, pathfilename
```

Discussion:

Surf is an integer value that indicates the number of the Non-sequential surface. If the program mode is set to Non-Sequential, use 1. Source refers to the object number of the desired source. If source is zero, all sources will be traced. If Split is non-zero, then splitting is on, otherwise, ray splitting is off. If Scatter is non-zero, then scattering is on, otherwise scattering is off. If Usepolar is non-zero then polarization will be used, otherwise polarization is off. If splitting is on polarization is automatically selected. If ignore_errors is non-zero, then errors will be ignored, otherwise ray errors will terminate the non-sequential trace and macro execution and an error will be reported.

Save, savefilename, and zrd_format refer to saving ZRD, DAT, or SDF files. If save is omitted or is zero, the arguments savefilename, filter, and zrd_format need not be supplied. If save is not zero, the rays will be saved to a file. The saved data file will have the name specified by the savefilename. The file naming convention and destination folder are the same as described in "Saving ray data to a file". The extension of savefilename should be provided, but no path should be specified. For ZRD files, the zrd_format can be 0, 1, or 2 for uncompressed full data, compressed basic data, or compressed full data, respectively. For more information on ZRD formats see "Ray database (ZRD) files".

Savepaths and pathfilename control the saving of a PAF file and have similar requirements to thesave and savefilename arguments above. If savepaths is not zero, the rays will be saved to a PAF file with a name specified by pathfilename. The extension .paf should be specified at the end of the savefilename string.

If either save or savepaths is not zero, then the optional filter name is either a string variable with the filter, or the literal filter in double quotes. If no filter is being used, enter an empty pair of double quotes like this: "". For information on filter strings see "The Filter String".

NSTR2 always calls UPDATE before tracing rays to make certain all objects are correctly loaded and updated.

Related Functions:

NSTR

NSDD

Example:

```
NSTR2 1, 0, 0, 0, 0, 1, 0, 1, "saverays.ZRD", "h2", 0, 1, "saverays.PAF"
```

10.2.14.97. NUMFIELD

This command is obsolete. See " SETSYSTEMPROPERTY, SYSP ".

10.2.14.98. NUMWAVE

This command is obsolete. See " SETSYSTEMPROPERTY, SYSP ".

10.2.14.99. OPEN

Opens an existing text file for reading by the READ command.

Syntax:

```
OPEN "filename"
OPEN A$
```

Discussion:

The filename provided must be a valid file within quotes, or a string variable name containing the file name. See the keywords READ and CLOSE. Always CLOSE a file after all the data has been read.

Related Functions:

EOFF

Related Keywords:

READ, READNEXT, CLOSE

Example:

```
PRINT "Reading the double-column file TEST.DAT!" OPEN "TEST.DAT"
READ x1, y1
READ x2, y2
READ x3, y3
CLOSE
```

10.2.14.100. OPENANALYSISWINDOW

Opens a new analysis window.

Syntax:

```
OPENANALYSISWINDOW type, settingsfilename
```

Discussion:

The type argument is a 3 character string code that indicates the type of analysis to be performed. For the full list of string codes, see the " String Codes " section of the The Programming Tab. If no string code is provided or recognized, no window is opened. The settingsfilename contains the name of the CFG settings file to use. If this argument is missing, the default settings will be used.

Related Functions:

WINL, WINN

Related Keywords:

CLOSEWINDOW, MODIFYSETTINGS

Example:

```
FORMAT 0.0
PRINT "There are ", WINN(), " Windows currently open."
OPENANALYSISWINDOW "mtf"
PRINT "Just opened window ", WINL(), "."
PAUSE TIME, 3000
PRINT "There are ", WINN(), " Windows currently open."
CLOSEWINDOW WINL()
PAUSE TIME, 1000
PRINT "There are ", WINN(), " Windows currently open."
```

10.2.14.101. OPTIMIZE (keywords)

Invokes the optimization algorithm to optimize the current lens with the current merit function.

Syntax:

```
OPTIMIZE
OPTIMIZE number_of_cycles
OPTIMIZE number_of_cycles, algorithm
```

Discussion:

The expression for number_of_cyles must evaluate to an integer value between 1 and 99, and the optimization algorithm will run the specified number of cycles. If number_of_cycles evaluates to zero, then the optimization runs in "Automatic" mode, stopping when the algorithm detects the process has converged. For the algorithm argument, use 0 for Damped Least Squares (the default) and 1 for Orthogonal Descent. For more information see " Performing an optimization ".

To update the merit function without optimizing, use the MFCN function.

Related Functions:

MFCN

Example:

```
PRINT "Starting merit function:", MFCN()

OPTIMIZE

PRINT "Ending merit function :", MFCN()
```

10.2.14.102. OPTRETURN

Used to return numerical values back to the optimization algorithm through the use of the ZPLM optimization operand.

Syntax:

```
OPTRETURN datafield, result
```

Discussion:

The datafield expression must evaluate to an integer between 0 and 50. The datafield refers to a position in an array where the value of the expression result may be stored. The sole purpose of OPTRETURN is to be able to optimize values computed within a ZPL macro.

The optimization operand ZPLM must be used in the merit function to call the ZPL macro and retrieve the value returned by OPTRETURN. See " Optimizing with ZPL macros " for details.

Example:

```
x = sqrt(thic(3) + radi(5))
OPTRETURN j, x+5
```

10.2.14.103. OUTPUT

Specifies destination for text output. Output is either to the screen, or to a file.

Syntax:

```
OUTPUT SCREEN
OUTPUT filename
OUTPUT filename, APPEND
```

Discussion:

If OUTPUT SCREEN is specified alone, then all subsequently executed PRINT commands will be directed to the screen. If a valid filename is provided, then subsequent PRINT commands will output to the filename specified. If a filename with no path is provided, output will be directed to the <data>\Macros folder.

To close the file created earlier, use OUTPUT SCREEN which will direct subsequent PRINT outputs to the screen. SHOWFILE will close the file and send it to the text viewer program for screen display. PRINTFILE will close the file and print it on the currently defined printer.

Note that when sending PRINT commands to an output file, the file will be in Unicode format. To convert the file to ANSI, first close the file (for example using OUTPUT SCREEN) and then use the CONVERTFILEFORMAT keyword.

If the keyword APPEND follows the file name, then subsequent output will be appended to the file. Otherwise, the contents of the file will be overwritten.

Example:

```
OUTPUT "x.txt"
PRINT "This will not appear on the screen, but in the file x.txt."
OUTPUT SCREEN
PRINT "This will appear on the screen."
OUTPUT "x.txt", APPEND
PRINT "This will appear after the first line in the file x.txt."
```

Related Keywords:

CLOSE, OPEN, SHOWFILE, PRINTFILE

10.2.14.104. PARM

For setting surface parameters use "SETSURFACEPROPERTY, SURP" keyword, and for getting surface parameters use the PARM(n,s) Numeric Function as discussed under " Numeric Functions ".

10.2.14.105. PARAXIAL (keywords, programming tab, about the zpl)

Used to control whether ray tracing is done with paraxial or real rays.

Syntax:

```
PARAXIAL ON
PARAXIAL OFF
```

Discussion:

The current paraxial mode can be established by a call to the function PMOD, which returns 0 if paraxial mode is off, 1 if it is on. This feature is used for switching between real and paraxial ray traces. Certain calculations, such as measuring distortion and computing first-order properties such as effective focal length, require the tracing of paraxial rays.

Example:

```
mode = PMOD()
IF mode THEN PRINT "Paraxial mode is on!"
IF !mode THEN PRINT "Paraxial mode is not on!"
PARAXIAL ON
PRINT "Now paraxial mode is on!"
PRINT "Restoring original mode..."
if !mode THEN PARAXIAL OFF
```

10.2.14.106. PAUSE

Pauses macro execution, optionally while displaying a status message. The status message can be a string or numerical value. The macro continues once the user presses the "OK" button on the status dialog.

Syntax:

```
PAUSE
PAUSE "Ready to continue..."
PAUSE TIME, time
PAUSE THREADS
```

Discussion:

This feature may be used for debugging, presenting results, or pausing the execution of the macro. If the keyword after the PAUSE command is TIME, then the macro execution will "sleep" for at least the number of milliseconds given by the value provided. For example, to pause the macro for 100 milliseconds, the syntax is

```
PAUSE TIME, 100
```

The macro execution will continue after the specified time interval has passed. No status message is displayed in this case. Note that the system timer has a resolution of about 10 milliseconds, so the actual delay introduced by PAUSE TIME will be between the requested time and the requested time plus (approximately) 10 milliseconds. The maximum allowed timer interval is 1,000,000 milliseconds (about 16.7 minutes).

If the keyword after the PAUSE command is THREADS, the macro execution will pause until all open windows complete whatever computation they are currently executing. This is a very useful feature when calling UPDATE ALL or LOADLENS using session files.

10.2.14.107. PIXEL

Turns on a single pixel on the current graphics screen.

Syntax:

```
PIXEL xcoord, ycoord
```

Discussion:

This feature is useful for making spot diagrams. See " GRAPHICS ".

10.2.14.108. PLOT

The PLOT keyword supports a number of arguments used to simplify the task of creating plots of numeric data.

Syntax:

```
PLOT NEW
PLOT TITLE, string
PLOT TITLEX, string
PLOT TITLEY, string
PLOT BANNER, string
PLOT WINASPECT, type
PLOT COMM1, string
PLOT COMM2, string
PLOT COMM3, string
PLOT COMM4, string
PLOT COMM5, string
```

```
PLOT COMM6, string
PLOT RANGEX, minx, maxx
PLOT RANGEY, miny, maxy
PLOT CHECK, x_increment, y_increment
PLOT TICK, x_increment, y_increment
PLOT FORMATX, format_string
PLOT FORMATY, format_string
PLOT DATA, x_array, y_array, number_of_points, color, style, options
PLOT LINE, x1, y1, x2, y2
PLOT LABEL, x, y, angle, size, string
PLOT GO
```

Discussion:

PLOT NEW is used to initialize a new plot. All data related to any previously generated plot is discarded. This should always be the first PLOT command used when making a new graphic.

PLOT TITLE, PLOT TITLEX, and PLOT TITLEY are used to define the main plot title, the x-axis title, and the y-axis title, respectively. The parameter "string" may be a literal string in quotes or may be a string variable.

PLOT BANNER is used to define the banner title. The parameter "string" may be a literal string in quotes or may be a string variable.

PLOT WINASPECT defines the aspect ratio format of the window that will display the window. The valid values for type are the integers 0 through 3, which yield aspect ratios of 4x3, 5x3, 3x4, and 3x5, respectively. If no WINASPECT command is issued a default aspect ratio is used.

PLOT COMMx is used to define up to 6 comments that appear on the bottom of the plot. The parameter "string" may be a literal string in quotes or may be a string variable.

PLOT RANGEX and PLOT RANGEY define the minimum and maximum values of the plot in the x- and y- directions respectively. If no RANGEX or RANGEY command is issued, a default range will be selected.

PLOT CHECK is used to define the size of the "X" marks used to mark data points in the PLOT DATA command. The units are dimensionless fractions relative to the width of the display. If no CHECK command is issued, a default size will be selected.

PLOT TICK is used to define the increments between tick marks on the x- and y-axis, respectively. If no TICK command is issued, a default increment will be selected.

PLOT FORMATX and PLOT FORMATY are used to define the format strings for numeric labels on the x- and y-axis, respectively. The parameter "format_string" may be a literal string in quotes or may be a string variable. The format_string must be a valid C-language format specifier. A common format string is "%M.nf" where M is the total number of spaces in the output string and n is the number of figures after the decimal point. For example, the value for pi will print as "3.1415" if the format string is "%6.4f". To define an exponential format, the format string is of the form "%M.nE" which would yield a total of M characters and n is the number of figures after the decimal point. The value for pi would print as "3.14E+000" if the format_string was "%9.3E". If no FORMATX or FORMATY is specified a default format is used.

PLOT DATA is used to define a series of data points to be plotted. The variables x_array and y_array MUST be array variables of 1 dimension. For information on defining array variables see "Array variables". The number_of_points defines how many points in the array are to be used. The color argument is 0 for black, or 1-16 to select different pen colors. To define pen colors, see "Colors 1-12, Colors 13-24". The style argument defines the line style; the current supported values are 0 for a solid line, or 1-4 for various styles of dashed lines. The options value is 0 for no marks indicating the location of data points, 1 for both a line and data point marks, or 2 for no line and data point marks only. Multiple PLOT DATA commands may be issued, and each will create a separate line or curve to be plotted.

PLOT LINE makes a line between the points (x1, y1) and (x2, y2). The units of x and y here are normalized screen coordinates between 0.0 and 1.0. A maximum of 1000 lines may be defined.

PLOT LABEL prints any text string on the plot starting at the coordinates (x, y). The units of x and y here are normalized screen coordinates between 0.0 and 1.0. The angle is the angle from the +x direction in degrees. The size is a scaling

constant greater than 0.0 that defines the size of the font. A size of 1.0 would print at the normal font size, while a value of 1.5 would print at 50% larger than normal size. The parameter "string" may be a literal string in quotes or may be a string variable. A maximum of 100 labels may be defined.

PLOT GO uses all settings and data from previous PLOT commands and generates the desired plot, and displays the plot in a window. After the plot is generated all the PLOT data is reset as though PLOT NEW had been executed.

Related Keywords:

PLOT2D

10.2.14.109. PLOT2D

The PLOT2D keyword supports a number of arguments used to simplify the task of creating plots of numeric data. This keyword makes it easy to generate surface, contour, grey scale, and false color plots.

Syntax:

```
PLOT2D NEW
PLOT2D TITLE, string
PLOT2D COMM1, string
PLOT2D COMM2, string
PLOT2D COMM3, string
PLOT2D COMM4, string
PLOT2D COMM5, string
PLOT2D RANGE, min, max
PLOT2D ASPECT, ratio
PLOT2D WINASPECT, type
PLOT2D DATA, arrayname
PLOT2D ACTIVECURSOR, left, right, bottom, top
PLOT2D DISPLAYTYPE, type
PLOT2D CONTOURINTERVAL, string
PLOT2D SURFACESCALE, scale
PLOT2D LOGPLOT, peak, decades
PLOT2D HIDEADDRESS, flag
PLOT2D CONFIG, configuration
PLOT2D GO
```

Discussion:

PLOT2D NEW is used to initialize a new plot. All data related to any previously generated plot2d is discarded. This should always be the first PLOT2D command used when making a new graphic.

PLOT2D TITLE is used to define the main plot title. The parameter "string" may be a literal string in quotes or may be a string variable.

PLOT2D COMMx is used to define up to 5 comments that appear on the bottom of the plot. The parameter "string" may be a literal string in quotes or may be a string variable.

PLOT2D RANGE define the minimum and maximum values of the plot. If no RANGE command is issued, a default range will be selected. The range should span the data values, or the data will be truncated to fit the range. The surface and contour plots ignore this command.

PLOT2D ASPECT defines the ratio of the x-width of the plot as compared to the y-width. The surface plots ignore this command.

PLOT2D WINASPECT defines the aspect ratio format of the window that will display the window. The valid values for type are the integers 0 through 3, which yield aspect ratios of 4x3, 5x3, 3x4, and 3x5, respectively. If no WINASPECT command is issued a default aspect ratio is used.

PLOT2D DATA is used to define the array variable containing the data to be plotted. The variable arrayname MUST be an array variables of 2 dimensions. For information on defining array variables see "Array variables". The number

of points in the x and y directions is defined by the array dimensions. Any number of points is allowed, however, the minimum number of points in either direction is 5. The data will be assuming the two dimensions form a rectangular grid with uniform spacing along each dimension, although an overall aspect ratio for the plot may be defined using PLOT2D ASPECT. Only one PLOT2D DATA command may be issued.

PLOT2D ACTIVECURSOR is used to define the left, right, bottom, and top boundaries of the plotted data for display on contour, grey scale, and false color plots.

PLOT2D DISPLAYTYPE is used to select the type of plot generated. The value for type should be an integer between 1 and 6, inclusive, for surface, contour, grey scale, inverse grey scale, false color, and inverse false color, respectively.

PLOT2D CONTOURINTERVAL is used to define the contour format string. Only contour plots use this command. For details on the contour interval format string, see "The Contour Format String".

PLOT2D SURFACESCALE is used to define an overall vertical scaling of the surface plot. Only surface plots use this command. The default scaling value is 0.5 arbitrary units.

PLOT2D LOGPLOT is used to generate a log scale display. Only grey scale, inverse grey scale, false color, and inverse false color plots use this command. The peak value is an integer indicating the maximum power of 10 to plot. For example, is peak is 3, then the maximum value plotted will be 1E+03. The decades value is an integer indicating the number of log decades below the peak to plot. If peak is 3, and decades is 5, then the plot will span the range from 1.0E+03 to 1.0E-02. Note that the use of LOGPLOT does not actually take the logarithm of the input array data. The macro itself must take the log base 10 of the values in the array before calling PLOT2D GO.

PLOT2D HIDEADDRESS is used to hide or show the address data. If the flag value is zero, the address is shown, otherwise the address is not shown. See also " Address ".

PLOT2D CONFIG is used to define the configuration number shown if the address box is displayed and the address box displays information about the current configuration. If configuration is 0, the address box will show the current configuration number. If the configuration is an integer between 1 and the number of configurations, inclusive, the specific configuration number will be shown. If configuration is an integer less than 0, then "All" is shown for the configuration number.

PLOT2D GO uses all settings and data from previous PLOT2D commands and generates the desired plot, and displays the plot in a window. After the plot is generated all the PLOT2D data is reset as though PLOT2D NEW had been executed.

Related Keywords:

PLOT

10.2.14.110. POLDEFINE

Defines the input polarization state for subsequent POLTRACE calls.

Syntax:

```
POLDEFINE Jx, Jy, PhaX, PhaY
```

Discussion:

The POLDEFINE keyword is used to define the input polarization state for subsequent polarization ray tracing. POLDEFINE requires the Jx and Jy electric field magnitudes, as well as the X and Y phase angles in degrees. For more information on the meaning of Jx and Jy, see " Defining the initial polarization ". The input values are automatically normalized to have unity magnitude. The default values are 0, 1, 0, and 0, respectively. Once the polarization state is defined, it remains the same until changed.

Example:

```
POLDEFINE 2.0, 2.0, 45.0, -66.0
```

Related Keywords:

POLTRACE

10.2.14.111. POLTRACE

Calls the OpticStudio polarization ray tracing routines to trace a particular ray through the current system.

Syntax:

```
POLTRACE Hx, Hy, Px, Py, wavelength, vec, surf
```

Discussion:

The expressions Hx and Hy must evaluate to values between -1 and 1, and represent the normalized object coordinates. The pupil coordinates are specified by the expressions Px and Py, which also must be between -1 and 1. For more information about normalized coordinates, see the chapter "Conventions and Definitions" under "Normalized field and pupil coordinates". The wavelength expression must evaluate to an integer between 1 and the maximum number of defined wavelengths. The vec expression must evaluate to a number between 1 and 4, inclusive. The surf expression must evaluate to an integer between 1 and the number of surfaces, inclusive.

The input polarization state of the ray is defined by the POLDEFINE keyword.

Once the ray is traced, the polarization data for the ray is placed in the vector variable specified by the vec expression. For example, if the command "POLTRACE Hx, Hy, Px, Py, w, 2, n" is issued, the data will be stored in VEC2. The data is stored in the following format, where the first number in each line refers to the array position:

```
0: n, the number of data entries in the vector
1: The ray intensity after the surface
2: E-Field X component, real
3: E-Field Y component, real
4: E-Field Z component, real
5: E-Field X component, imaginary
6: E-Field Y component, imaginary
7: E-Field Z component, imaginary
8: S-Polarization field amplitude reflection, real
9: S-Polarization field amplitude reflection, imaginary
10: S-Polarization field amplitude transmission, real
11: S-Polarization field amplitude transmission, imaginary
12: P-Polarization field amplitude reflection, real
13: P-Polarization field amplitude reflection, imaginary
14: P-Polarization field amplitude transmission, real
15: P-Polarization field amplitude transmission, imaginary
16: E-Field X direction phase Px
17: E-Field Y direction phase Py
18: E-Field Z direction phase Pz
19: Major axis length of polarization ellipse
20: Minor axis length of polarization ellipse
21: Angle of polarization ellipse in radians
22: The surface number at which the ray was vignetted or zero if not vignetted
23: S-Polarization ray amplitude reflection, real
24: S-Polarization ray amplitude reflection, imaginary
25: S-Polarization ray amplitude transmission, real
26: S-Polarization ray amplitude transmission, imaginary
27: P-Polarization ray amplitude reflection, real
28: P-Polarization ray amplitude reflection, imaginary
29: P-Polarization ray amplitude transmission, real
30: P-Polarization ray amplitude transmission, imaginary
```

If the value in array position 0 is 0, then an error occurred and the polarization data is invalid. This may occur if the specified ray cannot be traced. See the RAYTRACE command to extract extended error information.

Example:

```
POLDEFINE 0, 1, 0, 0
POLTRACE 0, 1, 0, 0, pwav(), 1, nsur()
PRINT "Transmission of chief ray at primary wavelength is ", vec1(1)
```

Related Keywords:

POLDEFINE, RAYTRACE

10.2.14.112. POP

Computes the Physical Optics Propagation (POP) of a beam through the optical system and saves the surface by surface results to ZBF files. For a description of the POP feature see " Physical Optics Propagation ". For information on the ZBF file format, see " Zemax Beam File (ZBF) binary format ".

Syntax:

```
POP outfilename, lastsurface, settingsfilename
```

Discussion:

This keyword requires the name of the output ZBF file, an expression that evaluates to the last surface to propagate to, and optionally the name of a settings file. The filename must be enclosed in quotes if any blank or other special characters are used. The created ZBF files will be placed in the <pop> folder. No paths should be provided with the file names.

The settings for the POP feature will be those settings previously saved for the current lens, unless a settings file name is provided. The settings file name must include the full path, name, and extension. To make adjustments to the settings, open a POP window, choose the appropriate settings, then press "Save". By default, all subsequent calls to POP within ZPL will use the saved settings. The exceptions are the output file name, which is specified as the first argument after the POP keyword, and the last surface number, which is optionally specified as the second argument after the POP keyword.

Example:

```
POP "pop_output.ZBF", 12, "MyPOP.CFG"
```

10.2.14.113. PRINT

Print is used to output constant text and variable data to either the screen or a file, depending upon the current status of the keyword OUTPUT.

Syntax:

```
PRINT
PRINT X
PRINT "The value of x is ", x
PRINT " x = ", x, " x + y = ", x + y
```

Discussion:

PRINT alone will print a blank line. PRINT with a list of text arguments and expressions will print each text string (enclosed in double quotes) and the numeric value of each expression. PRINT uses the numerical output format specified by FORMAT. If the last item in the list is followed by a comma, PRINT will not end the line with a carriage return.

Note that very large quantities of PRINT commands (i.e. thousands of lines or more) can negatively impact macro execution speed if the printed text is displayed on the screen (i.e. OUTPUT SCREEN).

The reason this can cause slowdown is because OpticStudio handles these print windows as a single string. As the string gets longer and longer (as you print more lines of data out to the window), it becomes more computationally costly to display it and store it. In other words, the problem is that the PRINT overhead linearly increases with each loop iteration.

So to retrieve a lot of textual data from a macro:

- The best solution is to use PRINT to save to a file instead of pushing to the window (i.e. use OUTPUT filename). It will make the macro more efficient, by avoiding the inefficiencies of printing to the ZPL output window.
- To display the results on the screen, consider using a "child" macro. The "child" macro will only print one string at a time.OUTPUT SCREENa$ = "This text gets really really long."CALLSETSTR 1, a$CALLMACRO PRINT.ZPLThe macro PRINT.ZPL will only display a$:A$ = $CALLSTR(1)PRINT A$

Example:

```
X = 3
PRINT "X equals ",x
```

Related Keywords:

REWIND

10.2.14.114. PRINTFILE

Prints a text file.

Syntax:

```
PRINTFILE filename
```

Discussion:

The filename must be a valid file name. The file must be a text file (as would be created by OUTPUT and PRINT commands in ZPL) and must be in the current folder. PRINTFILE also closes the file if no CLOSE command has been executed.

Example:

```
OUTPUT "test.txt"
PRINT "Print this to the printer."
PRINTFILE "test.txt"
```

Related Keywords:

OPEN, OUTPUT, CLOSE, PRINT, PRINTFILE

10.2.14.115. PRINTWINDOW

Prints any open graphic or text window.

Syntax:

```
PRINTWINDOW winnum
```

Discussion:

The winnum value may be either an integer or an expression that evaluates to an integer. The integer winnum corresponds to the window number that should be printed. OpticStudio numbers windows sequentially as they are opened, starting with 1. Any closed windows are deleted from the window list, without renumbering the windows which remain. Any windows opened after another window has been closed will use the lowest window number available.

Example:

```
PRINTWINDOW 5
```

10.2.14.116. PWAV

Sets the primary wavelength.

Syntax:

```
PWAV n
```

Discussion:

The expression n is evaluated and the primary wavelength is set to the specified number. The UPDATE command must be issued before the new data takes effect.

Example:

```
PWAV 1
```

Related Functions:

WAVL, WWGT, PWAV

10.2.14.117. QUICKFOCUS (keywords)

Adjusts the thickness of the surface prior to the image surface to minimize the specified criteria.

Syntax:

```
QUICKFOCUS mode, centroid
```

Discussion:

The expression for mode should evaluate to 0, 1, 2, or 3 for RMS spot radius, spot x, spot y, or wavefront OPD. The expression for centroid should evaluate to 0 or 1 to indicate the RMS should be referenced to the chief ray or image centroid, respectively. The "best" focus is chosen as a wavelength weighted average over all fields.

Example:

```
! Focus at best RMS wavefront to centroid
QUICKFOCUS 3, 1
```

10.2.14.118. QUICKSENSITIVITY (keywords)

Runs the Quick Sensitivity tool on the current system.

Syntax:

```
QUICKSENSITIVITY criterion, sampling, config, fields, out_file_name
```

Discussion:

Criterion evaluates as follows:

0 = RMS Spot Radius

1 = RMS Spot X

2 = RMS Spot Y

3 = RMS Wavefront

4 = Boresight Error

5 = RMS Angular Radius

6 = RMS Angular X

7 = RMS Angular Y

Sampling integers from 1-20

Config evaluates as follows:

0 = All

1 = 1/n (or however many configs there are)

2 = 2/n

3 = 3/n

.

.

.

x = x/n

Fields evaluates as follows:

0 = Y-Symmetric

1 = XY-Symmetric

2 = User Defined

Out_file_name is a text file (any extension may be used) and the file will be placed in the same folder as the current lens file. This field should not specify a full file path.

10.2.14.119. RADI

This command is obsolete. See " SETSURFACEPROPERTY, SURP ".

10.2.14.120. RANDOMIZE

RANDOMIZE seeds the random number generator.

Syntax:

```
RANDOMIZE seed
```

Discussion:

If seed evaluates to zero, OpticStudio seeds the random number generator with a value based upon the CPU clock. Otherwise, the value provided is used to seed the random number generator. Using the same seed will reproduce the identical series of random numbers created by the RAND function.

Example:

```
RANDOMIZE
RANDOMIZE 250
```

Related Functions:

RAND

10.2.14.121. RAYTRACE

Calls the OpticStudio ray tracing routines to trace a particular ray through the current system.

Syntax:

```
RAYTRACE hx, hy, px, py, wavelength
```

Discussion:

The expressions hx and hy must evaluate to values between -1 and 1, and represent the normalized field coordinates. The pupil coordinates are specified by the expressions px and py, which also must be between -1 and 1. For more information about normalized coordinates see "Normalized field coordinates". The wavelength expression is optional, defaulting to the primary wavelength, but if supplied must evaluate to an integer between 1 and the maximum number of defined wavelengths.

Once the ray is traced, the ray intercept coordinates and direction cosines may be determined using the ZPL functions RAYX, RAYY, RAYZ, RAYL, RAYM, and RAYN (use RAGX, RAGY, RAGZ, RAGL, RAGM, and RAGN to get results in global coordinates). If an error occurred during ray tracing, the function RAYE (for RAY Error) will return a value other than zero. If RAYE is negative, it indicates that total internal reflection occurred at the surface whose number is the absolute value of the value returned. If RAYE returns a value of -9999, the ray cannot be launched.

If RAYE is greater than zero, then the ray missed the surface number returned. Checking RAYE is optional, however, the RAYX, RAYY, ... functions may return invalid data if RAYE is not zero. The functions RANX, RANY, and RANZ return the intercept surface normal direction cosines, and OPDC returns the optical path difference for the ray. The function RAYV returns the surface number at which the ray was vignetted, or it returns zero if the ray was not vignetted. Values returned for surfaces past the surface of vignetting may not be accurate.

Example:

```
PRINT "Tracing the marginal ray at primary wavelength!"
n = NSUR()
RAYTRACE 0,0,0,1
y = RAYY(n)
PRINT "The ray intercept is ", y
PRINT "Tracing the chief ray at maximum wavelength!"
RAYTRACE 0,1,0,0,NWAV()
y = RAYY(n)
PRINT "The ray intercept is ", y
```

Related Keywords:

RAYTRACEX

10.2.14.122. RAYTRACEX

Calls the OpticStudio ray tracing routines to trace a particular ray from any starting surface through the current system.

Syntax:

```
RAYTRACEX x, y, z, l, m, n, surf, wavelength
```

Discussion:

The expressions x, y, z, l, m, and n define the input ray position and direction cosines in the local coordinates of the starting surface. The surface expression must evaluate to an integer between 0 and the number of surfaces minus one, inclusive. The wavelength expression is optional, defaulting to the primary wavelength, but if supplied must evaluate to an integer between 1 and the maximum number of defined wavelengths.

If the object has a thickness of infinity, and the surf parameter is zero, then the input coordinates are assumed to be relative to the first surface rather than the object surface; although the ray will still be defined in object space media. Otherwise, OpticStudio uses the specified coordinates without alteration.

Once the ray is traced, the ray intercept coordinates and direction cosines may be determined using the ZPL functions RAYX, RAYY, RAYZ, RAYL, RAYM, and RAYN (use RAGX, RAGY, RAGZ, RAGL, RAGM, and RAGN to get results in global coordinates). Note only data from surface AFTER the "surf" surface will be valid.

If an error occurred during ray tracing, the function RAYE (for RAY Error) will return a value other than zero. If RAYE is negative, it indicates that total internal reflection occurred at the surface whose number is the absolute value of the value returned. If RAYE is greater than zero, then the ray missed the surface number returned.

Checking RAYE is optional, however, the RAYX, RAYY, ... functions may return invalid data if RAYE is not zero. The functions RANX, RANY, and RANZ return the intercept surface normal direction cosines, and RAYT returns the optical path length up to the surface for the ray. The function RAYV returns the surface number at which the ray was vignetted, or it returns zero if the ray was not vignetted. Values returned for surfaces past the surface of vignetting may not be accurate.

Example:

```
n = NSUR()
RAYTRACEX 0,1,0,0,0,1,0,NWAV()
y = RAYY(n)
PRINT "The ray intercept is ", y
```

Related Keywords:

RAYTRACE

10.2.14.123. READ

Reads data from an existing text file opened for reading by the OPEN command.

Syntax:

```
READ x
READ x, y
READ x,y,z,a,b,c,q
```

Discussion:

The file must be already open, see the keyword OPEN for details. Each READ command reads a single line from the file. The first valid data field from this line is placed in the variable first listed. The data from the second field is placed in the second variable listed, if any.

Therefore, the number of variables listed in the read command should match the number of columns in the text file. Numeric data in the file should be delimited by spaces.

The data may be in free-form, and is internally promoted to double precision. In order to use decimal separator currently selected in Windows settings use READ_LOCALE keyword. A maximum of 2000 characters can be read in

on any single line. The maximum number of variable arguments is 199; for reading longer lines with more arguments use READNEXT instead. The variables listed must be valid ZPL variable names.

READNEXT does not support array variable names as arguments. The workaround is to read the data into a scalar variable and then set the array variable to the scalar variable value in a subsequent line like this:

```
READ x
data(i, j) = x
```

Always CLOSE a file after all the data has been read. See the function EOFF.

Example:

```
PRINT "Reading the double-column file TEST.DAT!"
OPEN "C:\DATA\TEST.DAT"
READ x1, y1
READ x2, y2
READ x3, y3
CLOSE
```

Related Functions:

EOFF

Related Keywords:

OPEN, CLOSE, READNEXT, READSKIP, READSTRING, READ_LOCALE

10.2.14.124. READ_LOCALE

Reads data from an existing text file opened for reading by the OPEN command.

Syntax:

```
READ_LOCALE x
READ_LOCALE x, y
READ_LOCALE x,y,z,a,b,c,q
```

Discussion:

The file must be already open, see the keyword OPEN for details. Each READ command reads a single line from the file. The first valid data field from this line is placed in the variable first listed. The data from the second field is placed in the second variable listed, if any. Therefore, the number of variables listed in the read command should match the number of columns in the text file. Numeric data in the file should be delimited by spaces. The data may be in free-form, and is internally promoted to double precision assuming character currently selected in Windows settings as decimal separator. In order to use period as decimal separator use READ keyword. A maximum of 2000 characters can be read in on any single line. The maximum number of variable arguments is 199; for reading longer lines with more arguments use READNEXT_LOCALE or READNEXT instead. The variables listed must be valid ZPL variable names.

Always CLOSE a file after all the data has been read. See the function EOFF.

Example:

```
PRINT "Reading the double-column file TEST.DAT!"
OPEN "C:\DATA\TEST.DAT"
READ_LOCALE x1, y1
READ_LOCALE x2, y2
READ_LOCALE x3, y3
CLOSE
```

```
Related Functions:
EOFF
Related Keywords:
OPEN, CLOSE, READ, READNEXT, READNEXT_LOCALE, READSKIP, READSTRING
```

10.2.14.125. READNEXT_LOCALE

Reads data from an existing text file opened for reading by the OPEN command.

Syntax:

```
READNEXT_LOCALE x
READNEXT_LOCALE x, y
READNEXT_LOCALE x,y,z,a,b,c,q
```

Discussion:

READNEXT_LOCALE is almost identical to READ_LOCALE. The key difference is READ_LOCALE will read the entire data line from the opened file, up to the newline character, while READNEXT reads only enough characters to fill the number of arguments.

For example, if a data file contains a line with this data:

```
3.0 4.0 5.0
```

The following two READNEXT_LOCALE commands will read the values 3.0, 4.0, and 5.0 for x, y, and z:

```
READNEXT_LOCALE x, y
READNEXT_LOCALE z
```

READNEXT_LOCALE is more useful than READ_LOCALE or READ if the line is very long, or the number of arguments is large. READNEXT_LOCALE does not support array variable names as arguments. The workaround is to read the data into a scalar variable and then set the array variable to the scalar variable value in a subsequent line like this:

```
READNEXT_LOCALE x
data(i, j) = x
```

Example:

```
OPEN "C:\DATA\TEST.DAT"
READNEXT_LOCALE x1, x2
READNEXT_LOCALE x3
CLOSE
```

Related Keywords:

OPEN, CLOSE, READ, READ_LOCALE, READNEXT, READSKIP, READSTRING

10.2.14.126. READNEXT

Reads data from an existing text file opened for reading by the OPEN command.

Syntax:

```
READNEXT x
READNEXT x, y
READNEXT x,y,z,a,b,c,q
```

Discussion:

READNEXT is almost identical to READ. The key difference is READ will read the entire data line from the opened file, up to the newline character, while READNEXT reads only enough characters to fill the number of arguments.

For example, if a data file contains a line with this data:

```
3.0 4.0 5.0
```

The following two READNEXT commands will read the values 3.0, 4.0, and 5.0 for x, y, and z:

```
READNEXT x, y
READNEXT z
```

READNEXT is more useful than READ if the line is very long, or the number of arguments is large. READNEXT does not support array variable names as arguments. The workaround is to read the data into a scalar variable and then set the array variable to the scalar variable value in a subsequent line like this:

```
READNEXT x
data(i, j) = x
```

READNEXT is assuming period as decimal separator. In order to use decimal separator currently selected in Windows settings use READNEXT_LOCALE keyword.

Example:

```
OPEN "C:\DATA\TEST.DAT"
READNEXT x1, x2
READNEXT x3
CLOSE
```

Related Keywords:

OPEN, CLOSE, READ, READSKIP, READSTRING, READNEXT_LOCALE

10.2.14.127. READSKIP

Reads any number of unwanted characters from an existing text file opened for reading by the OPEN command.

Syntax:

```
READSKIP n
```

Discussion:

The file must be already open, see the keyword OPEN for details. If n is positive, READSKIP reads off exactly n characters from the file. If n is negative, READSKIP reads to the end of the current line. The read characters are discarded. This function allows faster access to large files by effectively skipping over any number of unwanted characters. Always CLOSE a file after all the data has been read. See the function EOFF.

Example:

```
PRINT "Reading the contents of file TEST.DAT!"
OPEN TEST.DAT
READSKIP 59
READSTRING A$
PRINT A$
CLOSE
```

Related Keywords:

OPEN, CLOSE READ, READNEXT, READSTRING

10.2.14.128. READSTRING

Reads data from an existing text file opened for reading by the OPEN command.

Syntax:

```
READSTRING A$
```

Discussion:

The file must be already open, see the keyword OPEN for details. Each READ command reads a single line from the file. The entire line read is placed in the variable listed. A maximum of 359 characters can be read in and placed in a single string. The variable listed must be a valid ZPL string variable name, although it does not need to be previously referenced. Always CLOSE a file after all the data has been read. See the function EOFF.

Example:

```
PRINT "Reading the contents of file TEST.DAT!"
OPEN TEST.DAT
READSTRING A$
PRINT A$
CLOSE
```

Related Keywords:

OPEN, CLOSE READ, READNEXT, READSKIP

10.2.14.129. RELEASE

See " Array variables ".

10.2.14.130. RELOADOBJECTS

Reloads NSC objects into the NSC Editor.

Syntax:

```
RELOADOBJECTS surface, object
```

Discussion:

The expression for surface must evaluate to an integer surface number corresponding to the non-sequential component surface. For NSC mode, use 1. The expression for object must evaluate to an integer object number, or zero to reload all objects. This keyword may take a significant amount of time to execute, depending upon the number and type of objects defined.

Example:

```
RELOADOBJECTS 1, 0
```

10.2.14.131. REM, !,

REM, !, and # are used to define remarks or comments.

Syntax:

```
REM (text)
! (text)
(valid zpl line) # (text)
```

Discussion:

The exclamation symbol may also be used to indicate a remark. Both the REM command and the "!" symbol are only recognized as remark indicators if they appear at the very beginning of the line. The # symbol may be used anywhere on the line, but will only be interpreted as the beginning of a comment if the # is not inside a string. Everything after the # symbol is ignored.

Example:

```
REM any text can be placed after the REM command.
! any text can also be placed
! after the exclamation symbol.
x = 5 # this syntax allows comments to be placed on the same line as a command.
```

10.2.14.132. REMOVEVARIABLES (keywords)

Sets all currently defined variables to fixed status.

Syntax:

```
REMOVEVARIABLES
```

10.2.14.133. RESUMEUPDATES

SUSPENDUPDATES prevents any UI editor windows from being updated by ZPL commands while suspended, and RESUMEUPDATES ends the suspension. See SUSPENDUPDATES for more information.

10.2.14.134. RENAMEFILE

RENAMEFILE is used to rename a file.

Syntax:

```
RENAMEFILE oldfilename, newfilename
```

Discussion:

This keyword requires two file names, defined as literal string expressions in quotes or as string variables. The file oldfilename is renamed newfilename.

Example:

```
RENAMEFILE AFILE$, BFILE$
```

Related Keywords:

COPYFILE

DELETEFILE

10.2.14.135. RETURN

See GOSUB .

10.2.14.136. REWIND

REWIND erases the last line printed by the PRINT command, up to the previous end of line. This allows printing a counter or other data over an existing line in the text output file.

Syntax:

```
REWIND
```

Example:

```
PRINT "First line"
REWIND
PRINT "New First line"
```

Related Keywords:

PRINT

10.2.14.137. SAVEARCHIVE

Saves the current lens to a Zemax archive (*.ZAR) file.

Syntax:

```
SAVEARCHIVE filename
```

Discussion:

The filename must include the name of the archive file including the file extension. If the filename does not include a complete path to the archive file, the default folder for lenses is assumed. See "Backup To Archive File".

10.2.14.138. SAVEDETECTOR (keywords)

Saves the data currently on an NSC Detector Rectangle, Detector Color, Detector Polar, or Detector Volume object to a file.

Syntax:

```
SAVEDETECTOR surf, object, filename
```

Discussion:

This keyword requires numeric expressions that specify the surface and object, and a file name with or without a full path. Surf is the surface number of the non-sequential group; use 1 when using non-sequential mode. Object is the object number of the detector object. The filename may include the full path, if no path is provided the path of the current lens file is used. The extension should be DDR, DDC, DDP, or DDV for Detector Rectangle, Color, Polar, and Volume objects, respectively.

Related Keywords:

LOADDETECTOR

10.2.14.139. SAVELENS

Saves the current lens file.

Syntax:

```
SAVELENS filename, session
```

Discussion:

SAVELENS will save the current lens file to the specified file name. The full file path may be optionally specified. The name of the current lens in memory will also be changed. If the file name is absent, then the lens data is stored in the current file name. If the session argument evaluates to anything other than zero, the session file will also be saved.

Example:

```
SAVELENS
SAVELENS "NEWCOPY.ZMX"
SAVELENS NEW$
```

Related Keywords:

LOADLENS

10.2.14.140. SAVEMERIT (keywords)

Saves the current merit function to a file.

Syntax:

```
SAVEMERIT filename
```

Discussion:

SAVEMERIT will save the current merit function to a file. If the filename contains the complete path, such as C:\MYDIR\MYLENS.MF, then the specified path will be used. If the path is left off, then the <data>\MeritFunction folder will be used (see " Folders "). See also LOADMERIT .

10.2.14.141. SAVETOLERANCE (keywords)

Saves the current tolerances to a file.

Syntax:

```
SAVETOLERANCE "filename"
SAVETOLERANCE file$
```

Discussion:

If the filename contains the complete path, such as C:\MYDIR\MYLENS.TOL, then the specified file will be created. If the path is left off, then the <data>\Tolerance folder will be used (see " Folders "). See also LOADTOLERANCE .

10.2.14.142. SAVEWINDOW

Saves the text from any text window to a file.

Syntax:

```
SAVEWINDOW winnum, filename
```

Discussion:

The winnum value may be either an integer or an expression that evaluates to an integer. The integer winnum corresponds to the text window number that should be saved to a file. OpticStudio numbers windows sequentially as they are opened, starting with 1. Any closed windows are deleted from the window list, without renumbering the windows which remain. Any windows opened after another window has been closed will use the lowest window number available.

Example:

```
SAVEWINDOW 1, "C:\TEMP\TEXTFILE.TXT"
SAVEWINDOW 3, A$
```

10.2.14.143. SCATTER

Used to control whether sequential surface scattering is done while tracing rays.

Syntax:

```
SCATTER ON
SCATTER OFF
```

Discussion:

The default condition at the start of a macro is SCATTER OFF; and all rays will be traced deterministically. If SCATTER ON is executed, then sequential surface scattering will be enabled for all subsequent RAYTRACE commands.

10.2.14.144. SDIA

For setting surface properties use the " SETSURFACEPROPERTY, SURP " keyword, and for getting surface properties use the SPRO(surf, code) Numeric Function as discussed under " Numeric Functions ".

10.2.14.145. SETAIM

Sets the state of the ray aiming function.

Syntax:

```
SETAIM state
```

Discussion:

This keyword requires one numeric expression that must evaluate to either 0 or 1. The state is a code which is 0 for ray aiming off and 1 for ray aiming on.

Example:

```
SETAIM 1
```

Related Keywords:

SETAIMDATA

10.2.14.146. SETAIMDATA

Sets various data for the ray aiming function.

Syntax:

```
SETAIMDATA code, value
```

Discussion:

The code values are used as follows:

Code	Property
1	Sets "Use Ray Aiming Cache" to true if value is 1, or false if value is 0.
2	Sets "Robust Ray Aiming" to true if value is 1, or false if value is 0.
3	Sets "Scale Pupil Shift Factors by Field" to true if value is 1, or false if value is 0.
4, 5, 6	Sets the value of the x, y, and z pupil shift, respectively.
7, 8	Sets the value of the x and y pupil compress, respectively.

Example:

```
SETAIMDATA 5, 0.34
```

Related Keywords:

SETAIM

10.2.14.147. SETAPODIZATION

This command is obsolete. See "SETSYSTEMPROPERTY, SYSP".

10.2.14.148. SETCONFIG (keywords)

Sets the current configuration for multi-configuration (zoom) systems.

Syntax:

```
SETCONFIG config
```

Discussion:

The expression config must evaluate to an integer between 1 and the maximum number of configurations.

Example:

```
SETCONFIG 4
```

Related Functions:

CONF, NCON

10.2.14.149. SETDETECTOR

Sets the coherent or incoherent detector data for any pixel on a detector rectangle object.

Syntax:

```
SETDETECTOR surf, object, pixel, datatype, value
```

Discussion:

This keyword requires numeric expressions that specify the surface, object, pixel, and data type (all integers) and the new detector value. Surf is the surface number of the non-sequential group; use 1 when using non- sequential mode. Object is the object number of the detector. The object must be a detector rectangle. Pixel is a number between 1 and the number of pixels the object supports. Datatype should be 0 for incoherent intensity, 1 for incoherent intensity in angle space, 2 for coherent real part, 3 for coherent imaginary part, and 4 for coherent amplitude. For more information on the use and meaning of these values see " Detector Rectangle object ". The units for incoherent intensity are source units (see " Source Units "). The units for coherent amplitude are the square root of source units.

Example:

```
SETDETECTOR 1, 5, 12, 0, 1.0056
```

Related Functions:

NSDD

10.2.14.150. SETMCOPERAND

Sets any row or configuration of the Multi-Configuration Editor to any numeric value.

Syntax:

```
SETMCOPERAND row, config, value, datatype
```

Discussion:

This keyword requires numeric expressions that evaluate to integers specifying the row and configuration of the Multi-Configuration Editor.

If the config number is 0, then the value is interpreted as follows:

datatype = 0, value is a string literal or variable that specifies the name of the operand.

datatype = 1, 2, or 3, value is the number 1, 2, or 3 value used as part of the multi-configuration operand definition. See the description of the multi-configuration operand numbers defined in "Summary of multi- configuration operands".

If the config number corresponds to a defined configuration then the value is interpreted as follows:

datatype = 0, value is the value of the operand.

datatype = 1, value is the pickup offset of the operand.

datatype = 2, value is the pickup scale of the operand.

datatype = 3, value is the status of the operand, 0 for fixed, 1 for variable, 2 for pickup, 3 for thermal pickup.

datatype = 4, value is the pickup configuration number.

datatype = 5, value is the pickup row number.

Example:

```
SETMCOPERAND 3, 4, somevalue, 0
SETMCOPERAND 1, 0, "THIC", 0
```

Related Functions:

MCOP

10.2.14.151. SETNSCPARAMETER (keywords)

Sets the parameter values of any object in the NSC editor.

Syntax:

```
SETNSCPARAMETER surface, object, parameter, value
```

Discussion:

This keyword requires 3 numeric expressions that evaluate to integers specifying the non-sequential component surface number, the object number, and the parameter number. The fourth argument is the new value for the specified parameter.

Example:

```
SETNSCPARAMETER 4, 2, 15, newp15value
```

Related Functions:

NPOS, NPAR

Related Keywords:

INSERTOBJECT, SETNSCPOSITION, SETNSCPROPERTY

10.2.14.152. SETNSCPOSITION (keywords)

Sets the x, y, z or tilt x, tilt y, tilt z position of any object in the NSC editor.

Syntax:

```
SETNSCPOSITION surface, object, code, value
```

Discussion:

This keyword requires 3 numeric expressions that evaluate to integers specifying the non-sequential component surface number, the object number, and a code. The code is 1 through 6 for x, y, z, tilt x, tilt y, tilt z, respectively. The fourth argument is the new value for the specified position.

Example:

```
SETNSCPOSITION 4, 2, 2, newyvalue
```

Related Functions:

NPOS, NPAR

Related Keywords:

INSERTOBJECT, SETNSCPARAMETER, SETNSCPROPERTY

10.2.14.153. SETNSCPROPERTY (keywords)

Sets properties of NSC objects.

Syntax:

```
SETNSCPROPERTY surface, object, code, face, value
```

Discussion:

This keyword requires 4 numeric expressions that evaluate to integers specifying the non-sequential component surface number (use 1 if the program mode is non-sequential), the object number, a code which specifies what property of the object is being modified, and the face number (use 0 if not applicable to the property being set). The fifth argument is the new value for the specified property, and it may be either text in quotes, a string variable, or a numeric expression. The code is as follows:

Code	Property
The following codes set values on the NSC Editor.	
1	Sets the object comment.
2	Sets the reference object number.
3	Sets the "inside of" object number.
4	Sets the object material.
The following codes set values on the Type tab of the Object Properties dialog.	
0	Sets the object type. The value should be the name of the object, such as "NSC_SLEN" for the standard lens. The names for each object type are listed in the Prescription Report for each object type in the NSC editor. All NSC object names start with "NSC_".
13	Sets User Defined Aperture, use 1 for checked, 0 for unchecked.
14	Sets the User Defined Aperture file name.
15	Sets the "Use Global XYZ Rotation Order" checkbox, use 1 for checked, 0 for unchecked.
16	Sets the "Rays Ignore This Object" combo box, use 0 for Never, 1 for Always, and 2 for On Launch.
17	Sets the "Object Is A Detector" checkbox, use 1 for checked, 0 for unchecked.
18	Sets the "Consider Objects" list. The argument should be a string listing the object numbers to consider delimited by spaces, such as "2 5 14".
19	Sets the "Ignore Objects" list. The argument should be a string listing the object numbers to ignore delimited by spaces, such as "1 3 7".
20	Sets the "Use Pixel Interpolation" checkbox, use 1 for checked, 0 for unchecked.
30	Sets the "Use Consider/Ignore Objects When Splitting" checkbox, use 1 for checked, 0 for unchecked.
The following codes set values on the Coat/Scatter tab of the Object Properties dialog.	
5	Sets the coating name for the specified face.

6	Sets the profile name for the specified face.
7	Sets the scatter mode for the specified face: 0 = none, 1 = Lambertian, 2 = Gaussian, 3 = ABg, 4 = User Defined, 5 = BSDF, 6 = ABg File.
8	Sets the scatter fraction for the specified face.
9	Sets the number of scatter rays for the specified face.
10	Sets the Gaussian sigma (Gaussian scatter model) or the sample orientation angle (BSDF scatter model) for the specified face.
11	Sets the reflect ABg data name for the specified face.
12	Sets the transmit ABg data name for the specified face.
27	Sets the name of the user defined scattering DLL.
21-26	Sets parameter values on the user defined scattering DLL.
28	Sets the name of the user defined scattering data file.
29	Sets the "Face Is" property for the specified face. Use 0 for "Object Default", 1 for "Reflective", and 2 for "Absorbing".
31	Sets the reflect BSDF data file for the specified face. The value should be the name of the BSDF file with no path (i.e. BrownVinyl.bsdf).
32	Sets the transmit BSDF data file for the specified face. The value should be the name of the BSDF file with no path (i.e. BrownVinyl.bsdf).
33	Sets the reflect ABg File data file for the specified face. The value should be the name of the ABGF file with no path (e.g. SampleABGF.abgf).
34	Sets the transmit ABg File data file for the specified face. The value should be the name of the ABGF file with no path (e.g. SampleABGF.abgf).
37	Sets the Thin Window Scattering option for the specified face. Use 0 to turn the option off (i.e. unchecked option in checkbox) and 1 to turn the option on (i.e. checked option in checkbox).
The following codes set values on the Volume Physics tab of the Object Properties dialog.	
81	Sets the "Model" value on the bulk scattering tab. Use 0 for "No Bulk Scattering", 1 for "Angle Scattering", and 2 for "DLL Defined Scattering".
82	Sets the mean free path to use for bulk scattering.
83	Sets the angle to use for bulk scattering.
84	Sets the name of the DLL to use for bulk scattering.

85	Sets the parameter value to pass to the DLL, where the face value is used to specify which parameter is being defined. The first parameter is 1, the second is 2, etc.
86	Sets the wavelength shift string.
The following codes set values on the Diffraction tab of the Object Properties dialog.	
91	Sets the "Split" value on the diffraction tab. Use 0 for "Don't Split By Order", 1 for "Split By Table Below", and 2 for "Split By DLL Function".
92	Sets the name of the DLL to use for diffraction splitting.
93	Sets the Start Order value.
94	Sets the Stop Order value.
95, 96	Sets the parameter values on the diffraction tab. These are the parameters passed to the diffraction splitting DLL as well as the order efficiency values used by the "split by table below" option. The face value is used to specify which parameter is being defined. The first parameter is 1, the second is 2, etc. The code 95 is used for reflection properties, and 96 for transmission.
The following codes set values on the Sources tab of the Object Properties dialog.	
101	Sets the source object random polarization. Use 1 for checked, 0 for unchecked.
102	Sets the source object reverse rays option. Use 1 for checked, 0 for unchecked.
103	Sets the source object Jones X value.
104	Sets the source object Jones Y value.
105	Sets the source object Phase X value.
106	Sets the source object Phase Y value.
107	Sets the source object initial phase in degrees value.
108	Sets the source object coherence length value.
109	Sets the source object pre-propagation value.
110	Sets the source object sampling method; 0 for random, 1 for Sobol sampling.
111	Sets the source object bulk scatter method; 0 for many, 1 for once, 2 for never.
112	Sets the array mode; 0 for none, 1 for rectangular, 2 for circular, 3 for hexapolar, and 4 for hexagonal.
113	Sets the source color mode. For a complete list of the available modes, see "Defining the color and spectral content of sources". The source color modes are numbered starting with 0 for the System Wavelengths, and then from 1 through the last model listed in the dialog box control.
114-116	Sets the number of spectrum steps, start wavelength, and end wavelength, respectively.

117	Sets the name of the spectrum file.
161-162	Sets the array mode integer arguments 1 and 2.
165-166	Sets the array mode double precision arguments 1 and 2.
181-183	Sets the source color mode arguments, for example, the XYZ values of the Tristimulus.
The following codes set values on the Grin tab of the Object Properties dialog.	
121	Sets the "Use DLL Defined Grin Media" checkbox. Use 1 for checked, 0 for unchecked.
122	Sets the Maximum Step Size value.
123	Sets the DLL name.
124	Sets the Grin DLL parameters. These are the parameters passed to the DLL. The face value is used to specify which parameter is being defined. The first parameter is 1, the second is 2, etc.
The following codes set values on the Draw tab of the Object Properties dialog.	
141	Sets the do not draw object checkbox. Use 1 for checked, 0 for unchecked.
142	Sets the object opacity. Use 0 for 100%, 1 for 90%, 2 for 80%, etc.
143	Sets the drawing resolution for the object. Use 0 for Standard, 1 for Medium, 2 for High, 3 for Presentation, and 4 for Custom.
144	Sets the value of the first resolution input to the drawing resolution (e.g. "Angular" resolution for the Annular Aspheric Lens object) when the drawing resolution itself is set to Custom. The drawing resolution must first be set to Custom before this can be used.
145	Sets the value of the second resolution input to the drawing resolution (e.g. "Radial" resolution for the Annular Aspheric Lens object) when the drawing resolution itself is set to Custom. The drawing resolution must first be set to Custom before this can be used.
The following codes set values on the Scatter To tab of the Object Properties dialog.	
151	Sets the scatter to method. Use 0 for scatter to list, and 1 for importance sampling.
152	Sets the Importance Sampling target data. The argument should be a string listing the ray number, the object number, the size, and the limit value, all separated by spaces Here is a sample syntax to set the Importance Sampling data for ray 3, object 6, size 3.5, and limit 0.6: "3 6 3.5 0.6".
153	Sets the "Scatter To List" values. The argument should be a string listing the object numbers to scatter to delimited by spaces, such as "4 6 19".
The following codes set values on the Birefringence tab of the Object Properties dialog.	

171	Sets the Birefringent Media checkbox. Use 0 for unchecked, and 1 for checked.
172	Sets the Birefring ent Media Mode. Use 0 for Trace ordinary and extraordinary rays, 1 for Trace only ordinary rays, 2 for Trace only extraordinary rays, and 3 for Waveplate mode.
173	Sets the Birefringent Media Reflections status. Use 0 for Trace reflected and refracted rays, 1 for Trace only refracted rays, and 2 for Trace only reflected rays.
174-176	Sets the Ax, Ay, and Az values.
177	Sets the Axis Length.
The following codes do not set values, but are included here to return values for the function NPRO.	
200	Used by function NPRO to determine the index of refraction of an object. The syntax is NPRO(surface, object, 200, wavenumber)
201-203	Used by function NPRO to determine the nd (201), vd (202), and dpgf (203) parameters of an object using a model glass. The syntax is NPRO(surface, object, 201, 0)

The NSC Object type codes are listed below. These codes are not case sensitive.

NSC Object	Object type codes
Annular Aspheric Lens	NSC_AASL
Annular Axial Lens	NSC_AAXL
Annular Volume	NSC_AVOL
Annulus	NSC_ANNU
Array	NSC_ARRA
Array Ring	NSC_ARRR
Aspheric Surface	NSC_ASUR
Aspheric Surface 2	NSC_SSU2
Axicon Surface	NSC_AXC1
Biconic Lens	NSC_BLEN
Biconic Surface	NSC_BSUR
Biconic Zernike Lens	NSC_BZER
Biconic Zernike Surface	NSC_BZSR
Binary 1	NSC_BIN1
Binary 2	NSC_BIN2
Binary 2A	NSC_BN2A

Boolean CAD	NSC_COMB
CAD PartSTEP/IGES/SAT	NSC_IMPT
CAD PartSTL	NSC_STLO
CAD Part Zemax Part Designer	NSC_ZPDO
Cone	NSC_CONE
CPC	NSC_CPCO
CPC Rectangular	NSC_CPCR
Cylinder 2 Pipe	NSC_CPP2
Cylinder 2 Volume	NSC_CBL2
Cylinder Pipe	NSC_CPIP
Cylinder Volume	NSC_CBLK
Detector Color	NSC_DETC
Detector Polar	NSC_DETP
Detector Rectangle	NSC_DETE
Detector Surface	NSC_DETS
Detector Volume	NSC_DETV
Diffraction Grating	NSC_DGRL
Dual BEF Surface	NSC_DBEF
Ellipse	NSC_ELLI
Elliptical Volume	NSC_ELIV
Even Asphere Lens	NSC_EVAS
Ext. Poly. Lens	NSC_XPLE
Ext. Poly. Surface	NSC_XPSU
Extended Odd Asphere Lens	NSC_XOAS
Extruded	NSC_EXTR
Faceted Surface	NSC_FSUR
Freeform Z	NSC_SPLZ
Fresnel 1	NSC_FRES
Fresnel 2	NSC_FRE2
Hexagonal Lenslet Array	NSC_HEX1
Hologram Lens	NSC_HOLO
Hologram Surface	NSC_HOLS
Jones Matrix	NSC_JONE
Lenslet Array 1	NSC_LET1
Lenslet Array 2	NSC_LET2

MEMS	NSC_DMDT
Odd Asphere Lens	NSC_OVAS
Paraxial Lens	NSC_PARL
Polygon Object	NSC_POBJ
Ray Rotator	NSC_ROTA
Rect. Torus Surface	NSC_RTSU
Rect. Torus Volume	NSC_RTVO
Rectangle	NSC_RECT
Rectangular Corner	NSC_RCOR
Rectangular Pipe	NSC_RPIP
Rectangular Pipe Grating	NSC_RPIG
Rectangular Roof	NSC_RROF
Rectangular Volume	NSC_RBLK
Rectangular Volume Grating	NSC_RBLG
Slide	NSC_SLID
Source Diffractive	NSC_SDIF
Source Diode	NSC_SDIO
Source DLL	NSC_SDLL
Source Ellipse	NSC_SRCE
Source EULUMDAT File	NSC_SEUL
Source Filament	NSC_SHLX
Source File	NSC_SFIL
Source Gaussian	NSC_SGAU
Source IESNA File	NSC_SIES
Source Imported	NSC_SIMP
Source Object	NSC_SOBJ
Source Point	NSC_SRCP
Source Radial	NSC_SRAD
Source Ray	NSC_SRAY
Source Rectangle	NSC_SRCR
Source Tube	NSC_STUB
Source Two Angle	NSC_SR2A
Source Volume Cylinder	NSC_VSRC
Source Volume Ellipse	NSC_VSRE
Source Volume Rectangle	NSC_VSRR

Sphere	NSC_SPHE
Standard Lens	NSC_SLEN
Standard Surface	NSC_SSUR
Swept	NSC_SWEE
Tab. Faceted Radial	NSC_TFRA
Tab. Faceted Toroid	NSC_TFTO
Tab. Fresnel Radial	NSC_TFRR
Toroidal Hologram	NSC_THOL
Toroidal Lens	NSC_TLEN
Toroidal Surface	NSC_TSUR
Toroidal Surface Odd Asphere	NSC_TSOA
Torus Surface	NSC_TFHO
Torus Volume	NSC_TFSO
Triangle	NSC_TRIA
Triangular Corner	NSC_TCOR
User Defined Object	NSC_UOBJ
Wolter Surface	NSC_WSUR
Zernike Surface	NSC_ZSUR

Example:

```
SETNSCPROPERTY 1, 2, 0, 0, "NSC_SLEN"
```

Related Functions:

NPOS, NPAR, NPRO

Related Keywords:

INSERTOBJECT, SETNSCPARAMETER, SETNSCPOSITION

10.2.14.154. SETOPERAND (keywords)

Sets any row or column of the Merit Function Editor to any numeric value.

Syntax:

```
SETOPERAND row, col, value
```

Discussion:

This keyword requires numeric expressions that evaluate to integers specifying the row and column of the Merit Function Editor.

The col integer is:

- 1 for operand type,

- 2 for int1,
- 3 for int2,
- 4-7 for data1-data4,
- 8 for target, and
- 9 for weight.

To set the comment string associated with an operand use col 10.

To set the operand type, use a col value of 1 and the integer operand returned by the function ONUM. An alternate method for setting the operand type is to use a col value of 11 with the string name of the operand type, such as EFFL or DIST.

If col is 10 or 11, value should be a string constant or variable.

The col integer is 12 for data5 and 13 for data6. Note the value and percent contribution columns cannot be set but must be computed.

Example:

```
SETOPERAND 1, 8, tarvalue
SETOPERAND 3, 11, "EFFL"
SETOPERAND 5, 10, "Operand Number 5"
```

Related Functions:

MFCN, OPER, ONUM

10.2.14.155. SETSTDD

This command is obsolete. See " SETSURFACEPROPERTY, SURP ".

10.2.14.156. SETSURFACEPROPERTY, SURP

Sets properties of surfaces.

Syntax:

```
SETSURFACEPROPERTY surface, code, value1, value2
SURP surface, code, value1, value2
```

Discussion:

Surface is an expression that evaluates to an integer specifying the surface number. The code may either be an expression that evaluates to an integer or a mnemonic which specifies what property of the surface is being modified. The third and fourth arguments are the new values for the specified property, and they may be either text in quotes, a string variable, or a numeric expression, depending upon the code. For most codes, the property value being modified is defined by the value1 argument. A few operands require both a value1 and a value2, as described in the table below.

If the property being modified is under control of the Multi-Configuration Editor, then the multi-configuration data for the current configuration only will also be modified to reflect the changed property.

To set a surface as the stop surface, see "STOPSURF".

SURP is a shorthand for SETSURFACEPROPERTY and is functionally identical.

Code	Property
Basic Surface Data. See "Lens Data.	

0 or TYPE	Surface type. The value should be the name of the object, such as "STANDARD" for the standard surface. The names for each surface type are listed in the Prescription Report in the Surface Data Summary for each surface type currently in the Lens Data Editor. See also "Surface names" summary table at the bottom of this page. To change the surface type to a user defined surface, first set the DLL name using code 9 (SDLL), then set the new surface type to USERSURF. See also Code 17.
1 or COMM	Comment.
2 or CURV	Curvature (not radius) in inverse lens units. Use zero for an infinite radius.
3 or THIC	Thickness in lens units.
4 or GLAS	Glass name. See also Code 18.
5 or CONI	Conic constant.
6 or SDIA	Clear semi-diameter or Semi-diameter. If the value is zero or positive, the clear semi-diameter or semi-diameter solve is set to "Fixed". If the value is negative, the clear semi-diameter or semi-diameter solve is set to "Automatic" and the clear semi-diameter or semi-diameter will be computed with the next UPDATE keyword.
7 or TCE	Thermal coefficient of expansion.
8 or COAT	Coating name. Use a blank string for value1 to remove the coating.
9 or SDLL	User defined surface DLL name.
10 or PARM	Parameter value. Value1 is the new value. Value2 is the parameter number.
11 or EDVA	Extra Data value. Value1 is the new value. Value2 is the extra data number.
12	Surface color, Use 0 for default.
13	Surface opacity.
14	Row color.
15	Surface cannot be hyperhemispheric. Use 1 to avoid surface being hyperhemispheric.
16	Ignore surface. Use 1 to ignore surface, 0 to not ignore surface.
17 or CODE	The integer code for the surface type. The integer code is an alternative to the surface name used by code 0. For full details see the documentation for code 0 above.
18 or GLAN	Glass number. See also Code 4.
Surface aperture data. See "Surface properties aperture tab".	
20 or ATYP	Surface aperture type code.
21 or APP1	Surface aperture parameter 1.
22 or APP2	Surface aperture parameter 2.

23 or APDX	Surface aperture decenter x.
24 or APDY	Surface aperture decenter y.
25 or UDA	User Defined Aperture (UDA) file name.
26 or APPU	Surface aperture pick up from surface number. Use 0 for no pickup.
27 or CHZN	Chip Zone of surface.
28 or MCSD	Mechanical Semi-Diameter. If the value is zero or positive, the mechanical semi-diameter solve is set to "Fixed". If the value is negative, the mechanical semi-diameter solve is set to "Automatic" and it will be computed with the next UPDATE keyword.
Physical Optics Propagation Settings. See "Surface specific settings".	
30	Physical Optics setting "Use Rays To Propagate To Next Surface". Use 1 for true, 0 for false.
31	Physical Optics setting "Do Not Rescale Beam Size Using Ray Data". Use 1 for true, 0 for false.
32	Physical Optics setting "Use Angular Spectrum Propagator". Use 1 for true, 0 for false.
33	Physical Optics setting "Draw ZBF On Shaded Model". Use 1 for true, 0 for false.
34	Physical Optics setting "Recompute Pilot Beam Parameters". Use 1 for true, 0 for false.
35	Physical Optics setting "Resample After Refraction". Use 1 for true, 0 for false.
36	Physical Optics setting "Auto Resample". Use 1 for true, 0 for false.
37	Physical Optics setting "New X Sampling". Use 1 for 32, 2 for 64, etc.
38	Physical Optics setting "New Y Sampling". Use 1 for 32, 2 for 64, etc.
39	Physical Optics setting "New X-Width". New total x direction width of array.
40	Physical Optics setting "New Y-Width". New total y direction width of array.
41	Physical Optics setting "Output Pilot Radius". Use 0 for best fit, 1 for shorter, 2 for longer, 3 for x, 4 for y, 5 for plane, 6 for user.
42, 43	Physical Optics setting "X-Radius" and "Y-Radius", respectively.
44	Physical Optics setting "Use X-axis Reference". Use 1 for true, 0 for false.
Coating Settings. See "Surface coating tab". See also code 8 above.	
50	Use Layer Multipliers and Index Offsets. Use 1 for true, 0 for false.
51	Layer Multiplier value. Value1 is the new value. Value2 is the layer number.
52	Layer Multiplier status. Value 1 is the status, use 0 for fixed, 1 for variable, or n+1 for pickup from layer n. Value2 is the layer number.

53	Layer Index Offset value. Value1 is the new value. Value2 is the layer number.
54	Layer Index Offset status. Value 1 is the status, use 0 for fixed, 1 for variable, or n+1 for pickup from layer n. Value2 is the layer number.
55	Layer Extinction Offset value. Value1 is the new value. Value2 is the layer number.
56	Layer Extinction Offset status. Value 1 is the status, use 0 for fixed, 1 for variable, or n+1 for pickup from layer n. Value2 is the layer number.
Surface Tilt and Decenter Data. See "Surface tilt/decenter tab".	
60 or BOR	Before tilt and decenter order. Use 0 for dec/tilt, 1 for tilt/dec.
61 or BDX	Before decenter x.
62 or BDY	Before decenter y.
63 or BTX	Before tilt about x.
64 or BTY	Before tilt about y.
65 or BTZ	Before tilt about z.
66 or APU	After pick up status: 0 for explicit, 1/2 for pickup/reverse current surface, 3/4 for pickup/reverse current surface minus 1, 5/6 for pickup/reverse current surface minus 2, etc...
70 or AOR	After tilt and decenter order. Use 0 for dec/tilt, 1 for tilt/dec.
71 or ADX	After decenter x.
72 or ADY	After decenter y.
73 or ATX	After tilt about x.
74 or ATY	After tilt about y.
75 or ATZ	After tilt about z.
76	Coordinate Return status. Valid only on Coordinate Break surfaces. Use 0 for None, 1 for Orientation Only, 2 for Orientation XY, and 3 for Orientation XYZ.
77	Coordinate Return To Surface. Valid only on Coordinate Break surfaces.
Surface scatter data. See "Surface properties scattering tab".	
80	Sets the scatter code: 0 for none, 1 for Lambertian, 2 for Gaussian, 3 for ABg, 4 for DLL, 5 for BSDF, and 6 for ABg File.
81	Sets the scatter fraction, should be between 0.0 and 1.0.
82	Sets the Gaussian scatter sigma.
83	Sets the ABg file name.
84	Sets the name of the user defined scattering DLL. To set the parameters see Code 181.
85	Sets the name of the data file used by the user defined scattering DLL.

86	Sets the BSDF file name. The value should be the name of the BSDF file with no path (i.e. BrownVinyl.bsdf).
87	Sets the ABg File data file name. The value should be the name of the ABGF file with no path (e.g. SampleABGF.abgf).
Surface draw data. See "Surface properties draw tab".	
90	Sets the "Hide Rays To This Surface" checkbox status: 0 for off, 1 for on.
91	Sets the "Skip Rays To This Surface" checkbox status: 0 for off, 1 for on.
92	Sets the "Do Not Draw This Surface" checkbox status: 0 for off, 1 for on.
93	Sets the "Do Not Draw Edges From This Surface" checkbox status: 0 for off, 1 for on.
96	Sets the "Draw Edges As" status: 0 for squared, 1 for tapered, 2 for flat.
97	Sets "Mirror Substrate" status: 0 for none, 1 for flat, 2 for curved.
98	Sets the mirror substrate thickness value.
User defined surface scatter DLL parameters. See "Surface properties scattering tab".	
181-186	Sets the user defined scatter DLL parameters 1-6.

Changes to surface properties usually require that a subsequent UPDATE (see "UPDATE") be performed to update pupil positions, solves, and other data required for correct ray tracing. The UPDATE is not performed automatically because it is faster to execute all SETSURFACEPROPERTY keywords and then execute just one UPDATE. The function SPRO uses a very similar syntax and identical code values to "get" rather than "set" these same values.

Example:

```
! Set the glass type on surface 7 to BK7
SETSURFACEPROPERTY 7, GLAS, "BK7"
! Set the thickness of surface 2 to the thickness of surface 1
SETSURFACEPROPERTY 2, THIC, THIC(1)
! Set the value of parameter 4 on surface 11 to 7.3
SURP 11, PARM, 7.3, 4
```

Surface Type	Name
Standard	STANDARD
Even Asphere	EVENASPH
Odd Asphere	ODDASPHE
Paraxial	PARAXIAL
Paraxial XY	PARAX_XY
Biconic	BICONICX
Toroidal Grating	TOROGRAT
Cubic Spline	CUSPLINE

Hologram 1	HOLOGRM1
Hologram 2	HOLOGRM2
Coordinate Break	COORDBRK
Polynomial	POLYNOMI
Fresnel	FRESNELS
ABCD	ABCDSURF
Alternate Even	ALTERNAT
Alternate Odd	ALTERNAO
Diffraction Grating	DGRATING
Conjugate	CONJUGAT
Tilted	TILTSURF
Irregular	IRREGULA
Gradient 1	GRINSUR1
Gradient 2	GRINSUR2
Gradient 3	GRINSUR3
Gradient 4	GRINSUR4
Gradient 5	GRINSUR5
Gradient 6	GRINSUR6
Gradient 7	GRINSUR7
Gradium	GRINSUR8
Gradient 9	GRINSUR9
Gradient 10	GRINSU10
Gradient 12	GRINSU12
Retro Reflect	RETROREF
Toroidal	TOROIDAL
Zernike Fringe Sag	FZERNSAG
Zernike Fringe Phase	FZERNPHA
Zernike Standard Sag	SZERNSAG
Zernike Standard Phase	SZERNPHA
Zernike Annular Phase	AZERNPHA
Biconic Zernike	BICONICZ
Extended Polynomial	XPOLYNOM
Binary Optic 1	BINARY_1
Binary Optic 2	BINARY_2
Binary Optic 3	BINARY_3

Binary Optic 4	BINARY_4
Extended Cubic Spline	XCUSPLIN
Extended Asphere	XASPHERE
Extended Odd Asphere	XOSPHERE
Variable Line Space Grating	VARLSGRT
Elliptical Grating 1	ELLIGRAT
Elliptical Grating 2	ELLIGRA2
Superconic	SUPERCON
Extended Fresnel	XFRESNEL
Cylinder Fresnel	CFRESNEL
Grid Sag	GRID_SAG
Grid Phase	GRID_PHA
Generalized Fresnel	GEN_FRES
Periodic	PERIODIC
Toroidal Hologram	TOROHOLO
Atmospheric Refraction	ATMOSPHR
Zone Plate	ZONEPLAT
Q-Type Asphere	QED_TYPE
Jones Matrix	JONESMAT
Zernike Annular Standard Sag	AZERNSAG
Chebyshev Polynomail	CHEBYSHV
Black Box Lens	BLACKBOX
Slide	SLIDESRF
User defined	USERSURF
Birefringent In	BIRE__IN
Birefringent Out	BIRE_OUT
Optically Fabricated Hologram	OFABHOL1
Radial NURBS	RADNURBS
Toroidal NURBS	TORNURBS
Extended Toroidal Grating	TOROGRAX
Radial Grating	RGRATING
Odd Cosine	ODDCOSIN
Non-Sequential Component	NONSEQCO
Data	DATASURF
Off-Axis Conic Freeform	OFFAXISCF

Q-Type Freeform	QFREEFORM

Related Functions:

SPRO

Related Keywords:

SETSYSTEMPROPERTY, UPDATE

10.2.14.157. SETSYSTEMPROPERTY, SYSP

Sets properties of the system, such as system aperture, field, wavelength, and other data.

Syntax:

```
SETSYSTEMPROPERTY code, value1, value2
SYSP code, value1, value2
```

Discussion:

This keyword requires a numeric expression that evaluates to an integer code which specifies what property is being modified. The second and third arguments are the new values for the specified property, and they may be either text in quotes, a string variable, or a numeric expression, depending upon the code. For most codes, the property value being modified is defined by the value1 argument. A few operands require both a value1 and a value2, as described in the table below.

If the property being modified is under control of the Multi-Configuration Editor, then the multi-configuration data for the current configuration only will also be modified to reflect the changed property.

SYSP is a shorthand for SETSYSTEMPROPERTY and is functionally identical.

Code	**Property**
4	Adjust Index Data To Environment. Use 0 for off, 1 for on. See "Adjust Index Data To Environment".
10	Aperture Type code. See "Aperture Type" for code values and discussion.
11	Aperture Value. See "Aperture Value".
12	Apodization Type code. Use 0 for uniform, 1 for Gaussian, 2 for cosine cubed. See "Apodization Type".
13	Apodization Factor. See "Apodization Factor".
14	Telecentric Object Space. Use 0 for off, 1 for on. See "Telecentric Object Space".
15	Iterate Solves When Updating. Use 0 for off, 1 for on. See "Iterate Solves When Updating".
16	Lens Title. See "Lens Title".
17	Lens Notes. See "Notes".
18	Afocal Image Space. Use 0 for off, 1 for on. See "Afocal Image Space".
21	Global coordinate reference surface. See "Global Coordinate Reference Surface".

23	Glass catalog list. Use a string or string variable with the glass catalog name, such as "SCHOTT". To specify multiple catalogs use a single string or string variable containing names separated by spaces, such as "SCHOTT HOYA OHARA".
24	System Temperature in degrees Celsius. See "Temperature in degrees C".
25	System Pressure in atmospheres. See "Pressure in ATM".
26	Reference OPD method. Use 0 for absolute, 1 for infinity, 2 for exit pupil, and 3 for absolute 2. See "Reference OPD".
30	Lens Units code. Use 0 for mm, 1 for cm, 2 for inches, or 3 for Meters. Changing lens units does not scale or convert the lens data in any way, it only changes how the lens prescription data is interpreted. See "Lens Units".
31	Source Units Prefix. Use 0 for Femto, 1 for Pico, 2 for Nano, 3 for Micro, 4 for Milli, 5 for None, 6 for Kilo, 7 for Mega, 8 for Giga, and 9 for Tera. See "Source Units".
32	Source Units. Use 0 for Watts, 1 for Lumens, and 2 for Joules. See "Source Units".
33	Analysis Units Prefix. Use 0 for Femto, 1 for Pico, 2 for Nano, 3 for Micro, 4 for Milli, 5 for None, 6 for Kilo, 7 for Mega, 8 for Giga, and 9 for Tera. See "Analysis Units".
34	Analysis Units "per" Area. Use 0 for square mm, 1 for square cm, 2 for square inches, 3 for square Meters, and 4 for square feet.
35	MTF Units code. Use 0 for cycles per millimeter, or 1 for cycles per milliradian. See "MTF Units".
40	Coating File name. See "Coating File".
41	Scatter Profile name. See "Scatter Profile".
42	ABg Data File name. See "ABg Data File".
43	GRADIUM Profile name. See "GRADIUM Profile".
50	NSC Maximum Intersections Per Ray. See "Maximum Intersections Per Ray" .
51	NSC Maximum Segments Per Ray. See "Maximum Segments Per Ray".
52	NSC Maximum Nested/Touching Objects. See "Maximum Nested/Touching Objects".
53	NSC Minimum Relative Ray Intensity. See "Minimum Relative Ray Intensity".
54	NSC Minimum Absolute Ray Intensity. See "Minimum Absolute Ray Intensity".

55	NSC Glue Distance In Lens Units. See "Glue Distance In Lens Units".
56	NSC Missed Ray Draw Distance In Lens Units. See "Missed Ray Draw Distance in Lens Units".
57	NSC Retrace Source Rays Upon File Open. Use 0 for no, 1 for yes. See "Retrace Source Rays Upon File Open".
58	NSC Maximum Source File Rays In Memory. See "Maximum Source File Rays In Memory".
59	Simple Ray Splitting. Use 0 for no, 1 for yes. See "Simple Ray Splitting".
60	Polarization Jx. See "Jx, Jy, X-Phase, Y-Phase".
61	Polarization Jy. See "Jx, Jy, X-Phase, Y-Phase".
62	Polarization X-Phase. See "Jx, Jy, X-Phase, Y-Phase".
63	Polarization Y-Phase. See "Jx, Jy, X-Phase, Y-Phase".
64	Convert thin film phase to ray equivalent. Use 0 for no, 1 for yes. See "Convert thin film phase to ray equivalent".
65	Unpolarized. Use 0 for no, 1 for yes. See "Unpolarized".
66	Method. Use 0 for X-axis, 1 for Y-axis, and 2 for Z-axis. See "Method".
70	Ray Aiming. Use 0 for off, 1 for on, 2 for aberrated. See "Ray Aiming".
71, 72, 73	Ray aiming pupil shift x, y, and z. See "Pupil Shift, Pupil Compress".
74	Use Ray Aiming Cache. Use 0 for no, 1 for yes. See "Use Ray Aiming Cache".
75	Robust Ray Aiming. Use 0 for no, 1 for yes. See "Robust Ray Aiming (slow)".
76	Scale Pupil Shift Factors By Field. Use 0 for no, 1 for yes. See "Pupil Shift, Pupil Compress".
77, 78	Ray aiming pupil compress x, y. See "Pupil Shift, Pupil Compress".
100	Field type code. See "Field Type".
101	Number of fields.
102, 103	The field number is value1, value2 is the field x, y coordinate.
104	The field number is value1, value2 is the field weight.
105, 106	The field number is value1, value2 is the field vignetting decenter x, decenter y
107, 108	The field number is value1, value2 is the field tangential compression x, compression y

109	The field number is value1, value2 is the field tangential angle.
110	The field normalization method, value 1 is 0 for radial and 1 for rectangular.
200	Primary wavelength number. See "Wavelengths".
201	Number of wavelengths
202	The wavelength number is value1, value 2 is the wavelength in micrometers.
203	The wavelength number is value1, value 2 is the wavelength weight
901	The number of CPU's to use in multi-threaded computations, such as optimization. If the passed value is zero, the number of CPU's will be set to the default value. When testing this value using the function SYPR, this returns the total number of CPU's available as reported by the operating system.

Usually, changes to system properties do not become effective until after the UPDATE keyword is executed.

Example:

```
! Set the number of wavelengths to 3
SETSYSTEMPROPERTY 201, 3
! Set the number of fields to 4
SYSP 101, 4
```

Related Functions:

SYPR

Related Keywords:

SETSURFACEPROPERTY, UPDATE

10.2.14.158. SETTEXTSIZE

Changes the size of the characters drawn by the GTEXT command.

Syntax:

```
SETTEXTSIZE xsize, ysize
```

Discussion:

The arguments refer to the fraction of the graphic screen width that each character represents. For example, the default text size is 70 40. This means each character is 1/70 of the graphic screen width, and 1/40 of the screen height. An argument of zero restores the text size to the default.

Example:

```
! Make text twice default size
SETTEXTSIZE 35, 20
! Restore text size to default
SETTEXTSIZE
```

10.2.14.159. SETTITLE

This command is obsolete. See " SETSYSTEMPROPERTY, SYSP ".

10.2.14.160. SETTOL (keywords)

Sets the data in the various columns for any tolerance operand in the tolerance data editor.

Syntax:

```
SETTOL row, column, data
```

Discussion:

The value row must evaluate to an integer expression greater than 0 and less than or equal to the current number of tolerance operands. To set the operand type. use a column value of 0. The data is either a literal string or a string variable that contains the name of the operand, such as "TTHI". To set the integer 1, 2, or 3 values, use a column number of 1, 2, or 3. To set the min or max tolerance values, use column 4 or 5, respectively.

To set the "Do Not Adjust During Inverse Tolerancing" or the "Ignore this Operand During Tolerancing" checkboxes, use column 6 or 7 respectively.

The data is 1 for checked, 0 for unchecked. To set the comment, use column 99. The data is either a literal string or a string variable that contains the comment. The nominal value cannot be set using SETTOL.

See also INSERTTOL and DELETETOL .

Example:

```
SETTOL 16, 0, "TRAD"
```

Related Functions:

TOLV, $TOLOPERAND, $TOLCOMMENT

10.2.14.161. SETUNITS

This command is obsolete. See " SETSYSTEMPROPERTY, SYSP ".

10.2.14.162. SETVAR

Changes the state of variables for optimization.

Syntax:

```
SETVAR surf, code, status, object
or
SETVAR config, M, status, operand
```

Discussion:

Surf must evaluate to an integer between 0 and the maximum number of surfaces. Config must evaluate to an integer between 1 and the number of configurations. Code must be one of the following strings:

R for radius of curvature

T for thickness

C for conic

G for glass

I for glass index

J for glass Abbe

K for glass dpgf

Pn for parameter n

D for thermal coefficient of expansion

En for extra data value n

M for multi-configuration data, see discussion below

Nn for non-sequential component position data, 1-6 for x, y, z, tx, ty, tz

On for non-sequential component parameter data, where n is the parameter number

If status evaluates to 0, then the variable status is removed. Otherwise, the value is made variable. If the code is Nn or On; the object number must be provided; otherwise, it should be omitted. If the code is M, then the syntax for this command is as shown under "syntax" above.

Examples:

```
SETVAR j+3, R, 1
SETVAR 5, P6, 0
SETVAR surfk+2, E06, status
SETVAR config, M, status, operand
SETVAR 1, O32, 1, 5
```

10.2.14.163. SETVECSIZE

Changes the maximum size of the VEC1, VEC2, VEC3, and VEC4 arrays.

Syntax:

```
SETVECSIZE n
```

Discussion:

The expression must evaluate to an integer argument between 1 and 18,000,000. All four vector variables are always the same size. All data in the vectors will be lost during the resize. The initial size of the vectors is 1000.

10.2.14.164. SETVIG (keywords)

Sets the vignetting factors for the lens.

Syntax:

```
SETVIG
```

Discussion:

See " Vignetting factors " for a description of vignetting factors.

10.2.14.165. SHOWBITMAP

Displays a BMP, JPG or PNG file in a viewer window.

Syntax:

```
SHOWBITMAP filename
```

Discussion:

This keyword requires the name of the BMP, JPG or PNG file. The extension must be included. The filename may be enclosed in quotes if any blank or other special characters are used. The file must be located in the <data>\<images> folder. This command will open a new window to display the file.

Example:

```
SHOWBITMAP "BARCHART.BMP"
```

10.2.14.166. SHOWFILE

Displays a text file to the screen using the OpticStudio file viewer.

Syntax:

```
SHOWFILE filename, saveflag
```

Discussion:

The filename must be a valid file name. The file must be a text file (as would be created by OUTPUT and PRINT commands in ZPL) and must be in the current folder. Once the file is displayed, it may be scrolled up and down and printed like any other text file. The ability to scroll and print the data is the primary advantage of using OUTPUT and SHOWFILE instead of PRINT commands. SHOWFILE also closes the file if no CLOSE command has been executed. If the saveflag is zero or omitted, then the file is erased when the window is closed. if saveflag is any value other than zero, then the file remains even after the window is closed.

Example:

```
OUTPUT "test.txt"
PRINT "Print this to a file."
SHOWFILE "test.txt"
```

Related Keywords:

OPEN, OUTPUT, CLOSE, PRINT, PRINTFILE

10.2.14.167. SOLVEBEFORESTOP

Enables ZPL solves to be placed on surfaces prior to the stop surface.

Syntax:

```
SOLVEBEFORESTOP
```

Discussion:

This keyword is used only for supporting ZPL Macro solves. It must be placed on the first line of the macro in order to prevent OpticStudio from issuing an error message. Usage of this keyword implies that the user understands when such solves are valid and when they are not.

Any solve placed prior to the stop should not be based upon ray data. For example, the RAYTRACE keyword and data that it returns should not be used. OpticStudio doesn't perform any checks to ensure this is the case. If the macro does contain ray-based data then the macro solve may compute erroneous values.

See " Using ZPL Macro solves ".

10.2.14.168. SOLVERETURN

Returns a computed solve value back to the editor calling the solve.

Syntax:

```
SOLVERETURN x
```

Discussion:

This keyword is used only for supporting ZPL Macro solves. The macro computes a single value, and returns the value to the editor calling the solve using this keyword. If more than one SOLVERETURN keyword exists, only the last SOLVERETURN called is considered and prior calls are ignored.

If an error or invalid condition occurs in the macro, and the solve value cannot be computed, then the macro should return without calling SOLVERETURN. The absence of the SOLVERETURN call will indicate to OpticStudio that the solve cannot be computed and the optical system is in an error condition. This is particularly important during optimization.

See " Using ZPL Macro solves ".

10.2.14.169. SOLVETYPE

Changes the solve status on a given surface and value. Only some solve types are supported; contact OpticStudio technical support for information on setting other types of solves.

Syntax:

```
SOLVETYPE surf, CODE, arg1, arg2, arg3, arg4
```

Discussion:

Surf must evaluate to an integer argument between 0 and the maximum number of surfaces. The CODE must be a mnemonic as listed in the following table. The arg1 - arg4 expressions evaluate to the first - fourth solve parameters as specified in "SOLVES". Note that for cross-column Pickup solves, the column numbers are defined in " Integer codes for column numbers ". For non-sequential pickup solves, the arguments are the first through fourth lines that follow the "Solve Type" selection on the NSC solve dialog box. Some codes do not use any or all of the arguments and arg2/arg3/arg4 may be omitted in these cases.; arg1 is always required with a value of 0 if it does not exist via the GUI menu.

CODES FOR SOLVETYPE KEYWORD

Solve Type	**CODE**
Curvature: Fixed (turn solve off)	CF
Curvature: Variable	CV
Curvature: Marginal Ray	CM
Curvature: Chief Ray	CC
Curvature: Pickup	CP
Curvature: Marginal Ray Normal	CN
Curvature: Chief Ray Normal	CO
Curvature: Aplanatic	CA
Curvature: Element Power	CE

Curvature: Concentric With Surface	CQ
Curvature: Concentric With Radius	CR
Curvature: F/#	CG
Curvature: ZPL Macro	CZ
Thickness: Fixed (turn solve off)	TF
Thickness: Variable	TV
Thickness: Marginal Ray Height	TM
Thickness: Chief Ray Height	TC
Thickness: Edge Thickness	TE
Thickness: Pickup	TP
Thickness: Optical Path Difference	TO
Thickness: Position	TL
Thickness: Compensator	TX
Thickness: Center Of Curvature	TY
Thickness: Pupil Position	TU
Thickness: ZPL Macro	TZ
Glass: Fixed (turn solve off)	GF
Glass: Model	GM
Glass: Pickup	GP
Glass: Substitute	GS
Glass: Offset	GO
Clear semi-diameter or Semi-Diameter: Automatic	SA
Clear semi-diameter or Semi-Diameter: User Defined	SU
Clear semi-diameter or Semi-Diameter: Pickup	SP
Clear semi-diameter or Semi-Diameter: Maximum	SM
Clear semi-diameter or Semi-Diameter: ZPL Macro	SZ
Conic: Fixed (turn solve off)	KF
Conic: Pickup	KP
Conic: ZPL Macro	KZ
Mech Semi-Dia: Automatic	XA
Mech Semi-Dia: Fixed (turn solve off)	XU
Mech Semi-Dia: Pickup	XK
Mech Semi-Dia: ZPL Macro	XM
Chip Zone: Automatic	OU
Chip Zone: Pickup	OP

Chip Zone: ZPL Macro	OZ
Parameter: Fixed (turn solve off). Replace "p" with the parameter number in the code, for example, PF_3 will turn off the solve on parameter 3.	PF_p
Parameter: Pickup. Replace "p" with the parameter number in the code, for example, PP_4 will set the solve on parameter 4.	PP_p
Parameter: Chief Ray. Replace "p" with the parameter number in the code, for example, PC_1 will set the solve on parameter 1.	PC_p
Parameter: Variable. Replace "p" with the parameter number in the code, for example, PV_1 will set the solve on parameter 1.	PV_p
Parameter: ZPL Macro. Replace "p" with the parameter number in the code, for example, PZ_1 will set the solve on parameter 1.	PZ_p
Thermal Coefficient of Expansion: Fixed (turn solve off)	HF
Thermal Coefficient of Expansion: Pickup	HP
Extra Data Value: Fixed (turn solve off). Replace "e" with the extra data number in the code, for example, EF_3 will turn off the solve on extra data value 3.	EF_e
Extra Data Value: Pickup. Replace "e" with the extra data number in the code, for example, EP_4 will set the solve on extra data value 4.	EP_e
Extra Data Value: ZPL Macro. Replace "e" with the extra data number in the code, for example, EZ_4 will set the solve on extra data value 4.	EZ_e
Non-sequential Component Pickup X, Y, Z, Tilt-X, Tilt-Y, Tilt-Z, Material. Replace "o" with the object number in the code, for example, NSC_PX_14 will set a pickup solve on object 14. NSC_PMAT_o is redundant with NSC_MATP_o.	NSC_PX_o, NSC_PY_o, NSC_PZ_o, NSC_PTX_o, NSC_PTY_o, NSC_PTZ_o, NSC_PMAT_o (see left)
Non-sequential Component Material is fixed, model glass, pick up, or offset. Replace "o" with the object number in the code, for example, NSC_MATM_11 will set the material on object 11 to be a model glass. NSC_MATP_o is redundant with NSC_PMAT_o.	NSC_MATF_o, NSC_MATM_o, NSC_MATP_o, NSC_MOFF_o (see left)
Non-sequential Component ZPL Macro solve on X, Y, Z, Tilt-X, Tilt-Y, Tilt-Z. Replace "o" with the object number in the code, for example, NSC_ZX_14 will set a macro solve on object 14.	NSC_ZX_o, NSC_ZY_o, NSC_ZZ_o, NSC_ZTX_o, NSC_ZTY_o, NSC_ZTZ_o (see left)
Non-sequential Component Parameter Pickup. Replace "o" with the object number and "p" with the parameter number in the code, for example, NSC_PP_11_7 will set a pickup solve on object 11, parameter 7.	NSC_PP_o_p (see left)
Non-sequential Component ZPL Macro solve. Replace "o" with the object number and "p" with the parameter number in the code, for example, NSC_ZP_11_7 will set a macro solve on object 11, parameter 7.	NSC_ZP_o_p (see left)

Example:

```
! The following line will add a glass pickup solve
! on surface 7, picking up from surface 5:
SOLVETYPE 7, GP, 5
! Add a thickness pickup with a scale factor of -1:
```

```
SOLVETYPE 7, TP, 5, -1

! Set a pickup solve on surface 1, NSC object 12 Z position,
! pick up from object 11, with a scale factor of 2, offset 3,
! from the parameter 7 column. Note the column number is argument 4.
! The column number is 0 for the same column, 1-6 for x, y, z, tilt x, tilt y,
tilt z,
! respectively. The column number for the parameter columns
! is 6 + the desired parameter number.
! In summary, the syntax is:
! SOLVETYPE, surf, code, object, scale, offset, column
! where code has the object/parameter number embedded as shown in the table
above.
! The syntax for this example is:
SOLVETYPE 1, NSC_PZ_12, 11, 2, 3, 13
```

Related Functions:

SOLV

10.2.14.170. STOPSURF

STOPSURF sets the current stop surface location by number.

Syntax:

```
STOPSURF surf
```

Discussion:

The UPDATE command must be issued before the new data takes effect.

Example:

```
STOPSURF n+2
```

Related Keywords:

UPDATE

10.2.14.171. SUB

See GOSUB.

10.2.14.172. SURFTYPE

This command is obsolete. See " SETSURFACEPROPERTY, SURP ".

10.2.14.173. SUSPENDUPDATES

SUSPENDUPDATES is intended to be used with RESUMEUPDATES .

Prevents any UI editor windows from being updated by ZPL commands while suspended. This command can greatly speed up macro execution when performing many updates to the system, such as inserting hundreds or thousands of merit function rows.

The number of suspend and resume calls are reference counted such that no updates will be performed until RESUMEUPDATES is encountered as many times as SUSPENDUPDATES (or the macro completes execution).

Note that changes will still be made to the internal lens state, however the user interface will not reflect the changes.

Also note that the UI is not automatically updated when RESUMEUDPATES is called; to do so, use the UPDATE EDITORS command.

Example:

```
SUSPENDUPDATES
n = 5
FOR i = 1, n, 1
INSERTMFO i
SETOPERAND i, 1, ONUM("OPDX")
SETOPERAND i, 6, (i/n)
SETOPERAND i, 8, 0
SETOPERAND i, 9, 1
NEXT
RESUMEUPDATES
UPDATE EDITORS
```

10.2.14.174. TELECENTRIC

This command is obsolete. See " SETSYSTEMPROPERTY, SYSP ".

10.2.14.175. TESTPLATEFIT

TESTPLATEFIT calls the test plate fitting routine. See "Test Plate Fitting".

Syntax:

```
TESTPLATEFIT tpd_file, log_file, method, number_cycles
```

Discussion:

This keyword requires string expressions for the test plate data file, and the name of a file for the output log file. The method is an integer between 0 and 4, inclusive, for try all methods, best to worst, worst to best, long to short, and short to long, respectively. The integer number_cycles is 0 for automatic or the maximum number of optimization cycles to execute. Note the tpd_file name should NOT include the path, since all test plate files are in a fixed folder; while the path should be included for the log file.

This keyword may take a long time to execute. It is advisable to display the log file after completion of the fitting, or use some other means to indicate when the fitting is complete.

Example:

```
TESTPLATEFIT "optico.tpd", "c:\temp\logfile.dat", 0, 0
SHOWFILE "c:\temp\logfile.dat"
```

10.2.14.176. THIC

For setting surface properties use the " SETSURFACEPROPERTY, SURP " keyword, and for getting surface properties use the SPRO(surf, code) Numeric Function as discussed under " Numeric Functions ".

10.2.14.177. TIMER

Resets the internal clock. This feature is used in conjunction with the ZPL function ETIM() for measuring the elapsed time since the last TIMER command.

Syntax:

```
TIMER
```

Discussion:

TIMER and ETIM() are used primarily for testing execution efficiency of the ZPL interpreter and various program architectures.

Example:

```
i = 0
TIMER
LABEL 1
x = RAND(1000)
i = i+1
if i < 10000 THEN GOTO 1
FORMAT .1
PRINT "Elapsed time:", ETIM(), "Seconds"
```

10.2.14.178. TOLERANCE

Runs a tolerance analysis and saves the tolerance report to a text file.

Syntax:

```
TOLERANCE top_file_name, out_file_name
```

Discussion:

TOLERANCE runs a tolerance analysis on the current lens based upon the settings in the tolerance options file defined by top_file_name. This file name should end in the extension TOP and be placed in the <data>\Configs folder (see " Folders "). TOP files are created by saving the tolerance options from the tolerance dialog, see "Other buttons". The out_file_name must end in the extension .ZTD to save a ZTD file or .txt to save a text file. The file will be placed in the same folder as the current lens file. Neither top_file_name nor out_file_name should specify a path.

Example:

TOLERANCE example.TOP, output.txt

10.2.14.179. UNLOCKWINDOW

Unlocks any one or all locked windows.

Syntax:

```
UNLOCKWINDOW winnum
```

Discussion:

See " Graphic windows operations ". If the winnum is zero, then all open windows are unlocked. If the winnum argument is -1, then the currently executing window will be unlocked at the end of the macro execution.

Example:

```
UNLOCKWINDOW 2
```

Related Keywords:

LOCKWINDOW

10.2.14.180. UPDATE

Updates pupil positions, index data, paraxial constants, clear semi-diameter or semi-diameters, maximum field normalization values, solves, non-sequential objects, and other data required for both sequential and non-sequential ray tracing.

The UPDATE keyword MUST be used before tracing or evaluating a system if the prescription data, such as surface type, radii, thicknesses, system apertures, wavelengths, or non-sequential object types or parameters, have been altered since the last UPDATE.

If the UPDATE command is followed by the keyword "ALL", then all open windows, except ZPL macro windows, will be updated as well. If the UPDATE command is followed by an expression which evaluates to an integer corresponding to an open window, then the specified window is updated, as long as it is not the window of the macro currently executing. If UPDATE is followed by EDITORS, any open spreadsheet editors will be refreshed to show recent changes in the lens data. If UPDATE is followed by the keyword MACROS, then all open ZPL macro windows other than the calling window are updated.

Syntax:

```
UPDATE
UPDATE ALL
UPDATE n
UPDATE EDITORS
UPDATE MACROS
```

10.2.14.181. VEC1, VEC2, VEC3, VEC4

These keywords are used to set the vector variables VEC1, VEC2, VEC3, and VEC4. Each vector can store an array of double-precision floating point numbers.

Syntax:

```
VEC1 subscript, value
VEC2 subscript, value
VEC3 subscript, value
VEC4 subscript, value
```

Discussion:

VEC1..4 are used to store data in an array. The subscript value can be any expression which is rounded down to an integer. The resulting integer expression must be between 0 and the current maximum vector size, which is initially 1000. The ZPL functions VEC1..4 can be used to extract the data. To change the array size use the SETVECSIZE keyword.

Example:

```
i = 0
LABEL 1
i = i + 1
VEC1 i, i
IF i < 10 THEN GOTO 1
j = 0
LABEL 2
```

```
j = j + 1
VEC2 j, VEC1(j) * VEC1(j)
IF j < 10 THEN GOTO 2
i = 0
LABEL 3
i = i + 1
PRINT "x = ", VEC1(i), " x*x = ", VEC2(i)
IF i < 10 THEN GOTO 3
PRINT
PRINT "All done!"
```

10.2.14.182. WAVL, WWGT

These commands are obsolete. See " SETSYSTEMPROPERTY, SYSP ".

10.2.14.183. XDIFFIA

Computes the Extended Diffraction Image Analysis feature and saves the result to a ZBF file. For a description of the ZBF (Zemax Beam File) format see "Zemax Beam File (ZBF) binary format". For a description of the Extended Diffraction Image Analysis feature see "Extended Diffraction Image Analysis".

Syntax:

```
XDIFFIA outfilename, infilename
```

Discussion:

This keyword requires the name of the output ZBF file, and optionally, the name of the input IMA or BIM file. If the extension to the outfilename is not provided, the extension ZBF will be appended. The extension must be provided on the infilename. The filenames must be enclosed in quotes if any blank or other special characters are used. The outfilename will be placed in the <pop> folder. The infilename must be placed in the <data>\<im- ages> folder. No paths should be provided with the file names.

The settings for the Extended Diffraction Image Analysis feature will be those settings previously saved for the current lens. To make adjustments to the settings, open an Extended Diffraction Image Analysis window, choose the appropriate settings, then press "Save". All subsequent calls to XDIFFIA will use the saved settings. The exceptions are the output file name, which is specified as the first argument after the XDIFFIA keyword, and the input source file, which is optionally specified as the second argument after the XDIFFIA keyword.

Example:

```
XDIFFIA "output will be in this file.ZBF", "SOMEIMAFILE.IMA"
```

Related Functions:

ZBFCLR, ZBFSHOW, ZBFSUM, ZBFMULT

10.2.14.184. ZBFCLR

Clears the complex amplitude data in a ZBF file.

Syntax:

```
ZBFCLR filename
```

Discussion:

This keyword requires only the name of the ZBF file.

General comments about all ZBF related keywords

For a description of the ZBF (Zemax Beam File) format see "Zemax Beam File (ZBF) binary format". OpticStudio requires the filename extension to be ZBF, and will append or replace the extension as required. The filename may be enclosed in quotes if any blank or other special characters are used. The file must be located in the <pop> folder. All ZBF output files will be placed in this same folder.

Example:

```
ZBFCLR "some beam file name.ZBF"
ZBFCLR N$
```

10.2.14.185. ZBFMULT

Multiplies the complex amplitude data in a ZBF file by a complex factor.

Syntax:

```
ZBFMULT filename, Ax, Bx, Ay, By
```

Discussion:

This keyword requires the name of the ZBF file to modify, and factors by which to multiply the real and imaginary components of the complex electric field for each point in the ZBF. The factors are applied to the x- and y-components of the electric field as follows, respectively, where E_{x0} and E_{y0} represent the original values in the ZBF.

$E_x = E_{x0}(Ax + Bx*i)$

$E_y = E_{y0}(Ay + By*i)$

The resulting data is written back to the same ZBF file name.

See "Purpose" for comments that apply to all ZBF related keywords.

Example:

```
ZBFMULT "some beam file name.ZBF", 0.0, 1.0, 0.0, -1.0
```

10.2.14.186. ZBFPROPERTIES

Opens the specified ZBF file and places various data about the beam in a vector variable.

Syntax:

```
ZBFPROPERTIES filename, vector
```

Discussion:

This keyword requires the name of the ZBF file and a value for the vector number to place the data in. The value of vector must be between 1 and 4, inclusive. After the ZBFPROPERTIES function executes, the following beam data will be placed in the specified vector: nx, ny, dx, dy, waist_x, waist_y, position_x, position_y, rayleigh_x, rayleigh_y, wavelength (in lens units), total power, peak irradiance (power per area), the is_polarized flag (0 for no, 1 for yes), and the media index; the values are placed in vector positions 1 through 15.

See "Purpose:" for comments that apply to all ZBF related keywords.

Example:

```
ZBFPROPERTIES "TEST.ZBF", 1
rayleighx = VEC1(9)
```

10.2.14.187. ZBFREAD

Opens the specified ZBF file and places the electric field and beam property data in two user-defined array variables.

Syntax:

```
ZBFREAD filename, beamname, propertyname
```

Discussion:

This keyword requires the name of the ZBF file and the name of two arrays defined by a previous call to DECLARE. The beamname must be a 3 dimensional array, of minimum size (nx, ny, 2) for an unpolarized beam and minimum size (nx, ny, 4) for a polarized beam. The propertyname array must be a one dimensional array of minimum size 14. After the ZBFREAD function executes, the following beam data will be placed in the specified propertyname array: nx, ny, dx, dy, waist_x, waist_y, position_x, position_y, rayleigh_x, rayleigh_y, wavelength (in lens units), total power, peak irradiance (power per area), the is_polarized flag (0 for no, 1 for yes), and the media index; the values are placed in array positions 1 through 15. The electric field data will be placed in the beamname array. The third dimension of the beamname array is 1 for Ex Real, 2 for Ex Imaginary, and if the beam is polarized, 3 for Ey Real, and 4 for Ey Imaginary.

See "Purpose:" for comments that apply to all ZBF related keywords. See also "ZBFWRITE".

Example:

```
! First get the beam size
ZBFPROPERTIES "TEST1.ZBF", 1
nx = vec1(1)
ny = vec1(2)
ip = vec1(14) ! The "is polarized" flag

! Allocate enough memory to hold the beam
IF (ip == 0) THEN DECLARE B, DOUBLE, 3, nx, ny, 2
IF (ip == 1) THEN DECLARE B, DOUBLE, 3, nx, ny, 4
DECLARE P, DOUBLE, 1, 20
ZBFREAD "test1.zbf", B, P

FOR j, 1, ny, 1
FOR i, 1, nx, 1
FORMAT 4.0
PRINT i, j,
FORMAT 12.6
IF (ip == 1)
PRINT B(i, j, 1),
PRINT B(i, j, 2),
PRINT B(i, j, 3),
PRINT B(i, j, 4)
ELSE
PRINT B(i, j, 1),
PRINT B(i, j, 2)
ENDIF
NEXT
NEXT

! save the beam
ZBFWRITE "TEST2.ZBF", B, P

! release the allocated memory
RELEASE B
RELEASE P
```

10.2.14.188. ZBFRESAMPLE

Resamples a ZBF file to a new width and point spacing.

Syntax:

```
ZBFRESAMPLE filename, nx, ny, wx, wy, decenterx, decentery
```

Discussion:

This keyword requires the name of the ZBF file and six numbers. The beam will be resampled and interpolated as required to create a new beam file with nx and ny points, of total width wx and wy, in the x and y directions respectively. The nx and ny values must be powers of 2; such as 32, 64, 128, etc. The decenterx and decentery values may be provided to optionally decenter the new beam relative to the old beam. If either nx or ny is zero, no change is made to the existing beam sampling. If either wx or wy is zero, no change is made to the existing beam width. The length units in the ZBF file are converted automatically to the current lens units. The resulting data is written back to the same file name.

Example:

```
ZBFRESAMPLE "TEST.ZBF", 128, 128, 25.4, 25.4, 0, 0.4
```

10.2.14.189. ZBFSHOW

Displays a ZBF file in a viewer window.

Syntax:

```
ZBFSHOW filename
```

Discussion:

This keyword requires only the name of the ZBF file. This command will open a new window to display the beam file.

Example:

```
ZBFSHOW "new beam data.ZBF"
ZBFSHOW N$
```

10.2.14.190. ZBFSUM

Sums either coherently or incoherently the data in two ZBF files and places the resulting data in a third ZBF file.

Syntax:

```
ZBFSUM coherent, filename1, filename2, outfilename
```

Discussion:

This keyword requires an integer to indicate if the sum is coherent (any value other than zero) or incoherent (zero), and the names of three ZBF files. If an incoherent sum is performed, then the output data will be real valued only. If the two source files do not have the same number of data points, point spacing, and reference radii in both x and y directions, then the second source file listed is first scaled and interpolated, and the reference radii is adjusted, to match the first file before the sum is performed. The length units in the ZBF files are converted automatically to the current lens units. The outfilename may be the same as one of the source file names; in which case the original file is overwritten.

Example:

```
ZBFSUM 1, "a.ZBF", "b.zbf", "coherent a plus b.zbf"
ZBFSUM 0, "a.ZBF", "b.zbf", "incoherent a plus b.zbf"
ZBFSUM 0, A$, B$, C$
```

10.2.14.191. ZBFTILT

Multiplies the data in a ZBF file by a complex phase factor to introduce phase tilt to the beam.

Syntax:

```
ZBFTILT filename, cx, cy, tx, ty
```

Discussion:

This keyword requires the name of the ZBF file and four numbers. The phase of the beam is modified by a phase angle given by θ = (x – cx)tx + (y – cy)ty . The cx and cy values are the center of the phase tilt, and tx and ty are the slopes of the tilt in units of radians per lens unit length. The coordinates x and y refer to positions within the beam file, with the center coordinate (x = 0, y = 0) being at the point (nx/2 + 1, ny/2 + 1) where nx and ny are the number of points in the x and y directions. The length units in the ZBF file are converted automatically to the current lens units. The resulting data is written back to the same file name.

Example:

```
ZBFTILT "TEST.ZBF", 0.0, 0.0, 0.0, 0.01237
```

10.2.14.192. ZBFWRITE

Writes electric field and beam property data arrays to a ZBF file.

Syntax:

```
ZBFWRITE filename, beamname, propertyname
```

Discussion:

This keyword requires the name of the ZBF file and the name of two arrays defined by a previous call to DECLARE. The beamname must be a 3 dimensional array, of minimum size (nx, ny, 2) for an unpolarized beam and minimum size (nx, ny, 4) for a polarized beam. The propertyname array must be a one dimensional array of minimum size 14.

The following beam data must be placed in the specified propertyname array: nx, ny, dx, dy, waist_x, waist_y, position_x, position_y, rayleigh_x, rayleigh_y, wavelength (in lens units), total power, peak irradiance (power per area), the is_polarized flag (0 for no, 1 for yes), and the media index; the values are placed in array positions 1 through 15. The electric field data must be placed in the beamname array. The third dimension of the beamname array is 1 for Ex Real, 2 for Ex Imaginary, and if the beam is polarized, 3 for Ey Real, and 4 for Ey Imaginary.

See also " ZBFREAD ".

10.2.14.193. ZRDAPPEND

Appends data from one ZRD file onto the end of a second file. The first file is left unaltered. This command only makes sense to use if the two ZRD files were traced in the same system, as no attempt is made to modify object numbers or coordinates.

Syntax:

```
ZRDAPPEND infilename, outfilename
```

Discussion:

This keyword requires the name of two input ZRD files. For information on the ZRD format, see "Ray database (ZRD) files". The file names may each be defined by either a literal string or a single string variable. The file names must include the full path name. Although any extensions may be specified, the extension ZRD is recommended for both file names. The input and output file names must be distinct. The input and output files must use the same ZRD format. See also ZRDSUM .

Example:

```
ZRDAPPEND "C:\TEMP\CART.ZRD", "C:\TEMP\HORSE.ZRD"
```

10.2.14.194. ZRDFILTER

Opens a ZRD ray database file, applies a filter, and saves the filtered subset of rays to a new ZRD file.

Syntax:

```
ZRDFILTER infilename, outfilename, filterstring
```

Discussion:

This keyword requires the name of an input ZRD file, the name of the output (filtered) ZRD file, and the filter string to apply. For information on the ZRD format, see "Ray database (ZRD) files". For information on the filter string format see "The filter string". The file names and filter string may each be defined by either a literal string or a single string variable. The file names must include the full path name. Although any extensions may be specified, the extension ZRD is recommended for both the input and output files. The input and output file names must be distinct.

Example:

```
ZRDFILTER "C:\TEMP\TEST1.ZRD", "C:\TEMP\TEST1_H4.ZRD", "H4"
```

10.2.14.195. ZRDPLAYBACK

This keyword reads a ZRD file, and adds ray amplitude data to one or all detectors. This command only makes sense to use if the ZRD file was generated with the currently loaded system, as no attempt is made to modify object numbers or coordinates.

Syntax:

```
ZRDPLAYBACK zrdfilename, surface, detector, clear, filterstring
```

Discussion:

This keyword requires the name of a ZRD file. For information on the ZRD format, see "Ray database (ZRD) files". The file name may each be defined by either a literal string or a single string variable. The file name must not include a path and the file must be in the same folder as the current lens file. The surface argument is the surface number of the Non-sequential surface, use 1 for Non-sequential mode. The integer detector is either 0 for all detectors, or the object number of a specific single detector. If clear is set to anything other than zero, then the specified detector(s) are cleared prior to playback. The optional filterstring is applied to the rays before being played back.

Example:

```
ZRDPLAYBACK "MYRAYS.ZRD", 1, 0, 1, "H3 & H5"
```

10.2.14.196. ZRDSAVERAYS

Opens a ZRD ray database file, applies a filter, and saves the rays in the filtered subset that intersect a specified object number.

Syntax:

```
ZRDSAVERAYS infilename, outfilename, filterstring, object
```

Discussion:

This keyword requires the name of an input ZRD file, the name of the output source ray DAT file, the filter string to apply, and the object number. For information on the ZRD format, see "Ray database (ZRD) files. Source ray files are described in "Source File". For information on the filter string format see "The filter string" . The file names and filter string may each be defined by either a literal string or a single string variable. The file names must include the full path name. Although any extensions may be specified, the extension ZRD is recommended for the input file, and the extension DAT is recommended for the output file. The input and output file names must be distinct. If no filtering is desired, the filterstring argument should be replaced with two double quotes like this: "".

Examples:

```
ZRDSAVERAYS "C:\TEMP\TEST1.ZRD", "C:\TEMP\TEST1_H6.DAT", "", 6
ZRDSAVERAYS "C:\TEMP\TEST2.ZRD", "C:\TEMP\TEST2_H5.SDF", "", 5
```

10.2.14.197. ZRDSUM

Concatenates two ZRD files into a third file. This command only makes sense to use if the two ZRD files were traced in the same system, as no attempt is made to modify object numbers or coordinates.

Syntax:

```
ZRDSUM infilename1, infilename2, outfilename
```

Discussion:

This keyword requires the name of two input ZRD files and the name of the output ZRD file. For information on the ZRD format, see "Ray database (ZRD) files". The file names may each be defined by either a literal string or a single string variable. The file names must include the full path name. Although any extensions may be specified, the extension ZRD is recommended for all three file names. The input and output file names must be distinct. Both input files must use the same ZRD format, and the output file will use this same ZRD format. See also ZRDAPPEND.

Example:

```
ZRDSUM "C:\TEMP\TEST1.ZRD", "C:\TEMP\TEST2.ZRD", "C:\TEMP\TEST_SUM.ZRD"
```

10.2.15. Example Macro 1

The following sample macro will print out the image surface intercept coordinates of the chief ray at every defined field angle:

```
nfield = NFLD()
maxfield = MAXF()
n = NSUR()
FOR i, 1 , nfield, 1
hx = FLDX(i)/maxfield
hy = FLDY(i)/maxfield
PRINT "Field number ", i
RAYTRACE hx,hy,0,0,PWAV()
PRINT "X-field angle : ",FLDX(i)," Y-field angle : ", FLDY(i)
```

```
PRINT "X-chief ray  : ",RAYX(n), " Y-chief ray  : ", RAYY(n)
PRINT
NEXT
PRINT "All Done!"
```

The first line of the macro calls the NFLD() function which returns the number of defined field angles, and assigns it to the variable "numfield". The second line calls MAXF(), which returns the maximum radial field and stores the value in the variable "maxfield". The number of surfaces is then stored in the variable "n" by calling the function NSUR().

The FOR loop is then defined with i being the counter for the field position, starting at 1, with a maximum value of nfield, and an increment of 1 on each loop. The two macro lines which define "hx" and "hy" use the functions FLDX() and FLDY(), which return the x- and y- field values for the current field position number "i".

The chief ray is then traced by the keyword RAYTRACE. Note the chief ray intersects the center of the pupil, which is why the two pupil coordinates are both zero. The PWAV() function returns the primary wavelength number, which is usually used for the chief ray. The various PRINT commands will output the chief ray coordinates on the image surface to the screen.

10.2.16. Example Macro 2

The following sample macro will estimate the RMS spot radius (on axis) of the current optical system. The macro traces many random rays through the system, and records the radial offset from the primary wavelength chief ray. The current wavelength weighting is applied to estimate the RMS spot radius.

```
PRINT "Primary wavelength is number ",
FORMAT .0
PRINT PWAV(),
FORMAT .4
PRINT " which is ", WAVL(PWAV()), " micrometers."
PRINT "Estimating RMS spot radius for each wavelength."

! How many random rays to trace to make estimate?
n = 100

! Initialize the timer
TIMER

! Store the number of surfaces for later use
ns = NSUR()

! Start at wavelength 1
weightsum = 0
wwrms = 0

FOR w, 1, NWAV(), 1
rms = 0
FOR i, 1, n, 1

hx = 0
hy = 0
angle = 6.283185 * RAND(1)
! SQRT yields uniform distribution in pupil
radius = SQRT(RAND(1))
px = radius * COSI(angle)
py = radius * SINE(angle)
RAYTRACE hx, hy, px, py, w
x = RAYX(ns)
y = RAYY(ns)
rms = rms + (x*x) + (y*y)
```

```
NEXT
rms = SQRT(rms/n)
wwrms = wwrms + ( WWGT (w) * rms )
weightsum = weightsum + WWGT(w)
FORMAT .4
PRINT "RMS spot radius for ", WAVL(w),
FORMAT .6
PRINT " is ", rms

NEXT
wwrms = wwrms / weightsum
PRINT "Wavelength weighted rms is ", wwrms
FORMAT .2
t = ETIM()
PRINT "Elapsed time was ",t," seconds."
```

Note the use of the trailing commas in the first two PRINT commands. These allow the data printed from the first three PRINT commands to appear on the same line. The intervening FORMAT commands changes the way the numerical output is printed, even on the same line. The "!" character is used to indicate a comment, which is ignored when the macro is run.

10.2.17. Calling a Macro from within a Macro

To call another ZPL macro from within a ZPL macro, use the CALLMACRO keyword. Data may also be passed between the macros using the keywords CALLSETDBL and CALLSETSTR and the functions CALD and $CALLSTR.

The first macro that is executed, usually from the ZPL Macro Dialog box, is the "parent" macro. The parent macro can call other macros, and these macros are called "child" macros. The parent macro creates a buffer of 51 numeric and 51 string values. The buffer can be used to set or retrieve numeric or string values, which allows macros to share data, or pass arguments to one another. The buffer is unique to the parent macro, and any child macros called by the parent. If more than one parent macro is executing in parallel (for example, in two windows each updating a different macro) each parent has their own buffer, and no data is shared between parents.

The easiest way to see how the parent and child macros work is by example. This example will use just two very simple macros, but there is no hard limit as to how many macros may be called or nested.

Consider two macros, called PARENT.ZPL and CHILD.ZPL The PARENT.ZPL macro is:

```
CALLSETDBL 1, 3.5
CALLSETSTR 1, "Hello World"
CALLMACRO CHILD.ZPL
PRINT CALD(1)
```

The CHILD.ZPL macro is:

```
PRINT "Executing child macro"
PRINT CALD(1)
A$ = $CALLSTR(1)
PRINT A$
CALLSETDBL 1, 7.11
```

When the PARENT.ZPL macro is executed, the macro uses CALLSETDBL to place the numeric value 3.5 in the parent's call numeric buffer in index position 1. The CALLSETSTR is then used to place the string "Hello World" in the parent's call string buffer in index position 1. Any index between 0 and 50, inclusive may be used for either the numeric or string buffers.

The PARENT.ZPL macro then executes the macro CHILD.ZPL. The child macro prints the message "Execut- ing child macro" to the child's output window, which is not visible during execution. The child then extracts the numeric value from numeric buffer at index 1 using CALD(1), and the string value from the string buffer index 1 using

$CALLSTR(1). These values are also printed. Finally, the child macro overwrites the numeric value in index 1 with the new value 7.11, and the child macro completes.

Control now returns to the parent macro, which copies the child macro output window contents to the parent's output window, and the child's output window contents are discarded. Lastly, the modified value from the numeric call buffer at index 1 is printed. The output of this macro looks like this:

```
Executing child macro
3.5000
Hello World
7.1100
```

The sample macros PARENT.ZPL and CHILD.ZPL are included with OpticStudio in the Macros folder.

10.2.18. Running Macros from the Command Line

OpticStudio supports running a ZPL macro by passing special flags via the command line. Optional string arguments can also be passed through the command line and retrieved within the macro allowing automation of complex tasks. The syntax is as follows:

```
[PATH_TO_OPTICSTUDIO.EXE] -zpl="[FULL_PATH_TO_ZPL_FILE]"
{-v[ARG_NAME_1]="[VALUE]", -v[ARG_NAME_2]="[VALUE]" ...}
```

Note that argument names cannot contain spaces, quotes ("") are required around the macro path and argument values, and the '-zpl' flag must come before any string arguments. All '-v' arguments are optional and can be retrieved using the $GETARG string function:

```
var$ = $GETARG("variable_name")
```

or

```
var$ = $GETARG(varName$)
```

Note that argument names are not case sensitive, so $GETARG("var1") is the same as $GETARG("Var1").

The "IsAutomated" string argument is always populated automatically, with a value of 'True' if the macro is executing via the command line, and 'False' otherwise.

OpticStudio will automatically shut down once the macro has finished running, however it is possible something might prevent the macro from completing, such as a dialog box waiting for user input.

Example command line:

```
C:\Program Files\Zemax OpticStudio 17.5\OpticStudio.exe -zpl="c:\temp\TestCL.zpl"
 -vSurface="1" -vOutputFile="c:\temp\test file.txt"
Example ZPL snippet:
IsAuto$ = $GETARG("IsAutomated")
IF (IsAuto$ $== "True")
                Surf$ = $GETARG("Surface")
                Outfile$ = $GETARG("OutputFile")
                n = SVAL(Surf$)
                OUTPUT Outfile$
                PRINT "This is running from the command line!"
                PRINT "Surface " $STR(n)
ELSE
                PRINT "This is running from OpticStudio!"
                n = NSUR()
ENDIF
```

10.2.19. Using ZPL Macro Solves

ZPL Macro solves call a user defined ZPL Macro to determine the solve value. For example, see the keyword "SOLVEBEFORESTOP". For more information on solves see the "Solves" section of the Lens Data Editor in the Setup Tab.

Macro solves work by invoking a user defined ZPL Macro to compute the solve. Any numerical value that can be computed in a macro can be returned to the editor calling the solve macro. The macro may make use of the data that exists elsewhere in the editors, such as previous surfaces. Once the data is computed, the macro uses the keyword SOLVERETURN to pass the data back to the editor.

For a simple example, here is a macro that computes the first -order optical power of the interface between surfaces 1 and 2, and returns this power as the solve value:

```
n1 = INDX(1)
n2 = INDX(2)
c2 = CURV(2)
SOLVERETURN (n2-n1)*c2
```

There is a disadvantage to referring to specific surfaces directly by surface number as this macro does. If new surfaces are inserted or deleted, the macro will need to be edited to reflect the new surface numbers. Also, certain features, such as mirror substrate drawing, will not work correctly if macro solves refer to fixed surface numbers. The solution is to use the ZPL function SURC to find surfaces with specific text in the comment column of the Lens Data Editor. Suppose the comment "My Surface" was placed on the desired first surface in the Lens Data Editor. The macro could support the power computation above with this revised macro:

```
A$ = "My Surface"
SURF = SURC(A$)
n1 = INDX(SURF)
n2 = INDX(SURF+1)
c2 = CURV(SURF+1)
SOLVERETURN (n2-n1)*c2
```

The surface or object number from which the macro solve is being called can also be extracted using the numeric ZPL SOSO function.

Use surface comments and the SURC function in solve macros whenever possible.

Be aware that when tolerancing, the SURC function will keep returning the original surface number even if additional surfaces are added automatically.

For example, if SURC(surface_name$) returns 10 in the original file, when the file is toleranced it will keep returning 10 even if surfaces are added during the tolerancing.

To avoid this behaviour, use the numeric function SRCN instead.

10.2.19.1. Important Considerations for ZPL Macro Solves

Macro solves are very general, allowing virtually any computation to be used to determine a solve value. All functions and keywords supported by ZPL are available. There are no differences between ZPL macros executed from the macro menu or executed as a solve. However, many ZPL keywords and some functions should not be used in a macro solve. For example, calling "UPDATE" from the solve will update all solves, which will invoke the macro again, resulting in an infinite loop. Other keywords, such as INPUT, will require the user to enter data every time the solve is called, which may be an enormous number of times. Other keywords that set values in the editors or create merit functions should never be used. In general, macro solves should be kept short and simple, avoid lengthy computations, and should not make modifications of any lens data. Macro solves should also never depend upon data in the editors that is subsequent to the solve; as this may also create erroneous data if the solve is called first and then the source data is modified by a separate, subsequent solve. OpticStudio makes no attempt to qualify or validate the solve macro. This feature offers great power and flexibility, but must be used carefully.

Macro solves may be placed prior to the stop surface only if the computations in the macro aren't based upon any ray data. Care should be taken to ensure that this is indeed the case. See the description of the ZPL keyword "SOLVEBEFORESTOP" for details.

If an error or invalid condition occurs in the macro, and the solve value cannot be computed, then the macro should return without calling SOLVERETURN. The absence of the SOLVERETURN call will indicate to OpticStudio that the solve cannot be computed and the optical system is in an error condition. This is particularly important during optimization.

10.2.19.2. Integer Codes for Column Numbers

To set a Pickup solve from the ZPL macro language, the column number is required. The column number is defined as follows:

0: The current column, which is the column the solve is placed in. This is the default for all pickup solves.

1-4: Radius, Thickness, Conic, and Clear Semi-Diameter or Semi-Diameter, respectively.

5-17: Parameter 0 through 12, respectively

10.3. String Codes

String codes are 3 character arguments used in the keywords "GETTEXTFILE" and "OPENANALYSISWINDOW".

The string codes are listed below. Note that the codes are case sensitive.

"Code"	**DESCRIPTION**
File Codes	
"New"	NEW_FILE
"Ope"	OPEN_FILE
"Sav"	SAVE_FILE
"Sas"	SAVE_AS
"Bac"	BACKUP_TO_ARCHIVE_FILE
"Res"	RESTORE_FROM_ARCHIVE_FILE
"Ins	INSERT_LENS
"Prf"	PREFERENCES
"Ext"	EXIT
"Upd"	UPDATE
"Upa"	UPDATE_ALL
"Gen"	GENERAL_LENS_DATA
"Fie"	FIELD_DATA
"Wav"	WAVELENGTH_DATA
"Nxc"	NEXT_CONFIGURATION
"Lac"	LAST_CONFIGURATION
"LDE"	LENS_DATA_EDITOR

"MFE"	MERIT_FUNCTION_EDITOR
"MCE"	MULTI_CONFIG_EDITOR
"TDE"	TOLERANCE_DATA_EDITOR
"NCE"	NON_SEQUENTIAL_EDITOR
"Und"	UNDO
"Red"	REDO
Layout Codes	
"Lay"	2D_LAYOUT
"L3d"	3D_LAYOUT
"Lsh"	SHADED_MODEL_LAYOUT
"Ele"	ZEMAX_ELEMENT_DRAWING
"ISO"	ISO_ELEMENT_DRAWING
"L3n"	NSC_3D_LAYOUT
"LSn"	NSC_SHADED_MODEL_LAYOUT
"Obv"	NSC_OBJECT_VIEWER
"Pvr"	CAD_PART_VIEWER
Fan Codes	
"Ray"	RAY_FAN
"Opd"	OPD_FAN
"Pab"	PUPIL_ABERRATION_FAN
Spots Codes	
"Spt"	SPOT_DIAGRAM
"Stf"	THROUGH_FOCUS_SPOT
"Sff"	FULL_FIELD_SPOT
"Sma"	SPOT_MATRIX
"Smc"	SPOT_MATRIX_CONFIG
MTF Codes	
"Mtf"	MODULATION_TF
"Tfm"	THROUGH_FOCUS_MTF
"Smf"	SURFACE_MTF
"Mth"	MTF_VS_FIELD
"Fmm"	FFT_MTF_MAP
"Gmm"	GEOMETRIC_MTF_MAP
"Hmf"	HUYGENS_MTF
"Hsm"	HUYGENS_SURFACE_MTF

"Hmh"	HUYGENS_MTF_VS_FIELD
"Gtf"	GEOMETRIC_MTF
"Tfg"	THROUGH_FOCUS_GTF
"Htf"	HUYGENS_THROUGH_FOCUS_MTF
"Gvf"	GEOMETRIC_MTF_VS_FIELD
PSF Codes	
"Fps"	FFT_PSF
"Pcs"	PSF_CROSS_SECTION
"Lsf"	FFT_LINE/EDGE_SPREAD
"Hps"	HUYGENS_PSF
"Hcs"	HUYGENS_PSF_CROSS_SECTION
Wavefront Codes	
"Wfm"	WAVEFRONT_MAP
"Int"	INTERFEROGRAM
"Ffa"	FULL-FIELD_ABERRATION
"Foa"	FOUCAULT_ANALYSIS
Surface Sag and Phase Codes	
"Srs"	SURFACE_SAG
"Srp"	SURFACE_PHASE
"Scv"	SURFACE_CURVATURE
"Ssc"	SURFACE_SAG_CROSS
"Spc"	SURFACE_PHASE_CROSS
"Scc"	SURFACE_CURVATURE_CROSS
RMS Codes	
"Rms"	RMS_VS_FIELD
"Rmw"	RMS_VS_WAVELENGTH
"Rmf"	RMS_VS_FOCUS
"Rfm"	RMS_FIELD_MAP
Encircled Energy Codes	
"Enc"	DIFF_ENCIRCLED_ENERGY
"Gee"	GEOM_ENCIRCLED_ENERGY
"Lin"	LINE/EDGE_SPREAD
"Xse"	EXTENDED_SOURCE_ENCIRCLED
"Rel"	RELATIVE_ILLUMINATION
"Foo"	FOOTPRINT_ANALYSIS

"Uni"	UNIVERSAL_PLOT_1D
"Un2"	UNIVERSAL_PLOT_2D
"Fcd"	FIELD_CURV/DISTORTION
"Grd"	GRID_DISTORTION
"Lon"	LONGITUDINAL_ABERRATION
"Lat"	LATERAL_COLOR
"Sim"	IMAGE_SIMULATION
"Ima"	GEOMETRIC_IMAGE_ANALYSIS
"Ibm"	GEOMETRIC_BITMAP_IMAGE_ANALYSIS
"Lsa"	LIGHT_SOURCE_ANALYSIS
Other Miscellaneous Codes	
"ABg"	ABG_DATA_CATALOG
"Agm"	ATHERMAL_GLASS_MAP
"Bfv"	BEAM_FILE_VIEWER
"C31"	SRC_COLOR_CHART_1931
"C76"	SRC_COLOR_CHART_1976
"Caa"	COATING_ABS_VS_ANGLE
"Car"	CARDINAL_POINTS
"Cas"	COAT_ALL_SURFACES
"Caw"	COATING_ABS_VS_WAVELENGTH
"Cca"	CONVERT_TO_CIRCULAR_APERTURES
"Cda"	COATING_DIA_VS_ANGLE
"Cdw"	COATING_DIA_VS_WAVELENGTH
"Cfa"	CONVERT_TO_FLOATING_APERTURES
"Cfm"	CONVERT_TO_MAXIMUM_APERTURES
"Cfo"	CONVERT_FORMAT
"Cfs"	CHROMATIC_FOCAL_SHIFT
"Cgl"	CONVERT_GLOBAL_TO_LOCAL
"Chk"	SYSTEM_CHECK
"Clg"	CONVERT_LOCAL_TO_GLOBAL
"Cls"	COATING_LIST
"Cna"	COATING_RET_VS_ANGLE
"Cng"	CONVERT_TO_NSC_GROUP
"Cnw"	COATING_RET_VS_WAVELENGTH
"Coa"	CONVERT_ASPHERE

"Cpa"	COATING_PHASE_VS_ANGLE
"Cpw"	COATING_PHASE_VS_WAVELENGTH
"Cra"	COATING_REFL_VS_ANGLE
"Crw"	COATING_REFL_VS_WAVELENGTH
"Csd"	NSC_CONCATENATE_SPECTRAL_SOURCE_FILES
"Csf"	NSC_CONVERT_TO_SPECTRAL_SOURCE_FILE
"Cta"	COATING_TRAN_VS_ANGLE
"Ctw"	COATING_TRAN_VS_WAVELENGTH
"Dbl"	DOUBLE_PASS
"Dim"	PARTIALLY_COHERENT_IMAGE_ANALYSIS
"Dip"	BIOCULAR_DIPVERGENCE/CONVERGENCE
"Dis"	DISPERSION_PLOT
"Drs"	NSC_DOWNLOAD_RADIANT_SOURCE
"Dvl"	DISPERSION_VS_WAVELENGTH_PLOT
"Dvr"	NSC_DETECTOR_VIEWER;
"Eca"	EXPLODE_CAD_ASSEMBLY
"Ecp"	EXPLODE_CREO_PARAMETRIC_ASSEMBLY
"Ect"	EDIT_COATING
"Eec"	EXPORT_ENCRYPTED_COATING
"Eia"	EXPLODE_INVENTOR_ASSEMBLY
"Fba"	FIND_BEST_ASPHERE
"Fcl"	FIBER_COUPLING
"Fld"	ADD_FOLD_MIRROR
"Flx"	DEL_FOLD_MIRROR
"Fov"	BIOCULAR_FIELD_OF_VIEW_ANALYSIS
"Fvw"	NSC_ZRD_FLUX_VS_WAVELENGTH
"Gbp"	PARAX_GAUSSIAN_BEAM
"Gbs"	SKEW_GAUSSIAN_BEAM
"Gcp"	GLASS_COMPARE
"Gft"	GLASS_FIT
"Gho"	GHOST_FOCUS
"Gip"	GRIN_PROFILE
"Gla"	GLASS_CATALOG
"Glb"	GLOBAL_OPTIMIZATION
"Gmf"	GENERATE_MAT_FILE

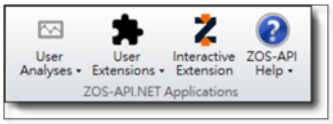
"Gmp"	GLASS_MAP
"Gpr"	GRADIUM_PROFILE
"Grs"	NSC_GENERATE_RADIANT_SOURCE
"Gst"	GLASS_SUBSTITUTION_TEMPLATE
"Ham"	HAMMER_OPTIMIZATION
"Hlp"	HELP
"Igs"	EXPORT_IGES_FILE
"Iht"	ANGLE_VS_IMAGEHEIGHT
"Imv"	IMA/BIM_FILE_VIEWER
"Jbv"	JPG/BMP_FILE_VIEWER
"Len"	LENS_SEARCH
"Lok"	LOCK_ALL_WINDOWS
"Ltr"	NSC_LIGHTNING_TRACE
"Mfl"	MERIT_FUNCTION_LIST
"Mfo"	MAKE_FOCAL
"Opt"	OPTIMIZATION
"Pal"	POWER_FIELD_MAP
"Pat"	NSC_ZRDA_PATH_ANALYSIS
"Pci"	PARTIALLY_COHERENT_IMAGE_ANALYSIS
"Per"	PERFORMANCE_TEST
"Pha"	POL_PHASE_ABERRATION
"Pmp"	POL_PUPIL_MAP
"Pol"	POL_RAY_TRACE
"Pop"	PHYSICAL_OPTICS_PROPAGATION
"Ppm"	POWER_PUPIL_MAP
"Pre"	PRESCRIPTION_DATA
"Ptf"	POL_TRANSMISSION_FAN
"Pzd"	PLAYBACK_ZRD_ON_DETECTORS
"Qad"	QUICK_ADJUST
"Qfo"	QUICK_FOCUS
"Raa"	REMOVE_ALL_APERTURES
"Rcf"	RELOAD_COATING
"Rda"	NSC_REVERSE_RADIANCE_ANALYSIS
"Rdb"	RAY_DATABASE
"Rdw"	NSC_ROADWAY_LIGHTING_ANALYSIS

"Rev"	REVERSE_ELEMENTS
"Rml"	REFRESH_MACRO_LIST
"Rtc"	RAY_TRACE_CONTROL
"Rtr"	RAY_TRACE
"Rva"	REMOVE_VARIABLES
"Rxl"	REFRESH_EXTENSIONS_LIST
"Sag"	SAG_TABLE
"Sca"	SCALE_LENS
"Sdi"	SEIDEL_DIAGRAM
"Sdv"	SRC_DIRECTIVITY
"Sei"	SEIDEL_COEFFICIENTS
"Sfv"	SCATTER_FUNCTION_VIEWER
"Sld"	SLIDER
"Slm"	STOCK_LENS_MATCHING
"Spj"	SRC_PROJECTION
"Spo"	SRC_POLAR
"Spv"	SCATTER_POLAR_PLOT
"Srv"	SRC_RSM_VIEWER
"Ssg"	SYSTEM_SUMMARY_GRAPHIC
"Ssp"	SRC_SPECTRUM
"Sti"	NSC_CONVERT_SDF_TO_IES
"Sur"	SURFACE_DATA
"Sys"	SYSTEM_DATA
"Tde"	TILT/DECENTER_ELEMENTS
"Tls"	TOLERANCE_LIST
"Tol"	TOLERANCING
"Tpf"	TEST_PLATE_FIT
"Tpl"	TEST_PLATE_LISTS
"Tra"	POL_TRANSMISSION
"Trw"	TRANSMISSION_VS_WAVELENGTH
"Tsm"	TOLERANCE_SUMMARY
"Unl"	UNLOCK_ALL_WINDOWS
"Vig"	VIGNETTING_PLOT
"Vop"	VISUAL_OPTIMIZER
"Vra"	VARIABLE_RAD

"Vth"	VARIABLE_THI
"Xdi"	EXTENDED_DIFFRACTION_IMAGE_ANALYSIS
"Xis"	EXPORT_IGES/STEP/SAT/STL_FILE
"Yni"	YNI_CONTRIBUTIONS
"Yyb"	Y-YBAR
"Zat"	ZERNIKE_ANNULAR_TERMS
"Zbb"	EXPORT_ZEMAX_BLACK_BOX_DATA
"Zex"	ZEMAX_EXTENSIONS
"Zfr"	ZERNIKE_FRINGE_TERMS
"Zpd"	NSC_ZEMAX_PART_DESIGNER
"Zpl"	EDIT/RUN_ZPL_MACROS
"Zst"	ZERNIKE_STANDARD_TERMS
"Zvf"	ZERNIKE_COEFFICIENTS_VS_FIELD

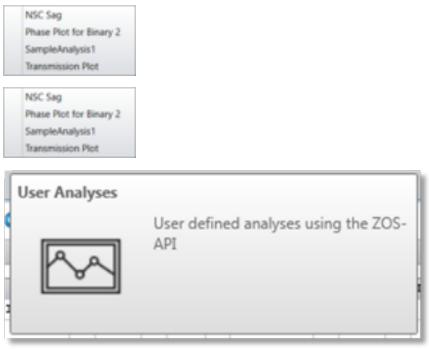
10.4. ZOS-API.NET Applications Group

ZOS-API Applications is a group of features the Programming Tab.



For more information, please see the section "About the ZOS-API".

10.4.1. User Analyses



The User Analyses button is found in the ZOS-API Applications section of the Programming tab. The feature shows a drop-down list of all analyses that have been created using ZOS-API and stored in the <data>\ZOS-API\User Analysis folder (see "Folders"). Clicking on the executable name will execute that analysis.

See "ZOS Inherent – User Analysis" in the section "ZOS Inherent (ZOS uses your Application)" for more details.

For a full description of the ZOS-API, please see "About the ZOS-API".