OptiFDTD

Visual Basic Scripting Reference

Finite-Difference Time-Domain

OptiFDTD Version 16.0.1 for Microsoft Windows® 10 64-bit

Optiwave

Table of Contents

Disclaimer

Optiwave makes no representation or warranty with respect to the adequacy of this documentation or the programs which it describes for any particular purpose or with respect to its adequacy to produce any particular result. In no event shall Optiwave, its employees, its contractors or the authors of this documentation, be liable for special, direct, indirect, or consequential damages, losses, costs, charges, claims, demands, or claim for lost profits, fees, or expenses of any nature or kind.

1 Introduction to Visual Basic Scripting

You can control many of the objects that you use when you create the layout using Visual Basic Script (VB Script). You can manipulate the geometric and physical properties of waveguides and input planes, and run simulations. This powerful feature eliminates many lengthy, repetitive manual tasks.

This book describes how to use VBScript with OptiFDTD, including examples and tutorials.

1.1 Quick Visual Basic Scripting reference

The following is a basic description of a selected part of the VB Script language. For more information and a complete reference, refer to Microsoft online documents.

In Visual Basic, every variable type is *variant*. Variants are tagged unions. A variant stores a value and information on the value type. Variants support the following types:

- 2-byte and 4-byte integer
- 4-byte and 8-byte floating points
- Strings
- Booleans
- HRESULT
- Pointers to IUnknown and IDispatch interfaces

You must be aware of the type of data required by the OptiFDTD VB Script programming interface. For example, consider the number 80 — the number 80 can be stored as a 2-byte integer, a 4-byte integer, a 4-byte float, an 8-byte float, or even as a string. When calling the application programming interface (API), you must ensure that you are passing the proper type. Otherwise, the API will not behave properly. You must convert to the proper type. Type conversions can be done as:

- **CInt(expression)**

returns a variant of subtype 2-byte integer (referred to as just an integer)

- **CLng(expression)**

returns a variant of subtype 4-byte integer (referred to as a long)

- **CSng(expression)**

returns a variant of subtype 4-byte floating point (referred to as a single)

- **CDbl(expression)**

returns a variant of subtype 8-byte floating point (referred to as a double)

- **CStr(expression)**

returns a variable of subtype string (a system string of type BSTR or VB string)

1.2 Example

```
Dim  MyDouble
Dim  MyString
MyDouble = 437.324  'MyDouble is now a double.
MyString =  CStr (MyDouble)  'MyString is now a string and contains "437.324".
```

For loops are the main source of iteration control of multiple iteration simulations. **For** loops must have a count and the loop body must be encapsulated with a **Next** statement. **Step** is used as a way to specify the amount that the loop counter is to be changed in each iteration. This value can be negative, providing a decrementing loop if **start > end**.

```
  For  counter = start  To  end < Step  step>
    <statements>
  Next
```

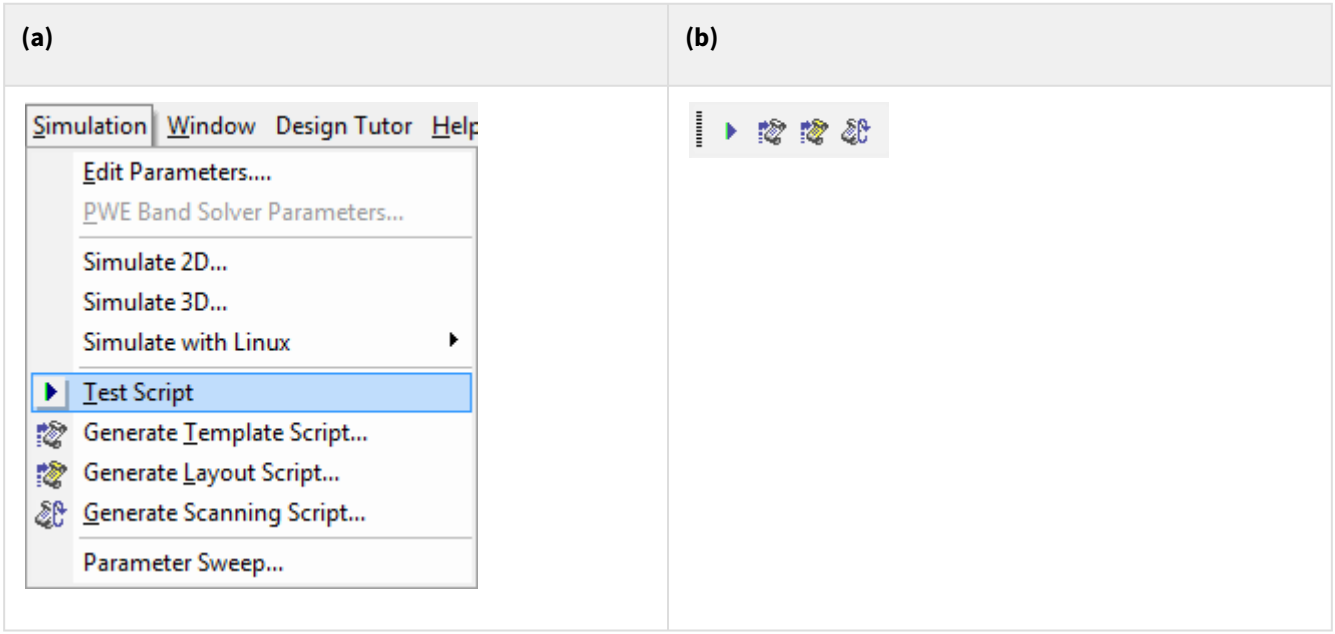
If structures provide flow control. For example:

```
  If  condition1 then
    <statements>
  ElseIf  condition2  Then
    <statements>
  Else
    <statements>
  End  If
```

VBScript also has many useful math functions. Methods for trigonometry, logarithms, exponential, square root, and randomize are all included. For more information, see refer to Microsoft online documents.

2 Running scripts

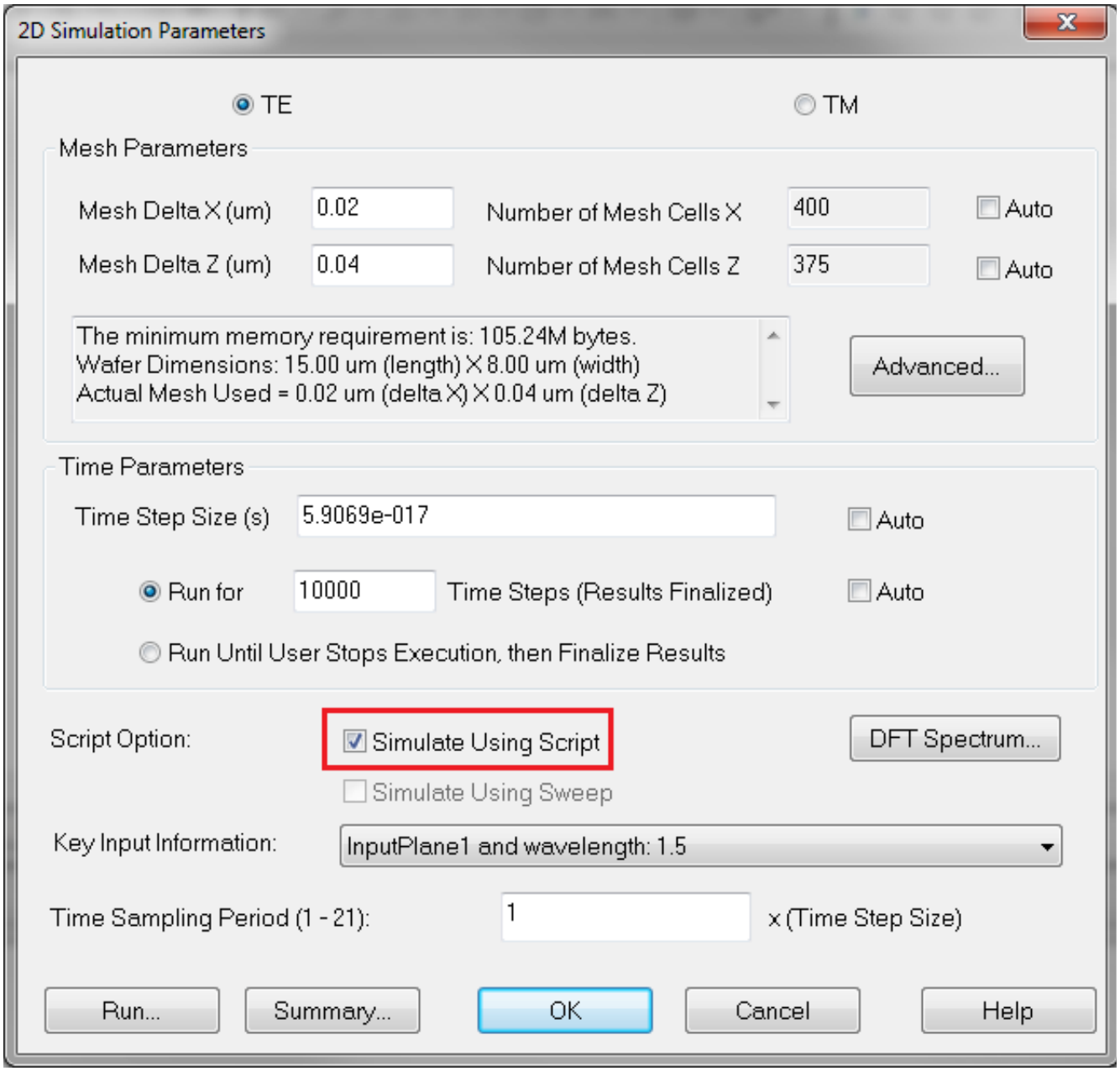
You can create scripts manually by typing script in the Scripting window, or automatically by using a combination of scripting tools (scripting tools will be discussed later) or any combination of manual or automatic script generation. You can automatically generate part of the script and then manually modify the script according to your needs.



1 Testing a script: (a) from the simulation menu, (b) from the scripting toolbar

To run a script, from the **Simulation** menu, select **Test Script** or click on the **play** button of the scripting toolbar. This will execute the script defined in the **Scripting** tab of OptiFDTD Designer. However, it will not run simulations. This feature is useful to either test if the script is valid or to generate complex layouts using scripting.

In order for scripts to call the simulator, you will need to call **ParamMgr.Simulate** in your script and check the **Simulate using script** option in the simulation parameters.



1 Simulate using script option in the simulation parameters window

It is often beneficial to put this method in a loop so that it is called after all manipulations are completed. Each time **ParamMgr.Simulate** is called, a simulation of the current layout is executed. While this is being done, the script will pause and cease to execute, which preserves the state of the layout. The simulator creates an output data file with an integer appended to it that specifies the

simulation iteration number. Starting from 1, the simulation iteration is incremented every time the simulate method is called.

3 Scripting tools

The following three features are tools to generate scripts:

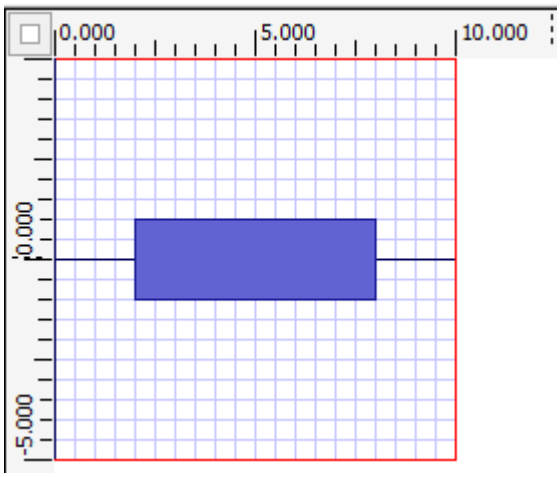
These scripting tools are a starting point for more complex scripting. They are a good way to familiarize yourself with the OptiFDTD application programming interface (API) and the properties of different objects. You can create an object, set its properties, and generate the script.

Note: Using any of these tools will overwrite the current script with the newly generated one.

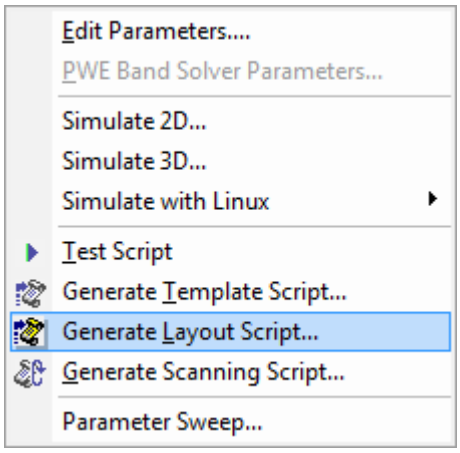
3.1 Layout scripts

Create a layout script that is an exact representation of the design. Like the template script, the layout script generates script equivalents for waveguides, input planes, and observation points. However, the labels for the objects will be the same as the labels in the layout.

Consider the following layout:



2 Example layout



3 Generate layout script

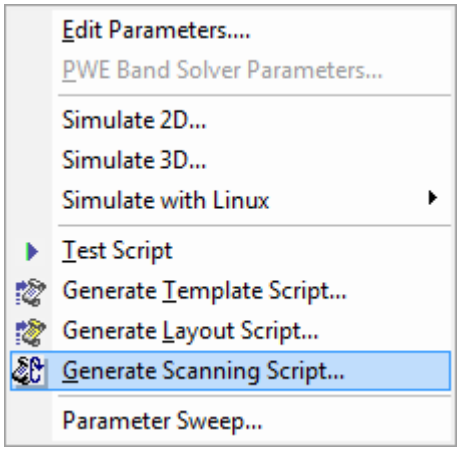
Clicking on the **Generate Layout script** option in the **Simulation** menu will create a script in the **scripting** tab containing the following:

```
WGMgr.DeleteAll
ObservePtMgr.DeleteAll
PointSourceMgr.DeleteAll
InputPlaneMgr.DeleteAll

Dim  Linear1
Set  Linear1 = WGMgr.CreateObj (  "WGLinear" ,  "Linear1"  )
Linear1.SetPosition    2,    0,    8,    0
Linear1.SetAttr  "WidthExpr" ,  "2"
Linear1.SetAttr  "Depth" ,  "0"
Linear1.SetAttr  "StartThickness" ,  "1.000000"
Linear1.SetAttr  "EndThickness" ,  "1.000000"
Linear1.SetProfileName  "WG Channel Example"
Linear1.SetDefaultThicknessTaperMode  True
```

The **DeleteAll** calls are used to clear the current layout and avoid duplicates. **Linear1** is the label used in the layout to represent the linear waveguide. The script successively create a new linear waveguide (using the type **"WGLinear"**), sets its position, width and material. For more information about these function, refer to the VBScript Commands page

3.2 Scanning scripts

Scanning scripts are used to launch several simulations in sequence. Usually, the layout is modified between 2 iterations of the simulator.



4 Generating a scanning script

Selecting the **Generate Scanning Script** entry in the **Simulation** menu will erase the current scripting tab and replace it with the following:

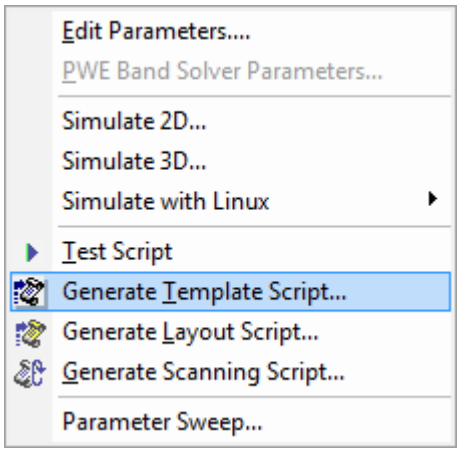
```
NumIterations = 1

For  x = 1 to NumIterations

    ParamMgr.Simulate
    WGMgr.Sleep( 50 )
Next
```

This is a VB **For-loop**. By default, the number of iterations is 1, so the loop is only executed once. By changing the **NumIterations** value to 10 for example, the loop will execute 10 times. Each time, the simulator will be called (using **ParamMgr.Simulate**), and a small pause of 50 ms will be added (to let the user interface of OptiFDTD Designer time to redraw itself).

You can add anything before the call to **ParamMgr.Simulate**. For example, an object can be modified at each iteration, or the angle of incidence of the light can be altered etc.

3.3 Template scripts

Creates a template script that is based on the layout objects. This means that a scripting equivalent of the design is created. The labels are generated dynamically. This script can be run repeatedly, creating a type of copy/paste function.



5 Generating template scripts

Clicking on **Generate Template Script** in the **Simulation** menu will erase the current content in the **scripting** tab and replace it with the following:

```
Dim Linear1
Set Linear1 = WGMgr.CreateObj (  "WGLinear" , WGMgr.FindID(  "Linear"  ) )
Linear1.SetPosition     2 ,     0 ,     8 ,     0
Linear1.SetAttr  "WidthExpr" ,  "2"
Linear1.SetAttr  "Depth" ,  "0"
Linear1.SetAttr  "StartThickness" ,  "1.000000"
Linear1.SetAttr  "EndThickness" ,  "1.000000"
Linear1.SetProfileName  "WG Channel Example"
Linear1.SetDefaultThicknessTaperMode True
```

Dynamic labels are created with the **WGMgr.FindID** method. This method takes a string as input. It returns the same string, but with an integer appended to it. The resulting new label is unique.

4 VBScript Commands

Managers are top-level items used by the scripting environment and are always available. They are created when the application starts. You can access most of OptiFDTD features and setting through these managers. They include:

4.1 Waveguide Manager

The Waveguide Manager is a top-level-item. This object exists the whole time that the script is running. It can be accessed during scripting by the string identifier: **“WGMgr”**. This object is also responsible for creation, deletion, ID access, and creating and breaking waveguide links.

Method signature	Description
WGMgr.Sleep *(Long Time_ms)*	Does nothing for the specified time (in ms)
Object WGMgr.CreateObj *(String Type,String ID)*	Creates a new object. First parameter is a string identifier specifying the type of waveguide to create. The second is the string ID that the waveguide assumes. If the ID already exists, this method returns the object that has the ID passed. The following types are supported: • **“WGLinear”** • **“WGTaperLinear”** • **“WGTaperParabolic”** • **“WGTaperExp”** • **“WGArc”** • **“WGElliptic”** • **“WGRing**" • **“WGSBendTaperSin”** • **“WGSBendSin”** • **“WGSBendTaperCos”** • **“WGSBendCos”** • **“WGSBendTaperArc”** • **“WGSBendArc”** • **“WGPolynomial”** • **“WGLensParabolic”** • **“WGLensHyperbolic”** • **“WGElliptic”** • **“WGLensElliptic”**

Method signature	Description
	• **"WGLensCircular"** • **"WGPBGCrysalStruct"** • **"WG3DSphere"** • **"WG3DEllipsoid"** • **"WG3DBlock"** • **"WG3DCylinder"** • **"G3DShape"**
Boolean WGMgr.DeleteObj *(Object ToDeleteObj)*	Deletes the object from the layout. As a parameter, this method only takes the object to be deleted.
Boolean WGMgr.ValidateID *(String ID)*	Given a string ID, this method returns **True** if no other waveguide has the given ID. If the given ID is already taken, this method returns **False**.
Object WGMgr.GetObjFromID *(String ID)*	Given a string ID, this method returns the waveguide object associated with the ID. If the ID was never associated with an object, this method returns **Null**.
String WGMgr.FindID *(String ID)*	Given a base-string ID, this method returns an ID with an integer appended to it. This new ID is guaranteed not to be taken by any other waveguide. For example, create three linear waveguides (without changing their IDs) throughout the layout. Now call this method with **"Linear"** as the parameter. It will return **"Linear4"**.
Boolean WGMgr.DeleteAll ( )	This method deletes all waveguide objects.
WGMgr.OutputMsg *(String Msg)*	Displays the string Msg passed in the **Output** window (**Notification** tab) .
WGMgr.UpdateVisAll ( )	This method updates the 3D Editor view and Tree View to match the current data. When dealing with many objects, this method might be expensive to execute.

Method signature	Description
	Please refrain from using this method inside a large loop - use the **UpdateVis** (**Object**) method instead. Common use of this method is after **WMGMgr**.**DeleteAll** ( ) or at the end of a script to update the whole visualization.
WGMgr.UpdateVis (*Object ToUpdateObj)*	This method updates the 3D Editor view and Tree View for a specific object **ToUpdateObj**. This method is efficient and can be used inside a large loop (e.g., to have a real-time visualization update while adding/removing/ modifying objects) . Common use of this method is after **WGMgr.CreateObj** and **WGMgr.DeleteObj** or after modifying parameters of an object.

The **Object** returned can use several methods defined below.

4.1.1 Waveguide Object

These methods are only performed on waveguide objects that are retrieved by either **CreateObj** or **GetObjFromID**.

4.1.1.1 Methods relative to regular waveguides

Method signature	Description
Boolean **SetPosition** *(Double XStart, Double YStart, Double XEnd, Double YEnd)*	Sets the start node position and the end node position. This method cannot be used for rods or rings because they only have a start position. When dealing with rods or rings, use the **SetStart** method.
Boolean **SetStart** *(Double XStart, Double YStart)*	Sets the start node position of the waveguide object. This can be used for all waveguides. Since rods and rings do not have an end node, this is the only method that can be used to set their position.

Method signature	Description
Boolean **SetEnd** *(Double XEnd, Double YEnd)*	Sets the end node position of the waveguide object. Cannot be used with rods and rings.
Boolean **SetAttr** *(String AttributeName, VARIANT AttributeValue)*	Sets an attribute of the waveguide object. The first parameter is a string specifying the attribute name. The second parameter is a **Variant**, which gives the parameter value. A variant is a union of types. It has a type (String, Double, Integer…) and a value. Every variable in VB script is treated as a variant. Every variant has a subtype. This subtype (String, Double, Integer…) is what needs to be passed to the different methods. For example, the **"WidthExpr"** attribute needs a **String** to be passed. This is really a variant of subtype **String**.
VARIANT **GetAttr** *(String AttributeType, String AttributeName)*	Retrieves the value of the given attribute. The first parameter is a **String** specifying the type the attribute has. The following string attribute types exist: "String", "Double", "Integer", and "Bool". Next there is a **String** identifying the name of the attribute. The return value is a **Variant** of whatever subtype was specified.
Boolean **SetStartExpr** *(String XExpr, String YExpr)*	Sets the start node expressions for a waveguide object.
Boolean **SetEndExpr** *(String XExpr, String YExpr)*	Sets the end node expressions for a waveguide object.
Boolean **GetStartExpr** *(String XExpr, String YExpr)*	Returns the start node expressions into the provided variables

Method signature	Description
Boolean **GetEndExpr** *(String XExpr, String YExpr)*	Returns the end node expressions into the provided variables
Boolean **GetPosition** *(Double XStart, Double YStart, Double XEnd, Double YEnd)*	Returns the start and end node positions into the provided variables
Boolean **GetStart** *(Double XStart, Double YStart)*	Returns the start node position into the provided variables
Boolean **GetEnd** *(Double XEnd, Double YEnd)*	Returns the end node position into the provided variables
Boolean **SetProfileName** *(String ProfileName)*	Sets the waveguide profile name.
Boolean **GetProfileName** *( )*	Gets the waveguide profile name.
Boolean **SetPositionEx** *(Double XStartOffset, Double YStartOffset, Double XEndOffset, Double YEndOffset, String XStartExpr, String YStartExpr, String XEndExpr, String YEndExpr)*	Extended version of **SetPosition**. Sets both the offsets and the expressions in one call.
Boolean ***SetStartEx (Double XStartOffset, Double YStartOffset, String XStartExpr, String YstartExpr)***	Extended version of **SetStart**. Sets both the offsets and the expressions of the start node.
Boolean **SetEndEx** *(Double XEndOffset, Double dYEndOffset, String XEndExpr, String YEndExpr)*	Extended version of **SetEnd**. Sets both the offsets and the expressions of the end node.
SetDefaultThicknessTaperMode *(Boolean bUseDefault)*	Toggle on/off default taper mode. In default taper mode, a default taper type is automatically selected

Method signature	Description
	for the waveguide. The default taper type is based on the profile type. For Fiber profiles, the default taper mode is 'Proportional'. For Channel profiles, the default taper mode is 'None'.
Boolean **GetDefaultThicknessTaperMode** ( )	Returns True if the waveguide is using default thickness tapering, False otherwise.
SetThicknessTaperType (*Integer TaperType*)	Set the thickness taper type (0=None, 1=Linear or 2=Proportional) . Proportional tapering can only be applied to waveguides using a Fiber profile.
Integer **GetThicknessTaperType ( )**	Returns the current tapering type (0=None, 1=Linear or 2=Proportional) .

The **attributes** are defined in the Waveguide Object attributes section.

4.1.1.2 Methods relative to 3D objects

These methods concern only objects of type "**WG3DSphere", "WG3DEllipsoid", "WG3DBlock" or "WG3DCylinder"**

Common methods

Method signature	Description
Boolean **SetPosition** (*Double X, Double Y, Double Z*)	Sets the 3D position of the shape by adjusting the position offsets. (in um)

Method signature	Description
Boolean **SetOrientation** (*Double Azimuth, Double Elevation, Double Twist*)	Sets the three resultant rotation angles to orient the shape, by adjusting the rotation angle offsets.
Boolean **SetOrientationOffset** (*Double Azimuth, Double Elevation, Double Twist*)	Sets the three rotation angle offsets. The resultant angle is the sum of the offset and expression.
Boolean **SetPositionExpr** (*String XStr,String YStr,String ZStr*)	Sets the expression strings for the position.
Boolean **SetOrientationExpr** (*String AzimuthStr, String ElevationStr, String TwistStr*)	Sets the three rotation angle expressions. The resultant angle is the sum of the offset and expression.
Boolean **GetPosition** (*Double X, Double Y, Double Z*)	Puts the position component values in X, Y, and Z
Puts the position component values in X, Y, and Z	
Boolean **GetOrientation** (*Double Azimuth, Double Elevation, Double Twist*)	Puts the orientation angle values in Azimuth, Elevation, and Twist
Boolean **GetPositionExpr** (*String XStr,String YStr,String ZStr*)	Puts the position expression strings in XStr, YStr, and ZStr
Boolean **GetOrientationExpr** (*String AzimuthStr, String ElevationStr, String TwistStr*)	Puts the orientation angle expression strings in AzimuthStr, ElevationStr, and TwistStr
Boolean **SetMaterial** (*String MaterialStr*)	Sets the material in MaterialStr

Method signature	Description
String **GetMaterial** ( )	Returns the name of the material assigned
Integer **GetNumOfClippingPlanes** ( )	Returns the number of clipping planes in the clipping plane list.
Boolean **CreateClippingPlane** (*Integer Index*)	Creates a new clipping plane and adds it to the end of the list of clipping planes. Its position in the list is returned in Index (starting from 0)
Boolean **SetClippingPlaneNormalExpr** (*Integer Index, String XStr, String YStr, String ZStr)*	Sets the X, Y, and Z expression strings for the normal vector of the clipping plane, whose position in the list is Index.
Boolean **SetClippingPlanePointExpr** (*Integer Index, String XStr, String YStr, String ZStr)*	Sets the X, Y, and Z expression strings for the position of the clipping plane, whose position in the list is Index
Boolean **GetClippingPlaneNormalExpr** (*Integer Index, String XStr, String YStr, String ZStr)*	Puts the X, Y, and Z expression strings for the normal vector of the clipping plane into XStr, YStr, and ZStr, respectively, for the clipping plane whose position in the list is Index
Boolean **GetClippingPlanePointExpr** (*Integer Index, String XStr, String YStr, String ZStr)*	Puts the X, Y, and Z expression strings for the position of the clipping plane into XStr, YStr, and ZStr, respectively, for the clipping plane whose position in the list is Index.
Boolean **GetClippingPlaneNormalVal** (*Integer Index, Double X, Double Y, Double Z)*	Puts the clipping plane normal vector component values into X, Y, and Z for the clipping plane whose normal vector the list is Index.

Method signature	Description
Boolean **GetClippingPlanePointVal** (*Integer Index, Double X, Double Y, Double Z)*	Puts the clipping plane position component values into X, Y, and Z for the clipping plane whose position in the list is Index.
Boolean **DeleteClippingPlane** *(Integer Index)*	Deletes the clipping plane found at position Index (starting from 0) in the list of clipping planes.

Sphere

Method signature	Description
Boolean **SetRadiusExpr** *(String RStr)*	Sets the sphere's radius expression string to RStr
String **GetRadiusExpr** *( )*	Returns the radius expression string.
Double **GetRadiusVal** *( )*	Returns the value of the sphere's radius

Ellipsoid

Method signature	Description
Boolean **SetAExpr** *(String AStr)*	Sets the A semi-axis expression string.
Boolean **SetBExpr** *(String BStr)*	Sets the B semi-axis expression string.
Boolean **SetCExpr** *(String CStr)*	Sets the C semi-axis expression string.
Boolean **GetAExpr** *(String AStr)*	Returns the value of the A semi-axis..
Boolean **GetBExpr** *(String BStr)*	Returns the value of the B semi-axis..
Boolean **GetCExpr** *(String CStr)*	Returns the value of the C semi-axis..

Block

Method signature	Description
Boolean **SetV1zExpr** *(String V1zStr)*	Sets the z component expression string for vector V1. This vector must always be aligned in the **Z-direction**, so the x and y components are constrained to 0.
Boolean **SetV2xExpr** *(String V2xStr)*	Vector V2 is constrained to the **X-Z plane**. Sets the **x** component expression string for vector V2,
Boolean **SetV2zExpr** *(String V2zStr)*	Vector V2 is constrained to the **X-Z plane**. Sets the **z** component expression string for vector V2,
Boolean **SetV3xExpr** *(String V3xStr)*	set the **x** component expression string for vector V3.
Boolean **SetV3yExpr** *(String V3yStr)*	set the **y** component expression string for vector V3.
Boolean **SetV3zExpr** *(String V3zStr)*	set the **z** component expression string for vector V3.
String ***GetV1xExpr*** *( )*	return the expression string for the **V1x** component
String ***GetV1yExpr*** *( )*	return the expression string for the **V1y** component
String ***GetV1zExpr*** *( )*	return the expression string for the **V1z** component
String ***GetV2xExpr*** *( )*	return the expression string for the **V2x** component
String ***GetV2yExpr*** *( )*	return the expression string for the **V2y** component
String ***GetV2zExpr*** *( )*	return the expression string for the **V2z** component
String ***GetV3xExpr*** *( )*	return the expression string for the **V3x** component

Method signature	Description
String ***GetV3yExpr*** *( )*	return the expression string for the **V3y** component
String ***GetV3zExpr*** *( )*	return the expression string for the **V3z** component
Double ***GetV1xVal*** *( )*	return the value of the **V1x** component
Double ***GetV1yVal*** *( )*	return the value of the **V1y** component
Double ***GetV1zVal*** *( )*	return the value of the **V1z** component
Double ***GetV2xVal*** *( )*	return the value of the **V2x** component
Double ***GetV2yVal*** *( )*	return the value of the **V2y** component
Double ***GetV2zVal*** *( )*	return the value of the **V2z** component
Double ***GetV3xVal*** *( )*	return the value of the **V3x** component
Double ***GetV3yVal*** *( )*	return the value of the **V3y** component
Double ***GetV3zVal*** *( )*	return the value of the **V3z** component

Cylinder

Method signature	Description
Boolean **SetRadiusExpr** *(String RStr)*	Sets the cylinder radius expression string to RStr
String **GetRadiusExpr** *( )*	Returns the cylinder radius expression string.
Double **GetRadiusVal** *( )*	Returns the cylinder radius value.

Method signature	Description
Boolean **SetHeightExpr** *(String HStr)*	Sets the height of the cylinder to HStr
String ***GetHeightExpr*** *( )*	Returns the cylinder height expression string.
Double ***GetHeightVal*** *( )*	Returns the cylinder height value.

4.1.1.3 Methods relative to 3D imported shapes

These methods concern only objects of type "**G3DShape**"

Method signature	Description
Boolean **Load** (*String Filename*)	Import a shape stored in an IGES file format.
Boolean **SetPosition** (*Double X, Double Y, Double Z*)	Sets the 3D position of the shape (in um) .
Boolean **GetPosition** (*Double X, Double Y, Double Z*)	Puts the position component values in X, Y, and Z
Boolean **SetOrientation** (*Double Azimuth, Double Elevation, Double Twist*)	Sets the three resultant rotation angles to orient the shape.
Boolean **GetOrientation** (*Double Azimuth, Double Elevation, Double Twist*)	Puts the orientation angle values in Azimuth, Elevation, and Twist.
Boolean **SetScale** (*Double ScaleX, Double ScaleY, Double ScaleZ*)	Scales the object in each coordinate direction.
Boolean **GetScale** (*Double ScaleX, Double ScaleY, Double ScaleZ*)	Puts the scale values in ScaleX, ScaleY, and ScaleZ
Boolean **SetMaterial** (*String MaterialStr*)	Sets the material in MaterialStr

Method signature	Description
String **GetMaterial** ( )	Returns the name of the material assigned

4.1.1.4 Waveguide Object attributes

Individual attributes can be accessed or set using **SetAttr(String AttributeName, Variant AttributeValue)** and **GetAttr(String AttributeType, String AttributeName)** functions. Here is a list of attributes for each object type:

Simple waveguides

Object type	Attribute name	Attribute type	Description
Linear waveguide	“WidthExpr”	"String"	Width of the linear waveguide
	"Depth"	"String"	Depth of the linear waveguide
Arc waveguide	“WidthExpr”	“String”	Arc width
	“RadiusExpr”	“String”	Arc radius
	"FixedRadius"	“Boolean”	If **False**, the radius equals the span. If**True**, the radius is equal to the radius expression
	“Depth”	“String”	Depth of the waveguide
	“Boundary Direction”	“Boolean”	Arc Orientation

Object type	Attribute name	Attribute type	Description
Elliptic waveguide	“MajorExpression”	“String”	Major expression of the ellipse
	“MinorExpression”	“String”	Minor expression of the ellipse
	"AngleExpression"	“String”	Tilt angle expression of the ellipse
	“MajorOffset”	“Double”	Major offset of the ellipse
	“MinorOffset”	“Double”	Minor offset of the ellipse
	"AngleOffset"	“Double”	Tilt angle offset of the ellipse
	“Depth”	“String”	Depth of the waveguide
Ring waveguide	“MajorOffset”	“Double”	Major offset of the ring
	“MinorOffset”	“Double”	Minor offset of the ring
	"AngleOffset"	“Double”	Tilt angle offset of the ring
	“MajorExpression”	“String”	Major expression of the ring
	“MinorExpression”	“String”	Minor expression of the ring
	"AngleExpression"	“String”	Tilt angle expression of the ring

Object type	Attribute name	Attribute type	Description
	“WidthExpr”	“String”	Width of the ring
	“Depth”	“String”	Depth of the waveguide

S-Bend waveguides

Object type	Attribute name	Attribute type	Description
S-Bend arc	“WidthExpr”	“String”	Width
	“RadiusExpr”	“String”	Radius Expression
	“RadiusType”	“Integer”	Fixed Radius (0) Min Radius (1)
	“NewPosBase”	“Integer”	Depth of the waveguide
	“RadiusUsed”	“Boolean”	Radius Used True for defining the waveguide in terms of radius
	“Depth”	“String”	Depth of the waveguide
	“AngleOffset”	“Double”	Rotation angle
S-Bend Sine	“WidthExpr”	“String”	Expression of the width
	“AngleOffset”	“Double”	Rotation angle
	“AngleOffset”	“Double”	Rotation angle
S-Bend Cosine	“WidthExpr”	“String”	Major expression of the ellipse

Object type	Attribute name	Attribute type	Description
	“Period”	“Double”	Period
	“Depth”	“String”	Depth of the waveguide
	“AngleOffset”	“Double”	Rotation angle
	“Amplitude”	“Double”	Set the value of the amplitude when *period = 1/2^n* This is the only case where the attribute should be used. The Amplitude must follow the period setting.

Taper waveguides

Object type	Attribute name	Attribute type	Description
Linear Taper waveguide	“StartWidthExpr”	"String"	Taper start width
	“EndWidthExpr”	"String"	Taper end width
	“Depth”	"String"	Depth of the taper waveguide
Parabolic Taper waveguide	“StartWidthExpr”	“String”	Taper start width
	“EndWidthExpr”	“String”	Taper end width
	"ParabolicType"	“Integer”	Parabolic type. Can be either 0 = convex or 1 = concave

Object type	Attribute name	Attribute type	Description
	“Depth”	“String”	Depth of the waveguide
Exponential Taper waveguide	“StartWidthExpr”	“String”	Taper start width
	“EndWidthExpr”	“String”	Taper end width
	"Alpha"	“Double”	Alpha value that defines the exponential taper
	“Depth”	“String”	Depth of the waveguide
S-Bend Arc Taper	“StartWidthExpr”	“String”	Start Width Expression
	“EndWidthExpr”	“String”	End Width Expression
	“RadiusExpr”	“String”	Radius Expression
	“RadiusType”	“Integer”	Fixed Radius (0) Min Radius (1)
	“NewPosBase”	“Integer”	New position based on Height (0) New position based on Length (1)
	“RadiusUsed”	“Boolean”	Radius Used True for defining the waveguide in terms of radius
	“Depth”	“String”	Depth of the waveguide
	“AngleOffset”	“Double”	Rotation angle

Object type	Attribute name	Attribute type	Description
S-Bend Sine Taper	“StartWidthExpr”	“String”	Expression of the start width
	“EndWidthExpr”	“String”	Expression of the end width
	“Depth”	“String”	Depth of the waveguide
	“AngleOffset”	“Double”	Rotation angle
S-Bend Taper Cosine	“StartWidthExpr”	“String”	Expression of the start width
	“EndWidthExpr”	“String”	Expression of the end width
	“Period”	“Double”	Period of the waveguide
	“Depth”	“String”	Depth of the waveguide
	“AngleOffset”	“Double”	Rotation angle
	“Amplitude”	“Double”	Set the value of the amplitude when *period = 1/2^n* This is the only case where the attribute should be used. The Amplitude must follow the period setting.

Polynomial waveguides

Object type	Attribute name	Attribute type	Description
Polynomial waveguide	“WidthExpr”	“String”	Width Expression
	“Coeff0”	“Double”	Coeff0
	“Coeff1”	“Double”	Coeff1
	“Coeff2”	“Double”	Coeff2
	“Coeff3”	“Double”	Coeff3
	“Coeff4”	“Double”	Coeff4
	“Coeff5”	“Double”	Coeff5
	“Coeff6”	“Double”	Coeff6
	“Coeff7”	“Double”	Coeff7
	“Coeff8”	“Double”	Coeff8
	“Coeff9”	“Double”	Coeff9
	“Coeff10”	“Double”	Coeff10
	“ClipUpOffset”	“Double”	Clip Up Offset
	“ClipDownOffset”	“Double”	Clip Down Offset
	“ClipUpExpression”	“String”	Clip Up Expression

Object type	Attribute name	Attribute type	Description
	“ClipDownExpression”	“String”	Clip Down Expression
	“Depth”	“String”	Depth
	“UpClipped”	“Boolean”	Indicates if the upper arm is clipped
	“DownClipped”	“Boolean”	Indicates if the lower arm is clipped

Lenses

Object type	Attribute name	Attribute type	Description
Elliptic lens	“MajorExpression”	“String”	Major expression of the ellipse
	“MinorExpression”	“String”	Minor expression of the ellipse
	“AngleExpression”	“String”	Orientation angle expression
	“AngleOffset”	“Double”	Orientation angle offset
	“MajorOffset”	“Double”	Major radius offset
	“MinorOffset”	“Double”	Minor radius offset
	“ClipRightOffset”	“Double”	Offset of the right clipping width

Object type	Attribute name	Attribute type	Description
	“ClipUpOffset”	“Double”	Offset of the upper clipping
	“ClipDownOffset”	“Double”	Offset of the lower clipping
	“ClipRightExpression”	“String”	Expression of the right clipping width
	“ClipLeftExpression”	“String”	Expression of the left clipping width
	“ClipUpExpression”	“String”	Expression of the upper clipping width
	“ClipDownExpression”	“String”	Expression of the lower clipping width
	“Depth”	“String”	Depth of the waveguide
	“RightClipped”	“Boolean”	Indicates if the right side of the lens is clipped
	“LeftClipped”	“Boolean”	Indicates if the left side of the lens is clipped
	“UpClipped”	“Boolean”	Indicates if the upper side of the lens is clipped
	“DownClipped”	“Boolean”	Indicates if the lower side of the lens is clipped

Object type	Attribute name	Attribute type	Description
Circular lens	“AngleExpression”	“String”	Angle expression of the ellipse
	“AngleOffset”	“Double”	Angle offset
	“ClipRightOffset”	“Double”	Offset of the right clipping width
	“ClipLeftOffset”	“Double”	Offset of the left clipping width
	“ClipUpOffset”	“Double”	Offset of the upper clipping
	“ClipDownOffset”	“Double”	Offset of the lower clipping
	“ClipRightExpression”	“String”	Expression of the right clipping width
	“ClipLeftExpression”	“String”	Expression of the left clipping width
	“ClipUpExpression”	“String”	Offset of the right clipping width
	“ClipDownExpression”	“String”	Offset of the left clipping width
	“Depth”	“String”	Depth of the waveguide
	“RadiusExpression”	“String”	Circular lens radius

Object type	Attribute name	Attribute type	Description
	“RightClipped”	“Boolean”	Indicates if the right side of the lens is clipped
	“LeftClipped”	“Boolean”	Indicates if the left side of the lens is clipped
	“UpClipped”	“Boolean”	Indicates if the upper side of the lens is clipped
	“DownClipped”	“Boolean”	Indicates if the lower side of the lens is clipped
Parabolic lens	“WidthExpr”	“String”	Width Expression
	“WidthOffset”	“Double”	Width Offset
	“RadiusExpr”	“String”	Radius Expression
	“RadiusOffset”	“Double”	Radius Offset
	“FixedRadius”	“Boolean”	Fixed Radius
	“ClipUpExpr”	“String”	Clip Up Expression
	“ClipDownExpr”	“String”	Clip Down Expression
	“ClipUpOffset”	“Double”	Clip Up Offset
	“ClipDownOffset”	“Double”	Clip Down Offset
	“Depth”	“String”	Depth

Object type	Attribute name	Attribute type	Description
	“UpperClipped”	“Boolean”	Indicates if the upper arm is clipped
	“LowerClipped”	“Boolean”	Indicates if the lower arm is clipped
Hyperbolic lens	“WidthExpr”	“String”	Width Expression
	“WidthOffset”	“Double”	Width Offset
	“RadiusExpr”	“String”	Radius Expression
	“RadiusOffset”	“Double”	Radius Offset
	“FixedRadius”	“Boolean”	Fixed Radius
	“ClipUpExpr”	“String”	Clipping width (+) Expression
	“ClipDownExpr”	“String”	Clipping width (-) Expression
	“ClipUpOffset”	“Double”	Clipping width (+) Expression
	“ClipDownOffset”	“Double”	Clipping width (-) Expression
	“EpsilonExpr”	“String”	Epsilon Expression
	“EpsilonOffset”	“Double”	Epsilon Offset
	“Depth”	“String”	Depth

Object type	Attribute name	Attribute type	Description
	“UpperClipped”	“Boolean”	Upper Clipped
	“LowerClipped”	“Boolean”	Lower Clipped

4.1.2 Photonic Crystal Structure Object

4.1.2.1 Methods that operate on the whole photonic crystal structure

Method signature	Description
Object **GetUnitCell** ( )	Returns the unit cell as a **PBGBasisCell** object which can be manipulated by its own API.
String **GetRotateAngleYExpr** ( )	Returns a string for the rotation about Y direction angle expression.
Boolean **SetRotateAngleYExpr** (*String*)	Sets the expression for the rotation about Y direction angle, to be interpreted in degrees.

4.1.2.2 Methods that operate on the unit cell level

Method signature	Description
Object **AddAtomToUnitCell** (*String type*)	Adds a waveguide to the unit cell atom list. The atom will be given a unique name in the layout, prefixed by "Atom". The following types are supported: • **“WGLinear”** • **“WGTaperLinear”** • **“WGTaperParabolic”** • **“WGTaperExp”** • **“WGArc”**

Method signature	Description
	• **“WGElliptic”** • **“WGRing"** • **“WGPolynomial”** • **“WGLensParabolic”** • **“WGLensHyperbolic”** • **“WGElliptic”** • **“WGLensElliptic”** • **“WGLensCircular”** • **“WG3DSphere”** • **“WG3DEllipsoid”** • **“WG3DBlock”** • **“WG3DCylinder”**
Boolean **RemoveAtomFromUnitCell** (*Object atom*)	Given a waveguide being used as an atom in the unit cell, this function removes it from the unit cell.
Object **GetFirstAtomFromUnitCell** ( )	Returns the first atom in the unit cell atom list, and sets the context for list traversal to the beginning. An atom is a waveguide. If there are no atoms in the basis cell, this function returns **Null**.
Object **GetNextAtomFromUnitCell** ( )	Returns the next atom in the unit cell atom list, if the list traversal context is not already at the end of the list, and increments the context. If a previous call already returned the last atom, then this function returns **Null**, and leaves the context as it is. To begin a new traversal, call **GetFirstAtomFromUnitCell** ( ) again.

4.1.2.3 Lattice manipulation methods

Method signature	Description
Boolean **SetLatticeType** (*String*)	Sets the lattice type. String can only be one of the following: • **"2D Rectangular"** • **"2D Hexagonal"** • **"3D Rectangular"** • **"3D Hexagonal"** • **"FCC"** • **"BCC"** • **"User Defined"**
String **GetLatticeType** ( )	Returns one of the above lattice types, as a string.
Double **GetLatticeScale** ( )	Returns the current value of the Scale (in um) .
String **GetLatticeScaleExpr** ( )	Returns the scale expression string.
Boolean **SetLatticeScaleExpr** (*String*)	Sets the scale expression string.
Integer **GetCountA** ( )	Returns the count for the specified lattice vector.
Integer **GetCountB** ( )	Returns the count for the specified lattice vector.
Integer **GetCountC ( )**	Returns the count for the specified lattice vector.
Boolean **SetCountA** (*Integer*)	Sets the count for the corresponding lattice vector.
Boolean **SetCountB** (*Integer*)	Sets the count for the corresponding lattice vector.
Boolean **SetCountC** (*Integer*)	Sets the count for the corresponding lattice vector.

Method signature	Description
Boolean **GetLatticeVectorA** (*Double, Double, Double*)	Returns the current values of the x, y, and z components of the corresponding lattice vector
Boolean **GetLatticeVectorB** (*Double, Double, Double*)	Returns the current values of the x, y, and z components of the corresponding lattice vector
Boolean **GetLatticeVectorC** (*Double, Double, Double*)	Returns the current values of the x, y, and z components of the corresponding lattice vector
Boolean **GetLatticeVectorAExpr** (*String*, *String*, *String*)	Returns the expression strings of the x, y, and z components of the corresponding lattice vector
Boolean **GetLatticeVectorBExpr** (*String*, *String*, *String*)	Returns the expression strings of the x, y, and z components of the corresponding lattice vector
Boolean **GetLatticeVectorCExpr** (*String*, *String*, *String*)	Returns the expression strings of the x, y, and z components of the corresponding lattice vector
Boolean **SetLatticeVectorAExpr** (*String, String, String*)	Sets the x, y, and z component expression strings for the corresponding lattice vector. Only valid for the **"User Defined"** type
Boolean **SetLatticeVectorBExpr** (*String, String, String*)	Sets the x, y, and z component expression strings for the corresponding lattice vector. Only valid for the **"User Defined"** type
Boolean **SetLatticeVectorCExpr** (*String, String, String*)	Sets the x, y, and z component expression strings for the corresponding lattice vector.

Method signature	Description
	Only valid for the **"User Defined"** type

4.1.2.4 Basis cell management methods

Method signature	Description
Object **GetBasisCell** (*Integer, Integer, Integer*)	Returns the basis cell object which has the lattice indices provided. If there is no basis cell at those coordinates, this function returns Null.
Boolean **ResetAllToUnitCell** ( )	Resets all basis cells to the unit cell (i.e. restores their atom lists, removing any modifications that may have been made to any atoms in that cell) .
Boolean **ResetUnmodifiedCellsToUnitCell** ( )	Resets all unmodified cells to the current unit cell.
Boolean **ResetBasisCellToUnitCell** (*Object*)	Given a basis cell object, this function restores its atom to be the same as the atoms in the unit cell.
Object **GetFirstBasisCell** ( )	Returns the first basis cell in the list of all basis cells, and sets the context for list traversal to the beginning.
Object **GetNextBasisCell** ( )	Returns the next basis cell in the list, if the list traversal context is not already at the end of the list, and increments the context. If a previous call already returned the last basis cell, then this function returns **NULL**, and leaves the context as it is. To begin a new traversal, call **GetFirstBasisCell** ( ) again

PBGBasisCell Object

These functions apply only to the basis cells which are returned by functions of the **PBGCrystalStruct**.

Method signature	Description
Integer **GetIndexA** ( )	Returns the integer index of the basis cell, corresponding to the number of spans in the direction of the A lattice vector, respectively. If this basis cell is the unit cell, this function returns 0.
Integer **GetIndexB** ( )	Returns the integer index of the basis cell, corresponding to the number of spans in the direction of the B lattice vector, respectively. If this basis cell is the unit cell, this function returns 0.
Integer **GetIndexC** ( )	Returns the integer index of the basis cell, corresponding to the number of spans in the direction of the C lattice vector, respectively. If this basis cell is the unit cell, this function returns 0.
Boolean **GetPosition** (*Double, Double, Double*)	Returns the 3D position of the origin of the basis cell. If this basis cell is the unit cell, this function returns 0 for all three components.
Boolean **SetOff** ( )	Hides the basis cell; its atoms will no longer be used in a simulation, nor appear in the layout designer.
Boolean **SetOn** ( )	Restores an atom which was previously hidden by **SetOff** ( ) , or hidden by modifications through the Photonic Crystal Structure dialogs or mouse editing features.

Method signature	Description
Boolean **IsOn** ( )	Returns **True** if the basis cell is not hidden, **False**, otherwise.
Boolean **IsModified** ( )	If the atoms in the basis cell have been modified to be different from those in the unit cell, this function returns **True**.
Boolean **SetModified** ( )	If an atom in the basis cell is modified through VBScript, the caller is obligated to call this function to mark the cell as modified. If this is not done, then the next call to **ResetUnmodifiedCellsToUnitCell** ( ) in the **PBGCrystalStruct** will reset the basis cell atoms to be the same as those in the unit cell. Call this function after changing any properties of atoms in this basis cell.
Object **GetFirstAtom** ( )	Returns the first atom in the basis cell atom list, and sets the context for list traversal to the beginning. An atom is a waveguide. If there are no atoms in the basis cell, this function returns **Null**.
Object **GetNextAtom** ( )	Returns the next atom in the list, if the list traversal context is not already at the end of the list, and increments the context. If a previous call already returned the last atom, then this function returns Null, and leaves the context as it is. To begin a new traversal, call **GetFirstAtom** ( ) again.

4.1.2.5 Miscellaneous methods

Method signature	Description
Integer **GetCurrentEditLayer** ( )	Returns the layer currently displayed in the layout.
Boolean **SetCurrentEditLayer** (*Integer*)	Sets the layer (i.e. the B index) to be displayed in the layout. This has no effect on the photonic crystal structure itself.

4.2 Parameter Manager

The Parameter Manager is a top-level item that is responsible for parameter scanning and simulation execution. It already exists at script-time and can be accessed from the FDTD-defined string identifier **"ParamMgr"**.

Method signature	Description
Boolean SetParam *(String Name, Double Value)*	Given a string name, sets the FDTD-defined parameter with the new value.
Boolean Simulate ( )	In the Designer, this method only updates all the expressions. It also validates the layout, providing a test-run before simulation. In the simulator, this method simulates the current layout.
VARIANT EvaluateExpr (String Expression)	**EvaluateExpr** can be used to evaluate any valid expression for the current layout. This expression can be retrieved from any object that contains an expression. This function can also be used to get the value of a project-defined parameter. It can be used like the **Get** counterpart of the above function, **ParamMgr.SetParam.**

Method signature	Description
	The return value is a **Variant** that will contain a double value if the expression can be evaluated successfully, otherwise it will contain a string to indicate an error.

4.3 Application Manager

AppMgr is a top level item that already exists at script-time and can be accessed from the FDTD-defined string identifier "**AppMgr".**

It handles generic functionality related to the application management, like output of messages, Layout Design data file location, etc.

Method signature	Description
OutputMsg *(String message)*	Sends the message text to the notification window. Useful for testing and debugging of the script. Only available in the layout Designer.
String **GetProjectFolderName ( )**	Returns the folder path name of the Layout Design data file location e.g.: **"C:\My Documents\My Project\"**
String **GetProjectFileName ( )**	Returns a name of the Layout Design data file with its extension e.g.: "**MyProject.fdt**"
String **GetAppFolder ( )**	Returns the folder path name of the location of Layout Design, or Simulator executables. It is the same as installation folder of the application binaries e.g.: "**C:\Program Files\Optiwave Software\OptiFDTD\bin**"
Boolean **RunsInSimulator ( )**	Verifies whether the script is executed within the simulator.

Method signature	Description
Integer **GetAppID ( )**	Returns an integer representing Code ID of the application executing the script. The function allows you to determine what application executes the script.

The application codes returned by **GetAppID ( )** are given in the following table:

Application code	Application
1001	OptiBPM Layout Designer
1051	OptiFDTD Designer
1101	OptiMode layout Designer
2001	OptiBPM 2D Simulator
2002	OptiBPM 3D isotropic Simulator
2003	OptiBPM 3D Anisotropic Simulator
2051	OptiFDTD 2D Simulator
2052	OptiFDTD 3D Simulator
2101	OptiMode Solver
3001	OptiBPM Analyzer
3051	OptiFDTD Analyzer
3101	OptiMode Analyzer

4.4 Point Source Manager

PointSourceMgr is a top-level item that already exists at script-time and can be accessed using the string identifier **"PointSourceMgr"**. The Point Source Manager is responsible for the creation, deletion, and access to point source objects.

Method signature	Description
Object **CreatePointSource** *(String label)*	Creates a Point Source object, if one with the specified label does not exist. If the ID already exists, this method returns the object that has the ID passed.
Boolean **DeletePointSource** (*Object PointSource*)	Deletes the specified Point Source object. Returns **True** if deletion was successful. Returns **False** if the specified object could not be deleted (i.e. it does not exist or it is not a Point Source object.
Boolean **ValidateID (** *String label*)	Validates whether the specified label is free - it has not been assigned to another object of the Layout
Object **GetPointSource (** *String label* **)**	Retrieves an existing Point Source object or creates one if it does not exist.
Boolean **DeleteAll ( )**	Deletes all existing Point Source objects. Returns **True** if successful or **False** if not.

The **Object** returned can use several methods defined in the Point Source Object section.

4.4.1 Point Source Object

These methods are only performed on Point Source objects that are retrieved by either **CreatePointSource** or **GetPointSource**.

Method signature	Description
Boolean **GetPosition** (double *horzVal, double vertVal*)	Retrieves coordinate values of the Point Source object and stores them in the arguments.
Boolean **GetPositionExt** (double *horzExpr, double horzOffset, double vertExpr, double vertOffset*)	Retrieves position components of the Point Source object coordinates using offsets and expressions
Boolean **SetPositionExt** (*double horzExpr, double horzOffset, double vertExpr,* double *vertOffset*)	Sets position components of the Point Source object coordinates using offsets and expressions
Double **GetDepth** ( )	Retrieves double value of Point Source object depth.
String **GetDepthExpr** ( )	Retrieves the string that represents the Point Source object depth expression.
Boolean **SetDepthExpr** (*String depthExpr*)	Sets the Point Source object depth expression. Returns **True** if the coordinates have been set successfully, **False** otherwise.
Double **GetWavelength** ( )	Retrieves the double value of the Point Source wavelength (in um)
String **GetWavelengthExpr** ( )	Retrieves a string that represents the Point Source wave wavelength expression.
Boolean **SetWavelengthExpr** (*String wavelengthExpr*)	Sets the expression of the Point Source wavelength. Returns **True** if the wavelength has been set successfully, **False** otherwise.
Double **GetAmplitude** ( )	Retrieves the double value of the Point Source amplitude.

Method signature	Description
Boolean **SetAmplitude** (*Double dAmplitude*)	Sets the double value of the Point Source amplitude.
Enable (*Boolean bEnable*)	Enables or disable the Point Source. **True** = enable, **False** = disable
Boolean **IsEnabled** ( )	Returns **True** if the Point Source is enabled, **False** otherwise.
Boolean **SetAsKeyPointSource** ( )	Designates the Point Source to be a Key Point Source. There can be only one Key Source Point. If another point source had been designated as the key one, it becomes a "regular" one. The first Point Source is set as the key by default.
Boolean **IsKeyPointSource** ( )	Returns **True** if the Point Source was set as the key, **False** otherwise
Boolean **SetField3D** (*Integer nFieldComponent*)	Specifies which field component of Point Source is going to be excited during 3D simulations. **0 = Ex. 1 = Ey, 2 = Ez.**
Integer **GetField3D** ( )	Returns a flag describing what field component of Point Source is to be excited during 3D simulations. **0 = Ex. 1 = Ey, 2 = Ez.**
Boolean **SetWaveformType** (*Integer nWaveFormType*)	Specifies the waveform type of the Point Source.**0 =CW, 1 = SMGP.**
Integer **GetWaveformType** ( )	Returns an integer flag describing the Point Source waveform type. .**0 =CW, 1 = SMGP.**
Double **GetTimeWidth** ( )	Returns the double value of the Sine-Modulated Gaussian Pulse full width at half maximum (FWHM)

Method signature	Description
String **GetTimeWidthExpr** ( )	Returns a string that represents the Sine-Modulated Gaussian Pulse FWHM expression.
Boolean **SetTimeWidthExpr** (String *widthExpr*)	Sets the expression of the Sine-Modulated Gaussian Pulse FWHM. Returns **True** if the half width has been set successfully, **False** otherwise.
Double **GetTimeOffset** ( )	Returns double value of the time delay of the Sine-Modulated Gaussian Pulse (in sec)
String **GetTimeOffsetExpr** ( )	Returns a string representing expression of the time delay of the Sine-Modulated Gaussian Pulse.
Boolean **SetTimeOffsetExpr** (*String timeoffsetExpr*)	Sets expression of the time delay of the Sine-Modulated Gaussian Pulse. Returns **True** if the time offset has been set successfully, **False** otherwise.
UseDefaultTimeWidthAndOffset (*Boolean bFlag*)	Set the flag to use the default values of the FWHM and time offset of the Sine-Modulated Gaussian Pulse. **True** = use default values. Needn't call **SetTimeWidthExpr** and **SetTimeOffsetExpr** to set those values. **False** = the values have to be set by calling **SetTimeWidthExpr** and **SetTimeOffsetExpr.**
Boolean **SetInitialPhase** (*Double phase*)	Sets the Initial Phase value of the input wave (in degrees) . The input wave will start propagating with the specified phase.

4.5 Input Plane Manager

The **Input Plane Manager** object is a top-level-item. This object exists the whole time that the script is running. It can be accessed during scripting by the string identifier:**“InputPlaneMgr”**.

Method signature	Description
Object **CreateObj** (*String InputType, String FieldType, String label)*	The parameters are the same as **CreateInputObj**, with *strDirection* set to "**Vertical**". If the label already exists, the existing object is returned.
Boolean **DeleteObj** *(Object ToDeleteObj)*	Deletes the object from the layout. As a parameter, this method only takes the object to be deleted.
Boolean **ValidateID** *(String label)*	Given a label, this method returns **True** if no other input plane has the given label. Otherwise, if the given label is already taken, this method returns **False**.
Object **GetObjFromID** *(String label)*	Given a label, this method returns the input plane object associated with the label. If the label was never associated with an object, this method returns **Null**.
Object **CreateInputObj** (*String InputType, String FieldType, String label, String strDirection)*	Creates a new input plane object. The first parameter is the Input type that can be • **"Continuous"** • **"Pulse"** The second parameter "*FieldType*" is for the field type that can be • **"Mode"** • **"Gaussian"** • **"Rectangular"**

Method signature	Description
	• **"UserDefined"** The third parameter is the label that the input plane will be identified with. If the label already exists, the existing object is returned. The fourth parameter specifies that the input plane is either • **"Horizontal"** • **"Vertical"** • **"Y-Direction"**
String **FindID** *(String label)*	Given a base label, this method returns a label with an integer appended to it. This new label is guaranteed not to be taken by any other input plane.
Boolean **DeleteAll** *( )*	Deletes all input planes in the layout.
SetKeyInputPlane (*Object pInputPlane)*	Set the input plane specified by *pInputPlane* as the key input plane
Object **GetKeyInputPlane** *( )*	Returns an object that represents the key input plane.

The **Object** returned can use methods defined in Input Plane Object

4.5.1 Input Plane Object

The input plane object is returned after a **CreateInputObj** or **GetObjFromID** call.

4.5.1.1 Common methods

Method signature	Description
Boolean **IsEnabled** ( )	Returns **True**, if the Input Plane is enabled (set to active) , else, **False**.
SetEnabled *(Boolean bEnabled)*	Sets the Input Plane to be enabled. This function works properly only if all the input planes have already been created. When multiple input planes are created, the very last one created is set to enabled automatically. If a plane other than the last one created is to be enabled, this function should be called last in the sequence. Only one input plane can be enabled at a time.
SetZPosition *(Double zPos)*	Sets the Input Plane position along the Z-axis (in um) . Works only for Z-direction input planes
SetPosition *(Double Pos)*	Sets the Input Plane position along its axis (in um) .
Double **GetZPosition ( )**	Returns the position of the Input Plane on the Z-axis.
SetDirection *(String strDirection)*	Sets the direction of the input plane. Valid parameters: • **“Forward”** • **“Backward”.** Input planes are automatically created with direction set to “Forward”.
Boolean **SetCenterPos** *(String strVal)*	Sets the center position value (in um) , automatically sets the **IsCenterPosAuto** flag to **FALSE**. Only valid for 2D input planes.

Method signature	Description
RefreshInputField *(String RefreshInputField)*	Refreshes the Input Field values.
Boolean **SetWidth** *(String strVal)* *Note: Formerly named* **SetHalfWidth** *(String strVal)*	Sets the Rectangular field half-width or the Gaussian field width (in um) - only valid for 2D input planes. **Important Note (Gaussian field only)** : The width of the Gaussian field is now defined by its 1/e^2 width (it was previously defined by its half-width) . To match the half-width setting, set the field width to be *2 * Sqr (2)* times larger than the half width. For example if the half-width had been previously defined as 0.55 um, set the new value as follows: *wx = 0.55 * 2 * Sqr (2) .* *InputPlane1.SetWidth CStr (wx)* or directly *InputPlane1.SetWidth "0.55 * 2 * sqrt (2) "*
Boolean **SetTiltingAngle** *(String strVal)*	Sets the Tilting angle value in degrees. Only valid for 2D input planes.
SetAmplitudeOrPower *(String strType, String strVal)*	Sets the Transverse Input Type and its value. Valid types: • **“Amplitude”** • **“Power”**
AddModalWG *(String strID)*	Adds a waveguide to the list of selected waveguides for a Mode Input Type of Input Plane.
RemoveModalWG *(String strID)*	Removes a waveguide from the selected waveguides list. (“Mode” transverse field type of Input Plane.)
SetRefLocal ( )	Sets the Gaussian or Rectangular type of input plane’s reference index to “Local".

Method signature	Description
SetRefModeIdx *(Long nVal)*	Sets the mode number index for a Mode transverse input type plane and selects it. This is not the index as it appears in the list of modes dialog.
SetRefUser *(Double dReal, Double dImaginary)*	Sets the Gaussian or Rectangular type of input plane's reference index to "User". Adds the appropriate real and imaginary values.
SetModeIdx *(Long nModeNumber)*	Sets the mode number index for a Mode transverse input type plane and selects it. This is not the index as it appears in the list of modes dialog.
String **GetRefType** ( )	This function will return the reference type for a Gaussian or Rectangular type of input plane, else it will return an empty string. Valid types: • **"Mode"** • **"Local"** • **"User"**
String **GetLabel** ( )	Returns the Input Plane string label
SetInitialPhase (*Double dPhase* **)**	Sets the Initial Phase value of the input wave (in degrees) . The input wave will start propagating with the specified phase.
Double **GetInitialPhase ( )**	Returns the value of Initial Phase (in degrees) .
Boolean **GetDirection** *( )*	Retrieves the set direction. Input planes are created by default with this set to "**Forward**". The other valid setting is "**Backward**"

Method signature	Description
SetWaveLength *(String strExpr)*	Sets the wavelength for the input plane. If this function is not called, a default value of 1.5 is set.
String **GetWaveLength** *( )*	Returns the wavelength expression string.
GetAmplitudeOrPower *(String strType, String strVal)*	Retrieves the amplitude type (Amplitude or Power), and the value.
String **GetTransverseFieldType** *( )*	Returns the Transverse field type. Valid values: • **"Mode"** • **"Gaussian"** • **"Rectangular"** • **"UserDefined"**
String **GetInputType** *( )*	Returns the Input Type. This will be either "**Continuous**" or "**Pulse**".
Boolean **SetTimeWidth** *(String strTimeWidth)*	Sets the expression of the Sine-Modulated Gaussian Pulse FWHM. Returns **True** if the half width has been set successfully, **False** otherwise.
Boolean **SetTimeOffset** *(String strTimeOffset)*	Sets the time offset of the Sine-Modulated Gaussian Pulse.
String **GetTimeWidth ( )**	Returns a string that represents the Sine-Modulated Gaussian Pulse FWHM expression.
String **GetTimeOffset ( )**	Gets the time offset of the Sine-Modulated Gaussian Pulse.
UseDefaultTimeWidthAndOffset (*Boolean bFlag*)	Set the flag to use the default values of the FWHM and time offset of the Sine-Modulated Gaussian Pulse.

Method signature	Description
	True = use default values. Needn't call **SetTimeWidth** and **SetTimeOffset** to set those values. **False** = the values have to be set by calling **SetTimeWidth** and **SetTimeOffset.**
Boolean **ExportRawModalFields** (String strPath)	Will export modal fields to the specified path given as strPath.

4.5.1.2 Methods relative to 2D transverse input planes

Method signature	Description
SetFieldFile *(String strType, String strFile)*	Sets the input field data. It will assign the field file strFile to the component specified by strType. The contents of the strType can be "**Ey**" or "**Hx**".
GetFieldFile *(String strType, Variant strFile)*	Returns the input field file name via strFile, for the field component specified by strType. The contents of the strType can be "**Ey**" or "**Hx**".
Boolean **SetAutoCalWG** *(String strWGLabel)*	Sets a waveguide with the label **strWGLabel** as the reference waveguide for the calculation of "Center Position", "Half Width"and "Tilting Angle". This reference waveguide is for both 2D and 3D. This method will return **TRUE** if the setting succeeded, otherwise returns **FALSE**.
String **GetCenterPos ( )**	Retrieves the Center Position for a Gaussian or Rectangular transverse field type of Input Plane. It will return an empty string, if it is called on a Mode type of Input Plane.

Method signature	Description
String **GetWidth ( )** *Note:* Formerly named **GetHalfWidth** *(String)*	Retrieves the half width for a Rectangular transverse field type or the 1/e^2 full width for the Gaussian transverse field type. It will return an empty string, if it is called on a Mode type of Input Plane.
String **GetTiltingAngle ( )**	Returns the Tilting Angle of a Gaussian or Rectangular type of Input Plane. It will return an empty string if the input plane is either a Mode type, or if the Tilting Angle was set to auto.
Long **GetRefModeIdx ( )**	If the Input Plane is either Gaussian or Rectangular, and has its Reference index set to “Mode”, this function returns the mode index, (its position) as it would appear in the modes list dialog.
Long **GetModeIdx** ( )	If the Input Plane is a Mode type, a Mode index number can be set. This function does not return a calculated value, but returns the mode index, (its position) as it would appear in the modes list dialog.
Boolean **GetRefUserVals** *(Double dReal, Double dImaginary)*	Returns **TRUE**, retrieves the real and imaginary values if the Reference index was set to Modal. If the Reference type is not set to “User”, returns **FALSE**.

4.5.1.3 Methods relative to 3D transverse input planes

Method signature	Description
SetAmplitudeExpr3D *(String strVal)*	Sets the transverse input type to "**Amplitude**", and set its value to strVal that represents an expression.

Method signature	Description
SetPowerExpr3D *(String strVal)*	Sets the transverse input type to "**Power**", and set its value to strVal that represents an expression.
String **GetAmplitudeExpr3D** *( )*	Returns the expression that represents the value of the input Amplitude.
Double **GetAmplitude3D** *( )*	Returns the value of the input Amplitude.
String **GetPowerExpr3D** *( )*	Returns the expression that represents the value of the input Power.
Double **GetPower3D** *( )*	Returns the value of the input Power.
Boolean **SetCenterPosExpr3D** *(String strExpX, String strExpY)*	This method is for the input plane with input field transverse type that is Gaussian or Rectangular. This method will automatically set the related Auto flag to **FALSE** if the string is not empty; it will set the Auto flag to **TRUE** if a string is empty.
Boolean **GetCenterPosExpr3D** *(Variant strX, Variant strY)*	Retrieves the Center Position expression for a Gaussian or Rectangular transverse field type of Input Plane
Boolean **GetCenterPos3D** *(Variant dX, Variant dY)*	Retrieves the Center Position for a Gaussian or Rectangular transverse field type of Input Plane.
SetCenterPosAuto3D *(Boolean bXAuto, Boolean bYAuto)*	Sets the Auto flag related to the Center Position to the value as **bXAuto** and **bYAuto**. A waveguide must be set by calling the method **SetAutoCalWG** if an Auto flag is set to **TRUE**.
Boolean **SetWidthExpr3D** *(String strExpX, String strExpY)*	Sets the Rectangular field half width or the Gaussian field width (in um) .

Method signature	Description
Note: Formerly named ***SetHalfWidthExpr3D*** *(String strExpX, String strExpY)*	**Important Note (Gaussian field only)** : The width of the Gaussian field is now defined by its 1/e^2 width (it was previously defined by its half-width) . To match the half-width setting, set the field width to be *2 * Sqr (2)* times larger than the half width. For example if the half-width had been previously defined as 0.55 um (for both X and Y) , set the new values as follows: *wx = 0.55 * 2 * Sqr (2) .* *wy = 0.55 * 2 * Sqr (2) .* *InputPlane1.SetWidthExp3D CStr (wx) , CStr (wy)* or directly *InputPlane1.SetWidthExp3D "0.55 * 2 * sqrt (2) ", "0.55 * 2 * sqrt (2) "* This Boolean operation will automatically set the related Auto flag to **FALSE** if the string is not empty.The related Auto flag will be set to **TRUE** if a string is empty.
Boolean **GetWidthExpr3D** *(Variant strX, Variant strY)* *Note: Formerly named* **GetHalfWidthExpr3D** *(Variant strX, Variant strY)*	The strX and strY will contain the expressions for the 3D half width for Rectangular or full width (1/e^2) for a Gaussian transverse field type of Input Plane.
Boolean **GetWidth3D** *(Variant dX, Variant dY)* *Note: Formerly named* **GetHalfWidth3D** *(Variant dX, Variant dY)*	The dX and dY will contain the value of the 3D half width for Rectangular or full width (1/e^2) for a Gaussian transverse field type of Input Plane
SetWidthAuto3D *(Boolean bXAuto, Boolean bYAuto)*	Sets the Auto flag related to the width to the value as **bXAuto** and **bYAuto**. A waveguide must be set by calling the method **SetAutoCalWG** if an Auto flag is set to **TRUE**.

Method signature	Description
Note: Formerly named **SetHalfWidthAuto3D** *(Boolean bXAuto, Boolean bYAuto)*	
SetFieldFile3D *(String strType, String strFile)*	Sets the input field data. It will assign the field file strFile to the component specified by strType. The contents of the strType can be "**Ex**", "**Ey**", "**Hx**" or "**Hy**".
GetFieldFile3D *(String strType, Variant strFile)*	Returns the input field file name via strFile, that is for the field component specified by strType. The contents of the strType can be "**Ex**", "**Ey**", "**Hx**" or "**Hy**".
SetLXPolarization3D *( )*	Sets the polarization type to **"LinearX**" if the input field transverse type is set to Gaussian or Rectangular.
SetLYPolarization3D *( )*	Sets the polarization type to **"LinearY**" if the input field transverse type is set to Gaussian or Rectangular.
SetLZPolarization3D *( )*	Sets the polarization type to **"LinearZ**" if the input field transverse type is set to Gaussian or Rectangular.
SetGLPolarization3D *(String theta)*	Sets the polarization type to "**LinearTheta**" Sets the polarization angle expression to "**theta**" if the input field transverse type is set to Gaussian or Rectangular.
SetRHPolarization3D *( )*	Sets the polarization type to "**RightHandCircular**" if the input field transverse type is set to Gaussian or Rectangular.

Method signature	Description
SetLHPolarization3D *( )*	Sets the polarization type to "**LeftHandCircular**" if the input field transverse type is set to Gaussian or Rectangular.
Boolean **IsLXPolarization3D** *( )*	Returns **TRUE** if the polarization type is set to "**LinearX**".
Boolean **IsLYPolarization3D** *( )*	Returns **TRUE** if the polarization type is set to "**LinearY**".
Boolean **IsLZPolarization3D** *( )*	Returns **TRUE** if the polarization type is set to "**LinearZ**".
Boolean **IsGLPolarization3D** *( )*	Returns **TRUE** if the polarization type is set to "**LinearTheta**".
IsRHPolarization3D ( )	Returns **TRUE** if the polarization type is set to "**RightHandCircular**".
IsLHPolarization3D ( )	Returns **TRUE** if the polarization type is set to "**LeftHandCircular**".
String **GetLinearThetaExpr3D** *( )*	Returns the expression for the polarization angle "**Theta**".
Double **GetLinearTheta3D** *( )*	Returns the value of the polarization angle expression "**Theta**".
Boolean **SetAutoCalWG** *(String strWGLabel)*	Sets a waveguide with the label **strWGLabel** as the reference waveguide for the calculation of "Center Position", "Half Width"and "Tilting Angle". This reference waveguide is for both 2D and 3D. This method will return **TRUE** if the setting succeeded, otherwise returns **FALSE**.

Method signature	Description
String **GetAutoCalWG** *( )*	Returns the label of the reference waveguide for the calculation of "Center Position", "Half Width"and "Tilting Angle".
Boolean **Is3DInjectionModal** *( )*	Returns **TRUE** if the effective refractive index type is set to the type of "**Modal**".
Boolean **Is3DInjectionLocal** *( )*	Returns **TRUE** if the effective refractive index type is set to the type of "**Local**".
Boolean **Is3DInjectionUserDefined** *( )*	Returns **TRUE** if the effective refractive index type is set to the type of "**UserDefined**".
Boolean **SetTiltingAngleExpr3D** *(String strExpr)*	Sets the Tilting angle represented by the expression strExpr if the input field transverse type is Gaussian or Rectangular.
String **GetTiltingAngleExpr3D** *( )*	Returns the Tilting Angle expression if the input field transverse type is Gaussian or Rectangular. It will return an empty string if the input plane is either a Modal type, or a User Defined type.
Double **GetTiltingAngle3D** *( )*	Returns the value of the Tilting Angle expression.
SetTiltingAngleAuto3D *(Boolean bAuto)*	Sets the Auto flag for the Tilting Angle to the value as bAuto. A waveguide must be set by calling the method **SetAutoCalWG** if an Auto flag is set to **TRUE**. Not supported by Y-Direction input plane.
Boolean **SetTiltingAngleExpr3DV2** *(String strExprX, String strExprY)*	Sets the tilting angles represented by the expression strExprX and strExprY if the input field transverse type is Gaussian or Rectangular. Where X is the theta tilt angle

Method signature	Description
	(away from axis) and Y is the phi tilt angle (around the axis).
Boolean **GetTiltingAngleExpr3DV2** *(Variant strX, Variant strY)*	Retrieves the expressions of the tilting angles. Where X is the theta tilt angle (away from axis) and Y is the phi tilt angle (around the axis).
Boolean **GetTiltingAngle3DV2** *(Variant dX, Variant dY)*	Retrieves the double values of the tilting angles. Where X is the theta tilt angle (away from axis) and Y is the phi tilt angle (around the axis).
SetRefLocal3D *( )*	Sets the effective refractive index to "**Local**" if the input field transverse type is Gaussian, Rectangular or User Defined type.
SetModeIdx3D *(Long nModeIdx)*	Sets and selects the modal index as nModeIdx.
Long **GetModeIdx3D** *( )*	Returns the selected mode index.
AddModalWG3D *(String strID)*	Adds the waveguide that has the label as strID to the waveguide list
RemoveModalWG3D *(String strID)*	Removes the waveguide that has the label as strID from the waveguide list
SetRefUser3D *(Double dReal, Double dImaginary)*	This method is for the input plane that input field transverse type is Gaussian, Rectangular or User Defined. It will set the type of the effective refractive index to "User", and adds the appropriate real, imaginary values.
Boolean **GetRefUserVals3D** *(Variant dReal, Variant dImaginary)*	Returns **TRUE**, if the reference type is set to "User" and retrieves the real and imaginary values. Otherwise, returns **FALSE**.

Method signature	Description
Boolean **IsAmplitude3D (** *String strIsAmplitude 3D*)	Returns **TRUE** if amplitude value is 3D, if not, returns **FALSE**.
Boolean **IsPower3D (** *String strIsPower 3D*)	Returns **TRUE** if power value is 3D, if not, returns **FALSE**.

4.6 Wafer Parameters Manager

The **Wafer Parameters Manager** is a top-level-item. This object exists the whole time that the script is running. It can be accessed during scripting by the string identifier:**"WaferParams"**. This object is responsible for accessing and setting wafer properties.

Method signature	Description
SetLengthExpr *(String strExpr)*	The wafer length (Z-axis) is set to strExpr
SetWidthExpr *(String strExpr)*	The width (X-axis) of the wafer is set to strExpr
Double **GetLength** *( )*	Returns the length (Z-axis) of the wafer.
Double **GetWidth** *( )*	Returns the width (X-axis) of the wafer.
SetMaterial *(String strMaterialName)*	Sets the 2D wafer material as strMaterialName
String **GetMaterial** *( )*	Returns the 2D wafer material.
String **GetLengthExpr** *( )*	Returns the wafer length (Z-axis) expression.
String **GetWidthExpr** *( )*	Returns the wafer width (X-axis) expression.
SetSubstrateExpr *(String strExpr)*	The thickness (Y-axis) of the substrate is set to strExpr

Method signature	Description
SetCladdingExpr *(String strExpr)*	The thickness (Y-axis) of the cladding is set to strExpr.
Double **GetSubstrateThickness** *( )*	Returns the thickness (Y-axis) of the substrate.
Double **GetCladdingThickness** *( )*	Returns the thickness (Y-axis) of the cladding.
SetSubstrateMaterial *(String strMaterialName)*	The material for the substrateis set to strMaterialName
SetCladdingMaterial *(String strMaterialName)*	The material for the cladding is set to strMaterialName
String **GetSubstrateMaterial** *( )*	Returns the material for the substrate
String **GetCladdingMaterial** *( )*	Returns the material for the cladding.

4.7 Simulation Parameters Manager

There are two objects object for setting 2D and 3D simulation parameters, "**SimCtrlParams2D**" and "**SimCtrlParams3D**". All of the following methods can be executed from either object, however if the method only applies to the other object it will do nothing. For example, **SimCtrlParams2D.SetMeshSizeY ( )** does nothing, since only X and Z dimensions are used in 2D simulations.

Method signature	Description
Long **GetMeshCellNumX** *( )*	Retrieves the number of cells in the X direction.
Long **GetMeshCellNumY** *( )*	Retrieves the number of cells in the Y direction.
Long **GetMeshCellNumZ** *( )*	Retrieves the number of cells in the Z direction.

Method signature	Description
Double **GetMeshSizeX** *( )*	Retrieves mesh size in meters in the X direction (in um)
Double **GetMeshSizeY** *( )*	Retrieves mesh size in meters in the Y direction (in um)
Double **GetMeshSizeZ** *( )*	Retrieves mesh size in meters in the Z direction (in um)
SetMeshSizeX *(Double dMeshSizeX)*	Sets the mesh size in X direction (in um)
SetMeshSizeY *(Double dMeshSizeY)*	Sets the mesh size in Y direction (in um)
SetMeshSizeZ *(Double dMeshSizez)*	Sets the mesh size in Z direction (in um)
Long **GetTimeSteps** *( )*	Gets the number of time steps the simulation will run, excluding finalization.
SetTimeSteps *(Long nTimeSteps)*	Sets the number of time steps the simulation will run, excluding finalization.
Double **GetTimeStepSize** *( )*	Retrieves the time step size used in the simulation (in sec)
SetTimeStepSize *(Double dTimeStepSize)*	Sets the time step size used in the simulation (in sec)
SetAutoCalcTimeStepSize *(Boolean)*	Sets the time step size automatic calculation flag. Pass in **True** to set it. Pass in **False** to turn it off.
Long **GetAnisotropicPMLLayers** *( )*	Retrieves the number of Anisotropic PML layers used in the simulation.

Method signature	Description
SetAnisotropicPMLLayers *(Long nPMLLayers)*	Sets the number of Anisotropic PML layers used in the simulation.
Double **GetAnisotropicPMLReflectionCoeff** *( )*	Gets the Anisotropic PML reflection coefficient.
SetAnisotropicPMLReflectionCoeff *(Double dValue)*	Sets the Anisotropic PML reflection coefficient.
Double **GetAnisotropicPMLRealTensorParam** *( )*	Gets the Anisotropic PML real tensor parameter.
SetAnisotropicPMLRealTensorParam *(Double dValue)*	Sets real tensor value for Anisotropic PML layers.
Double **GetAnisotropicPMLGradingPolyPower** *( )*	Gets the power of grading polynomial.
SetAnisotropicPMLGradingPolyPower *(Double dValue)*	Sets the power of grading polynomial.
Boolean **IsTE** *( )*	Returns **True** if the **2D** simulation type is **TE**.
Boolean **IsTM** *( )*	Returns **True** if the **2D** simulation type is **TM**.
Boolean **Is3D** *( )*	Returns **True** if the simulation type is **3D**.
SetTE *( )*	Set the **2D** simulation type to **TE**
SetTM *( )*	Set the **2D** simulation type to **TM**
RunUntilStopped *( )*	Simulation will run till user stops it.

Method signature	Description
Set2DsimulationType *(String strVal)*	Same functionality as SetTE and SetTM. Pass either "**TE**" or "**TM**" to change the **2D** simulation type
String **Get2DsimulationType** *( )*	Returns a string indicating the **2D** simulation type "**TE**" or "**TM**"
SetBoundaryX (*String XNeg, String XPos*)	Sets the X boundary conditions. XNeg is for the negative X boundary, XPos for the positive X boundary. Accepted values are: • **"PML"** • **"PEC"** • **"PMC"** • **"PBC"**
SetBoundaryY (*String YNeg, String YPos*)	Sets the Y boundary conditions. YNeg is for the negative Y boundary, YPos for the positive Y boundary. Accepted values are: • **"PML"** • **"PEC"** • **"PMC"** • **"PBC"**
SetBoundaryZ (*String ZNeg, String ZPos*)	Sets the Z boundary conditions. ZNeg is for the negative Z boundary, ZPos for the positive Z boundary. Accepted values are: • **"PML"** • **"PEC"** • **"PMC"** • **"PBC"**
UseNonuniformMesh (*Boolean bflag*)	Toggles on/off the 3D non-uniform mesh

Method signature	Description
SetNonuniformMeshGenFlag (*Long direction, Long flag*)	Sets the 3D non-uniform mesh type for the specified direction. Valid directions are: **0 = X, 1 = Y, 2 = Z** Valid flag values are: • **0 = automatic** • **1 = from file** • **2 = predefined sections**
Long **GetNonuniformMeshGenFlag** (*Long direction*)	Gets the 3D non-uniform mesh type for the specified direction. Valid directions are: **0 = X, 1 = Y, 2 = Z** Return values are: • **0 = automatic** • **1 = from file** • **2 = predefined sections**
Boolean ***SetNonuniformMeshFileName*** (Long direction, String filename)	Sets the 3D non-uniform mesh file name for the specified direction. **SetNonuniformMeshGenFlag** ( ) must be used beforehand to set the proper flag. Valid directions are: **0 = X, 1 = Y, 2 = Z**
String ***GetNonuniformMeshFileName*** *(Long direction)*	Gets the 3D non-uniform mesh file name for the specified direction. Valid directions are: **0 = X, 1 = Y, 2 = Z**
Boolean **SetNonuniformMeshCtrlParam** *(Long direction, Double max, Double min, Double rate)*	Sets the mesh generation control parameters for the specified direction. Valid directions are: **0 = X, 1 = Y, 2 = Z** **min** controls the minimum mesh size, **max** the maximum mesh size and **rate** the increase rate

Method signature	Description
GetNonuniformMeshCtrlParam *(Long direction, Variant max, Variant min, Variant rate)*	Gets the mesh generation control parameters for the specified direction.
Boolean **AddNonuniformMinMeshSection** *(Long direction, Double start, Double end)*	Adds a predefined section of minimum mesh size for the specified direction. Valid directions are: **0 = X, 1 = Y, 2 = Z** **start** controls the start position and **end** controls the end position of the minimum mesh size section (in um)
Boolean **GetNonuniformMinMeshSection** *(Long direction, Long index, Variant start, Variant end)*	Returns the predefined section of minimum mesh size given by **index** for the specified direction. Valid directions are: **0 = X, 1 = Y, 2 = Z** **start** controls the start position and **end** controls the end position of the minimum mesh size section (in um)
Long **GetNonuniformMinMeshSectionSize** *(Long direction)*	Returns the number of minimum mesh size sections for the specified direction
Boolean **SetNonuniformMeshCoordinate** *(Long direction, Double value)*	Adds **value** to the list of coordinates for the predefined sections of minimum mesh size for the specified direction. Valid directions are: **0 = X, 1 = Y, 2 = Z**
Boolean **GetNonuniformMeshCoordinate** *(Long direction, Long index, Variant value)*	Returns the coordinates of predefined section of minimum mesh size for the given **index** and specified direction. Valid directions are: **0 = X, 1 = Y, 2 = Z**

Method signature	Description
Long **GetNonuniformMeshCoordinateSize** *(Long direction)*	Returns the number of coordinates of predefined sections of minimum mesh size for the specified direction. Valid directions are: **0 = X, 1 = Y, 2 = Z**
Boolean **VerifyNonuniformMeshParameters** *( )*	Returns **TRUE** if the non-uniform mesh parameters are valid, **FALSE** otherwise
Long **GetDFTNumberOfSpectrumSamples** *( )*	Returns the number of spectral samples used in DFT calculations.
Double **GetDFTStartWavelength** *( )*	Returns the start wavelength used in DFT calculations
Double **GetDFTEndWavelength** *( )*	Returns the end wavelength used in DFT calculations

4.8 Observation Point Manager

The **ObservePtMgr** object is responsible for the creation of Observation **Points**, **Lines**, and **Areas**. The following functions can be accessed through the string identifier: "**ObservePtMgr**".

Method signature	Description
Object **CreateObservationPoint** *(String strID)*	Checks if the **strID** has already been used by an observation object, returns **NULL** if yes. Otherwise a new **observation point object** is created.
Object **CreateObservationLine** *(String strID)*	Checks if the **strID** has already been used by an observation object, returns **NULL** if yes. Otherwise a new **observation line object** is created.

Method signature	Description
Object **CreateObservationArea** *(String strID, Boolean bX, Boolean bY, Boolean bZ)*	Checks if the **strID** has already been used by an observation object, returns **NULL** if yes. Otherwise a new **observation area object** is created. There are three kinds of observation areas, which are controlled by **bX**, **bY** and **bZ**. **bX, bY and bZ can't be True at the same time.** The observation area will be a **YZ** area **if bX = True**; a **XZ** area if **bY = True**; a **XY** area if **bZ = True.**
Boolean **DeleteObj** (*Object obj*)	Deletes the Object **obj**.
Boolean **ValidateID** *(Sting id)*	Returns **TRUE** if the label **id** is not used already, **FALSE** otherwise.
Object **GetObjFromID** *(String id)*	Retrieves the object corresponding to **id**. Returns **NULL** if no object is found.
Boolean **DeleteAll** *( )*	Deletes all observation objects in the layout.
String **FindID** *(String basename)*	Returns a unique label using the supplied **basename**

The **Object** returned can use several methods defined in the Observation objects section

4.8.1 Observation objects

4.8.1.1 Methods common to all object types

Method signature	Description
Boolean ***SetEnabled*** *(Boolean Enabled)*	Sets the enabled flag

Method signature	Description
Boolean **IsEnabled** *( )*	Returns the enabled flag
Boolean **SetCenterExpr** *(String XExpr, String YExpr)*	Sets the center expressions
Boolean **GetCenterExpr** *(String XExpr, String YExpr)*	Given two input variables (no immediate values) this method sets them to the center expression.
Boolean **GetCenter** *(Double XCenter, Double YCenter)*	Given two input variables (no immediate values) this method sets them to the center position.
Boolean **SetCenter** *(Double XCenter, Double YCenter)*	This method sets the center using the values passed.
Boolean **Is2DTEHxCollected** *( )*	Returns **True** if the **Hx** field component is collected for 2D TE simulation
Boolean **Is2DTEHzCollected** *( )*	Returns **True** if the **Hz** field component is collected for 2D TE simulation
Boolean **Is2DTMExCollected** *( )*	Returns **True** if the **Ex** field component is collected for 2D TM simulation
Boolean **Is2DTMEzCollected** *( )*	Returns **True** if the **Ez** field component is collected for 2D TM simulation
Boolean **Collect2DTE** *(Boolean bX, Boolean bZ)*	Sets the **2D TE** simulation fields collection on or off. The parameter **bX** is for the "**Hx**" component, the parameter **bZ** is for the "**Hz**" component. The "**Ey**" component is always collected in TE simulations.

Method signature	Description
Boolean **Collect2DTM** *(Boolean bX, Boolean bZ)*	Sets the **2D TM** simulation fields collection on or off. The parameter **bX** is for the "**Ex**" component, the parameter **bZ** is for the "**Ez**" component. The "**Hy**" component is always collected in TM simulations.
Boolean **GetCenterOffset** *(Variant dx, Variant dy)*	Given two input variables (no immediate values) this method sets them to the center offset values.
Boolean **SetCenterOffset** *(Double dx, Double dy)*	This method sets the center offset to the values passed.

4.8.1.2 Observation point

Method signature	Description
Boolean **Collect3D** *(Boolean bEx, Boolean bEy, Boolean bEz, Boolean bHx, Boolean bHy, Boolean bHz)*	Sets the **3D** simulation fields collection on or off. The parameters correspond to the **Ex**, **Ey, Ez, Hx, Hy, Hz** fields.
Boolean **SetDepthExpr** *(String strDepthExpr)*	Sets the center depth expression (Y-axis) of the observation point
String **GetDepthExpr** *( )*	Returns the center depth expression (Y-axis) of the observation point
Double **GetDepth** *( )*	Returns the center depth (Y-axis) of the observation point
Boolean **Is3DExCollected** *( )*	Returns **True** if the **Ex** field component is collected for 3D simulation

Method signature	Description
Boolean **Is3DEyCollected** *( )*	Returns **True** if the **Ey** field component is collected for 3D simulation
Boolean **Is3DEzCollected** *( )*	Returns **True** if the **Ez** field component is collected for 3D simulation
Boolean **Is3DHxCollected** *( )*	Returns **True** if the **Hx** field component is collected for 3D simulation
Boolean **Is3DHyCollected** *( )*	Returns **True** if the **Hy** field component is collected for 3D simulation
Boolean **Is3DHzCollected** *( )*	Returns **True** if the **Hz** field component is collected for 3D simulation
Object **GetTimeSeries** *(String FieldName)*	This method will return an observation points time series for the requested field. **FieldName** - One of "**Ex**", "**Ey**", "**Ez**", "**Hx**", "**Hy**", "**Hz**".

4.8.1.3 Observation line

Method signature	Description
Boolean **SetAngleExpr** (*String Expr*)	Sets the Angle expression.
Boolean **SetLengthExpr** (*String Expr*)	Sets the length expression
Boolean **SetLengthOffset** (*double LengthOffset)*	Sets the Length offset (in um)
Boolean **SetAngle** (*double Angle*)	Sets the Angle (in degrees)

Method signature	Description
Double ***PowerAtCenterWavelength*** *( )*	Calculates power at the Center Wavelength. The Center Wavelength value is specified by the primary (key) Input Plane
Double **PowerAtWavelength** *(Double wavelength)*	Calculates power at the specified Wavelength.
Boolean **CalcPowerSpectrum** *(Double startwl, Double endwl, Long npoints)*	Calculates power spectrum between **startwl** and **endwl** for the specified number of wavelength sampling points **npoints**.
Double **GetPowerAtSpectralPoint** *(long index)*	Retrieves the power at the specified index. **CalcPowerSpectrum (**) must be called prior to retrieving the values.
Boolean **CalcNormalizedPowerSpectrum** *(Double startwl, Double endwl, Long npoints, String InputPlaneName)*	Calculates power spectrum normalized to the power of the specified Input Plane, for a specified number of wavelength sampling points
Double **GetNormalizedPowerAtSpectralPoint** *(Long index)*	Retrieves the normalized power at the specified index. **CalcNormalizedPowerSpectrum (**) must be called prior to retrieving the values.
Double **GetNormalizedWavelengthAtSpectralPoint** *(Long index)*	Retrieves the wavelength corresponding to a specified index. **CalcNormalizedPowerSpectrum (**) must be called prior to retrieving the values.

4.8.1.4 Observation area

Method signature	Description
Boolean **SetWidthExpr** *(String expr)*	Sets the width expression.

Method signature	Description
Boolean **SetHeightExpr** *(String expr)*	Sets the height expression.
Boolean **SetWidthOffset** *(Double value)*	Sets the width offset value (in um)
Boolean **SetHeightOffset** *(Double value)*	Sets the height offset value (in um)
Boolean **SetDepthExpr** *(String expr)*	Sets the depth expression.
Boolean **SetDepthOffset** *(Double value)*	Sets the depth offset value (in um)
Boolean **GetDepth** *(Double value)*	Returns the depth value (in um)
Boolean **Is3DExCollected** *( )*	Returns **True** if the **Ex** field component is collected for 3D simulation
Boolean **Is3DEyCollected** *( )*	Returns **True** if the **Ey** field component is collected for 3D simulation
Boolean **Is3DEzCollected** *( )*	Returns **True** if the **Ez** field component is collected for 3D simulation
Boolean **Is3DHxCollected** *( )*	Returns **True** if the **Hx** field component is collected for 3D simulation
Boolean **Is3DHyCollected** *( )*	Returns **True** if the **Hy** field component is collected for 3D simulation
Boolean **Is3DHzCollected** *( )*	Returns **True** if the **Hz** field component is collected for 3D simulation
Boolean **Collect3D** *(Boolean bEx, Boolean bEy, Boolean bEz, Boolean bHx, Boolean bHy, Boolean bHz)*	Sets the **3D** simulation fields collection on or off. The parameters correspond to the **Ex**, **Ey, Ez, Hx, Hy, Hz** fields.

Method signature	Description
Boolean **SelectComponentForMovie** *(String ComponentName, String Domain)*	This method selects a component of the observation area for recording of a movie. **ComponentName** - Name of the component, one of "**Ex**", "**Ey**", "**Ez**", "**Hx**", "**Hy**", "**Hz**". **Domain** - Name of the domain, one of "**3D**", "**TE**", "**TM**".
Boolean **UnselectComponentForMovie** *(String ComponentName, String Domain)*	This method deselects a component of the observation area for recording of a movie. This method selects a component of the observation area for recording of a movie. **ComponentName** - Name of the component, one of "**Ex**", "**Ey**", "**Ez**", "**Hx**", "**Hy**", "**Hz**". **Domain** - Name of the domain, one of "**3D**", "**TE**", "**TM**".
Boolean **IsComponentSelectedForMovie** *(String ComponentName, String Domain)*	This method checks if a component of the observation area is selected for recording of a movie. **ComponentName** - Name of the component, one of "**Ex**", "**Ey**", "**Ez**", "**Hx**", "**Hy**", "**Hz**". **Domain** - Name of the domain, one of "**3D**", "**TE**", "**TM**".
Boolean **ExportGUISelectedFieldData** *(String RootFileName, double Lambda)*	This method will export the current iteration optical fields (those that are selected in the GUI) for the observation area. **RootFileName** - Unique filename prefix. "**<field_name>.f3d**" will be appended. Where <field_name> is one of "**Ex**", "**Ey**", "**Ez**", "**Hx**", "**Hy**", "**Hz**".

Method signature	Description
	Lambda - Selects the wavelength in the simulation spectrum. (in um) . The wavelength must exist in the DFT list.
Object **GetEIntensity** *(double Lambda)*	This method will compute the intensity of the selected "E" fields in the observation area object at the given wavelength. The wavelength must exist in the DFT list.
Object **GetHIntensity** *(double Lambda)*	This method will compute the intensity of the selected "H" fields in the observation area object at the given wavelength. The wavelength must exist in the DFT list.
Object **GetField** *(String FieldName, double Lambda)*	This method will return the requested field for the observation area at the given wavelength. **FieldName** - One of "**Ex**", "**Ey**", "**Ez**", "**Hx**", "**Hy**", "**Hz**" or "**ExEy**". **Lambda** - Selects the wavelength in the simulation spectrum.
Integer GetPowerTypeCount ***( )***	Returns the number of power data types (P_u, P_v, P_total) collected by the area.
String **GetPowerTypeID** *(Integer Ratio_Type_ID)*	Returns a string name assigned to the power ratio for a given direction, based on an identification number (ratio type) : • 0 - Returns name of the "**P_u**" type. • 1 - Returns name of the "**P_v**" type. • 2 - Returns name of the "**P_total**" type.
Double **CalculatePowerRatioCW** *(String ratio_name)*	Calculates the power ratio at the center wavelength, based on the given ratio name (type) .

Method signature	Description
Double **CalculatePowerRatioCW_u** *( )*	Calculates the power ratio **P_u** at the center wavelength for the Observation Area.
Double **CalculatePowerRatioCW_v** *( )*	Calculates the power ratio **P_v** at the center wavelength for the Observation Area.
Double **CalculatePowerRatioCWTotal** *( )*	Calculates the power ratio **P_total** at the center wavelength for the Observation Area.
Object **CalculatePowerSpectrum** *(String ratio_name)*	Calculates the power spectrum normalized to the **Key Input** for the Observation Area, based on the given ratio name (type) .The function returns an object of **VectorXY** type.
Object **CalculatePowerSpectrum_u** *( )*	Calculates the power spectrum **P_u** normalized to the **Key Input** for the Observation Area. The function returns an object of **VectorXY** type.
Object **CalculatePowerSpectrum_v** *( )*	Calculates the power spectrum **P_v** normalized to the **Key Input** for the Observation Area. The function returns an object of **VectorXY** type.
Object **CalculatePowerSpectrumTotal** *( )*	Calculates the power spectrum **P_total** normalized to the **Key Input** for the Observation Area. The function returns an object of **VectorXY** type.

VectorXY objects

The VectorXY objects are containers carrying 2D arrays (X, Y pairs) of data results. Data elements can be accessed by the functions below.

Method signature	Description
WriteF2D *(String File_Name)*	Write the data to an ".**f2d**" file.
Integer **GetSize** *( )*	Get the size of the data array.
Double **GetXAt** *(Integer index)*	Get the X value at the given index. For power spectrum results, this contains the wavelength in um.
Double **GetYAt** *(Integer index)*	Get the Y value at the given index. For power spectrum results, this contains the normalized power value.

3D Field objects

3D Field objects are obtained by using **GetEIntensity ( ), GetHIntensity ( ), or GetField ( )** methods**.**

Method signature	Description
Boolean **WriteF3D** *(String FileName)*	This method will write the contents of the object into an ".f3d" file. **FileName** - Sets the name of the file to write to.

4.9 Mask File (GDS II and DXF) Import Manager

The **MaskFileImporter** object can be used to import objects into the current layout that are stored in either the GDS II and DXF formats. The listed below can be accessed through the string identifier: "**MaskFileImporter**".

Method signature	Description
Object **CreateObjForImportSettings***()*	Returns the object that is to be used for setting the import parameters.

Method signature	Description
Object ***Import*** *(Object importSettings)*	Perform the import using the settings defined within the object **importSettings** and returns an object that holds result of the import. **Note:** If an imported object is outside the bounds of the wafer, the width or length of the wafer will be updated accordingly. New channel profiles might be created and used by the imported objects if the layers are not assigned to existing waveguide profiles. Whether the profiles will be created will be based on flags set in import settings. The object returned can be used to retrieve all the imported objects, layers, etc. Refer to the description of **import result** object.

Import Settings .

Import Results .

4.9.1 Object for Import settings

The following methods are only available to objects that are retrieved by calling the function **MaskFileImporter.CreateObjForImportSettings**

Method signature	Description
Boolean **SetFile** *(String filewithfullpath)*	Set the file along with its full path that is to be processed by the importer.
IgnoreALayer (String strLayerName)	Ignore a specific layer that is identified by the name given as parameter, strLayerName. The objects contained in this layer will not imported into the current layout.

Method signature	Description
IgnoreLayers (Array strLayerNames)	Ignore a group of layers that are identified by an array that contains all of the layer names.
SetDXFAccuracy (Double dVal)	Importing a DXF file requires setting the accuracy that is to be used in processing the DXF file. If the value is not set or there is an error in the value used, the default value will be used. **Default value - 0.01.**
Boolean **AssignAProfileToLayer** (String ProfileName, String LayerName)	This method assign a profile, a valid profile as defined in OptiFDTD or OptiBPM, to a layer within the file to import. This will result in all imported objects defined in the layer being assigned to corresponding profile. The ProfileName has to name a valid profile in the current layout. The LayerName is the layer within the file to import. The value can be set as empty making the specified profile the default profile for all the layers that don't have an associated profile.
Boolean **AssignProfilesToLayer**s (Array ProfileNames, Array LayerNames)	This method assigns a series of profiles, must be valid profiles as defined in OptiFDTD or OptiBPM, to a series of layers within the file to import. The profiles and layers are defined in two arrays: ProfileNames: An array containing strings that represent profile IDs in the current OptiFDTD or OptiBPM layout. LayerNames: An array containing strings that represent the layer names within the imported design. An empty string will set the paired profile as the default profile for all layers that don't have an associated profile. **Example:** **...** *Dim profiles*

Method signature	Description
	Dim layers *profiles = Array("Channel", "FiberA", "FiberB")* *layers = Array("0", "1", "")* *' Profile "FiberB" will be the default profile for all imported objects except objects in* *' layer "0" and "1".* *im_settings.AssignProfilesToLayers profiles, layers* *...* **Note:** If a a layer does not have an associated profile and the default profile is not set then the *IgnoreErrors* flag (see below) is used to determine the appropriate behaviour. If the flag is set to FALSE, the file import will stop. If the flag is TRUE, a new channel profile with "Air" material will be created.
IgnoreErrors(Boolean bFlag)	A boolean flag (TRUE or FALSE) used to set whether the importer should ignore minor errors in the parameter settings to continue file import. If the flag is set to true then the importer will use default values for required quantities or create channel profiles for use as required. **Default value - FALSE, telling the method to stop the import when encountering any setting errors.**
Boolean GetIgnoreErrorFlag()	Returns the boolean value of the *IgnoreErrors*flag that indicates if minor errors in parameter settings can be ignored.

4.9.2 Object for managing the import results

The following methods are only available to objects that are retrieved by calling **MaskFileImporter.Import**

Method signature	Description
Boolean **IsSuccessful**	*Returns whether the import call was successful (TRUE) or not (FALSE*) and that objects have been imported into the current layout.
String **GetErrorMessage**	Returns the last error from **MaskFileImporter**. If the the import was successful the return value is empty, "".
VARIANT **GetAllObjID**	Returns an array that contains the string IDs for all imported layout objects.
VARIANT **GetImportedLayerID**	Returns an array that contains the layer names in the original design that contain imported objects.
VARIANT **GetAllObjIDInLayer** (String LayerName)	Returns an array that contains all the string IDs of imported layout objects that are in the layer "LayerName" from the original design.