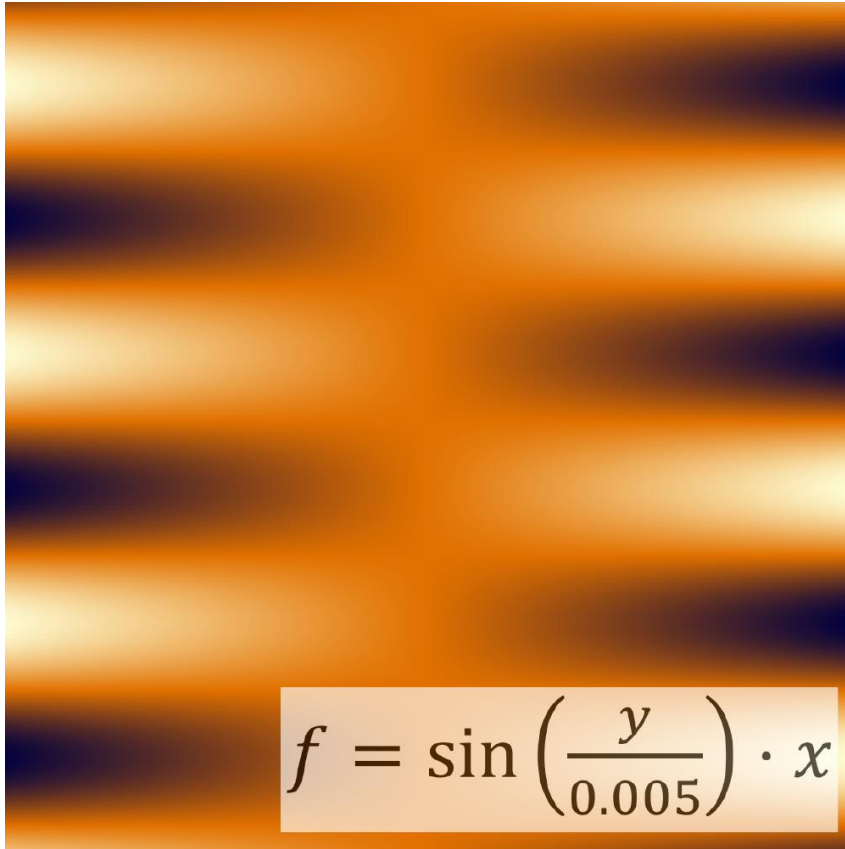


Programmable Light Source, Function, Interface and Medium

Abstract



VirtualLab offers always different ways for the specification of optical objects, e.g., light source, interface, and medium. We provide predefined objects, the objects can be specified by measurement data or the object can be described by using programmable objects. The programming language for these customized objects is C#. VirtualLab supports the development of the programmable items by the source code editor. Different object have different input and return parameters. This use case explains the most specific parameters of the programmable light source, interface, function and medium.

Programmable Light Source

- *Programmable Light Source* allows users to define the spatial distribution of a global polarized light mode in one plane. The mathematical representation is $\check{E}_0(x, y, z, \check{J}, \lambda, \check{n}(\lambda))$

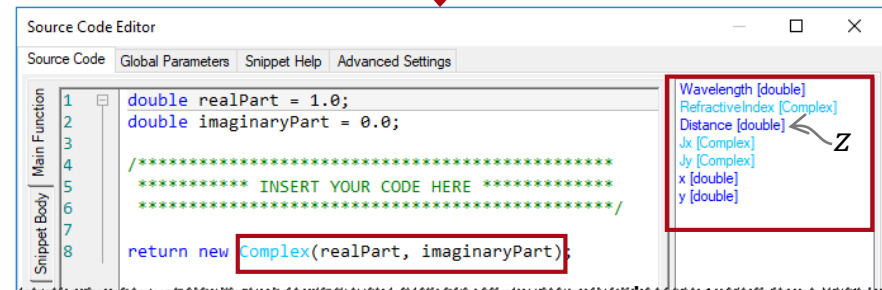
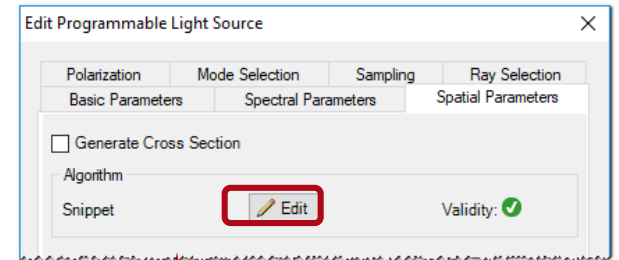
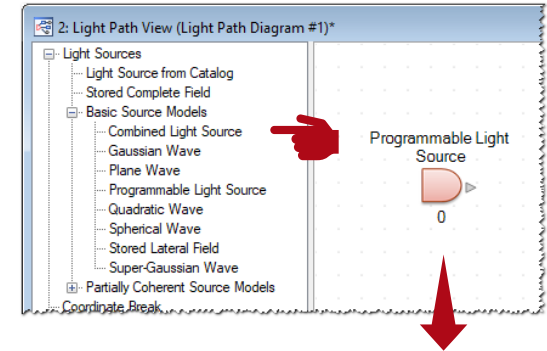
- (x, y, z) is the spatial coordinators

- $\check{J} = \begin{pmatrix} \check{J}_x \\ \check{J}_y \end{pmatrix}$ represents the Jones vector

$$\mathbf{E} = \begin{pmatrix} \check{E}_x \\ \check{E}_y \end{pmatrix} = \check{E}_0 \cdot \begin{pmatrix} \check{J}_x \\ \check{J}_y \end{pmatrix}$$

- λ is wavelength

- $\check{n}(\lambda)$ is the refractive index of the surrounding medium



\check{E}_0

input parameters:
 $x, y, z, \mathbf{J}, \lambda, \check{n}(\lambda)$

Example of Programmable Light Source

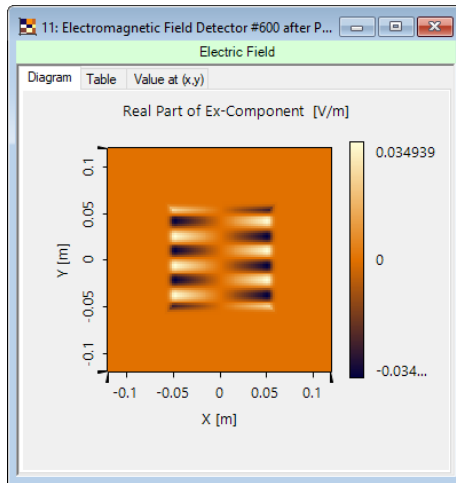
C# snippet

```
double realPart = 1.0;  
double imaginaryPart = 0.0;  
  
realPart = Math.Sin(y / 5e-3) * x;  
  
return new Complex(realPart, imaginaryPart);
```

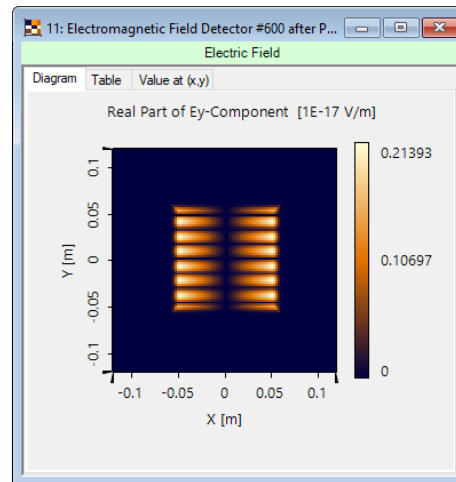
Source parameters

parameter	value
size	100 mm x 100 mm
polarization	right circularly polarized

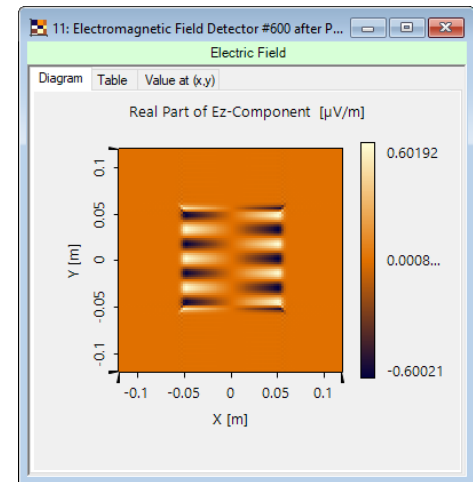
electric field [V/m]



$\Re(E_x)$



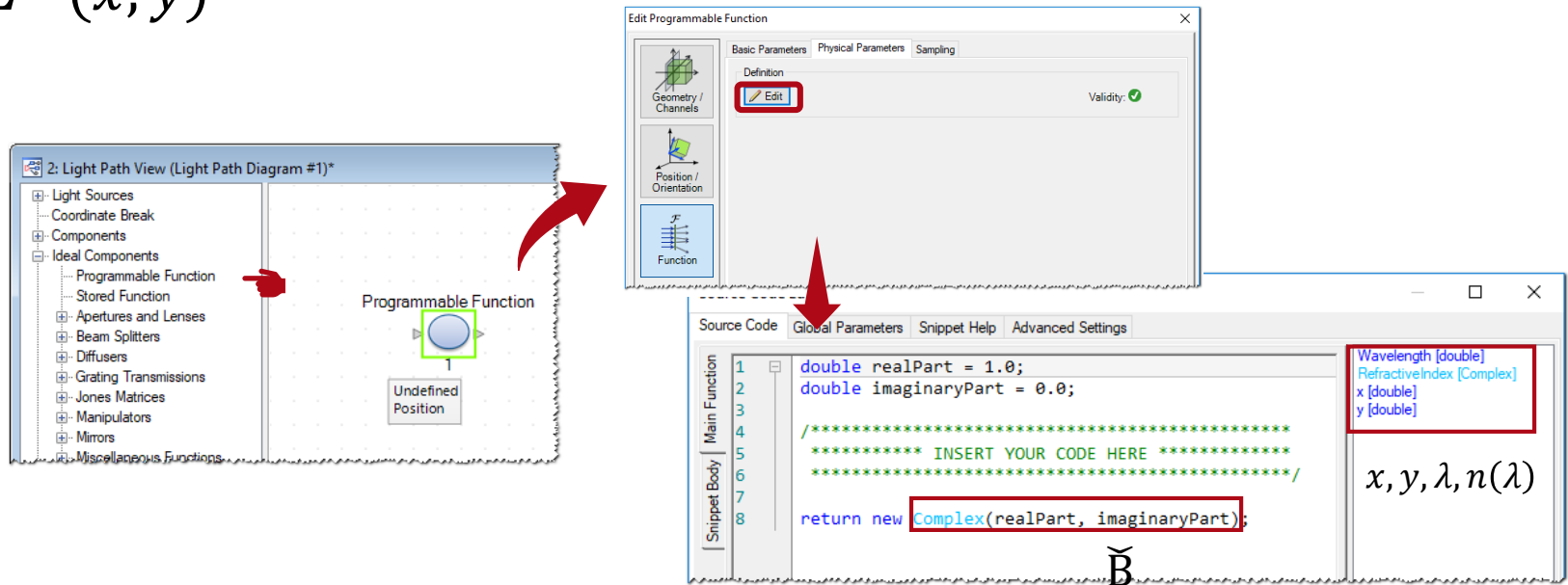
$\Re(E_y)$



$\Re(E_z)$

Programmable Function

- Programmable function allows users to define a specific boundary response, which is mathematically represented as $\vec{B} = f(x, y, \lambda, \tilde{n}(\lambda))$.
- Output field of *Programmable Function* is: $E^{\text{out}}(x, y) = \vec{B} \cdot E^{\text{in}}(x, y)$

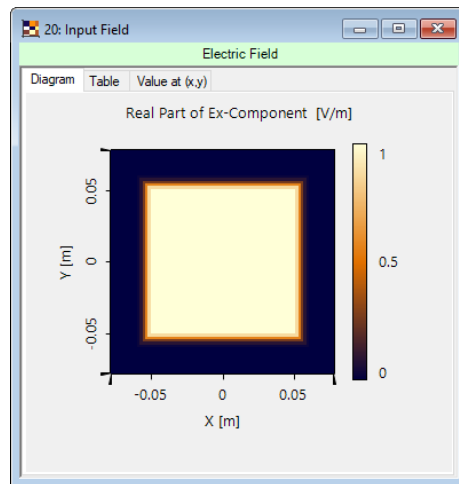


Example of Programmable Function

C# snippet

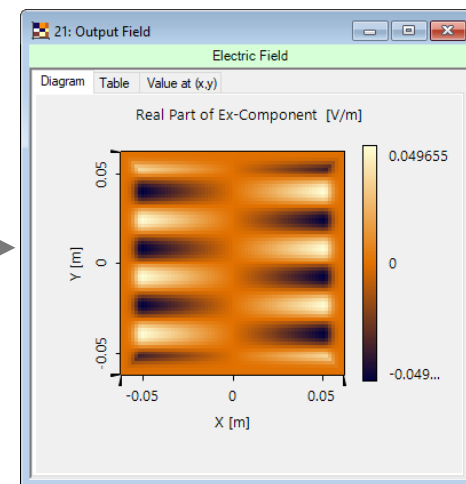
```
double realPart = 1.0;  
double imaginaryPart = 0.0;  
  
realPart = Math.Sin(y / 5e-3) * x;  
  
return new Complex(realPart, imaginaryPart);
```

input field



$\Re(E_x^{\text{in}})$

output field



$\Re(E_x^{\text{out}})$

programmable
function

Programmable Interface

- *Programmable Interface* allows users to define a height profile of an interface.
 - Height profile $h(x, y)$, e.g., $h(x, y) = \sin\left(\frac{y}{0.005}\right) \cdot x$

The image shows a software interface with several windows. The 'Interfaced Catalog' window is open, showing a list of interface types. The 'Programmable Interface' is selected. The 'Edit Programmable Interface' dialog is open, showing the 'Interface Specification' tab. The 'Snippet for Height Profile' is set to 'Numerical Gradient Calculation'. The 'Accuracy Factor' is set to 1. The 'Source Code Editor' window is open, showing the source code for the height profile:

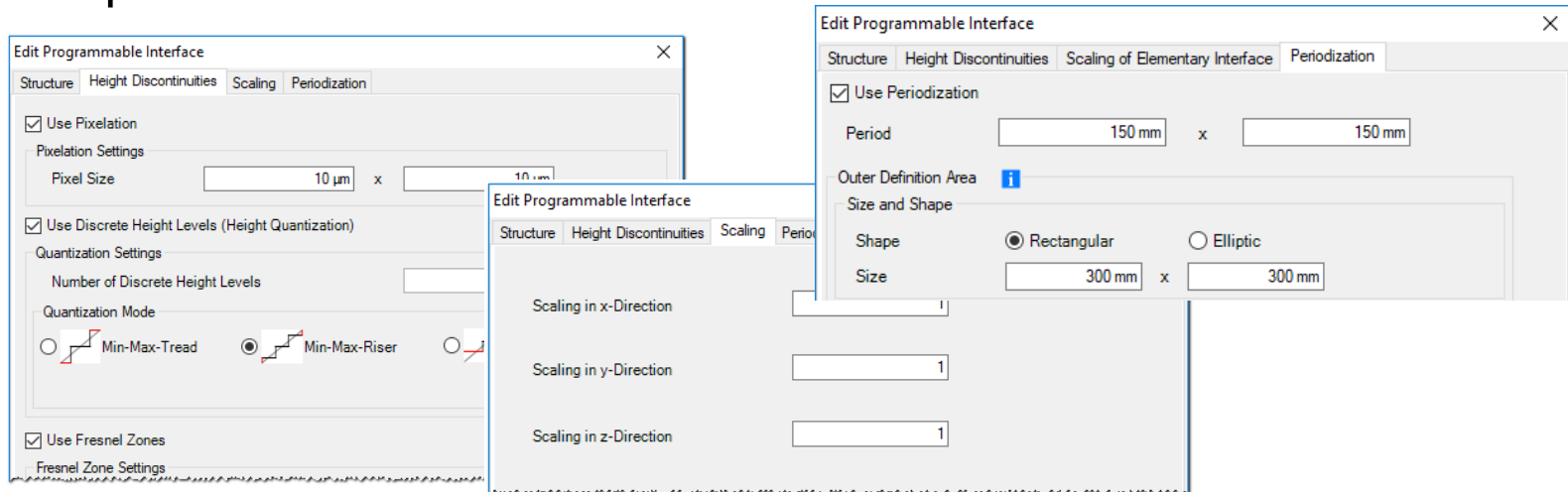
```
double height = Math.Sin(y / 5e-3) * x;
return height;
```

 The variable 'height' is highlighted in red. The '3D view of the interface' window shows a 3D model of the interface, which is a wavy surface. The 'Z-Extension' is set to 100 mm, Minimum to -50 mm, and Maximum to 50 mm. The 'ApertureDiameterX [double]', 'ApertureDiameterY [double]', 'x [double]', and 'y [double]' are also highlighted in red in the 'Source Code Editor' window.

3D view of the interface

Programmable Interface: Properties

- *Programmable Interface* contains all properties of general interfaces in VirtualLab Fusion.
 - discontinuities: the interface can be quantized
 - scaling: stretch the interface in specific direction
 - periodization: make a repetition of the interface to generate a periodic interface.



More details can be found in “Getting started” tutorial, i.e., Catalogs II: Interfaces

Programmable Medium

- *Programmable Medium* allows users to define an arbitrary, complex-valued refractive index distribution $\check{n}(x, y, z, \lambda)$. Variance of refractive index $\Delta\check{n}(x, y, z, \lambda)$ is programmed.

The screenshot illustrates the software interface for defining a programmable medium. It shows the 'Media Catalog' window with a list of medium types, including 'Programmable Medium (x-y-z-Modulated)'. The 'Edit Programmable Medium (x-y-z-Modulated)' dialog is open, showing the 'Basic Parameters' tab. The 'Base Material' is set to 'Fused_Silica'. The 'Index Modulation' section is active, and the 'Definition' field is highlighted. The 'Source Code Editor' shows the following code snippet:

```
1 double realPart = 0.0;  
2 double imaginaryPart = 0.0;  
3  
4 realPart = Math.Sin(y / 0.005) * x;  
5  
6 return new complex(realPart, imaginaryPart);
```

The refractive index distribution is defined as $\check{n}(x, y, z, \lambda)$, and the variance is $\Delta\check{n}(x, y, z, \lambda)$.

Programmable Medium

- *Programmable Medium* allows users to define an arbitrary, complex-valued refractive index distribution $\check{n}(x, y, z, \lambda)$. $\Delta\check{n}(x, y, z, \lambda)$ is programmed.
- There are two ways to define $\check{n}(x, y, z, \lambda)$:

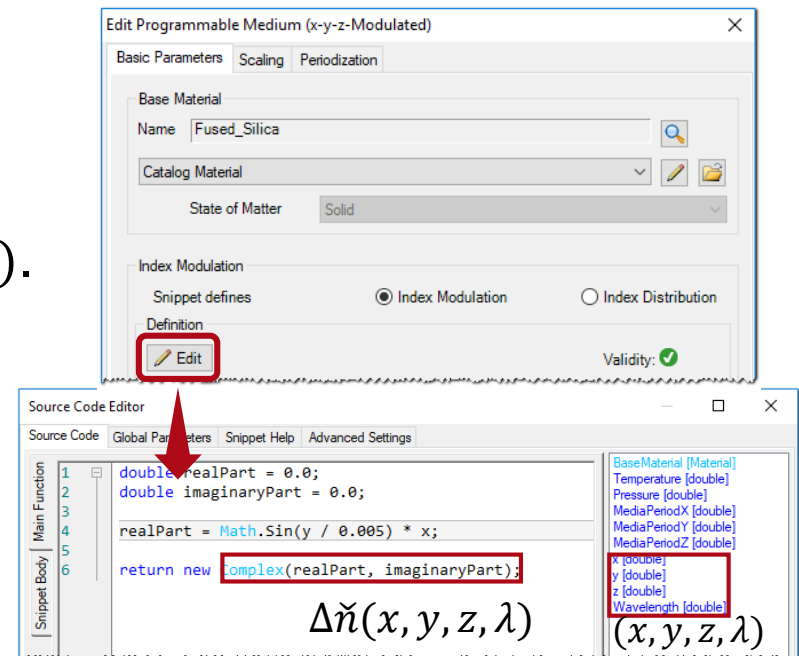
- *Index Distribution*:

$$\check{n}(x, y, z, \lambda) = \Delta\check{n}(x, y, z, \lambda)$$

- *Index Modulation*:

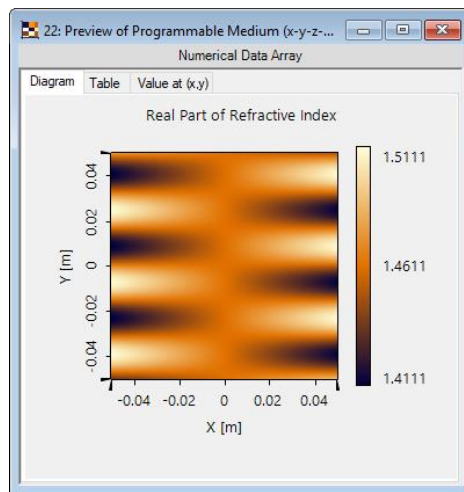
$$\check{n}(x, y, z, \lambda) = n_0(\lambda) + \Delta\check{n}(x, y, z, \lambda).$$

$n_0(\lambda)$ is given by *Base Material*.



Example of Programmable Medium

- *Programmable Medium* allows users to define an arbitrary, complex-valued refractive index distribution $\check{n}(x, y, z, \lambda)$.
 - E.g., $\check{n}(x, y, z, \lambda) = \check{n}^{\text{FS}}(\lambda) + \sin\left(\frac{y}{0.005}\right) \cdot x$, with $\check{n}^{\text{FS}}(\lambda)$ denoting the refractive index of fused silica.



refractive index

```
1 double realPart = 0.0;  
2 double imaginaryPart = 0.0;  
3  
4 realPart = Math.Sin(y / 0.005) * x;  
5  
6 return new complex(realPart, imaginaryPart);
```

$\Delta\check{n}(x, y, z, \lambda)$

(x, y, z, λ)

Document Information

title	Programmable Light Source, Function, Interface and Medium
version	1.0
VL version used for simulations	7.0.3.4
category	Feature Use Case
