Tutorial 507 (1.0)

Programmable Building Blocks, Components and Detectors of VirtualLab™

Authors: Christian Hellmann, Michael Kuhn (LightTrans)

Related Tutorials: Tutorial 501

Requirements: VirtualLab™ 5.5 – Starter Toolbox

LIGHTTRANS

Contents

- Introduction
  - Goal of the programming
  - Programming language
  - Some references
- Programmable Objects in VirtualLab
  - Modules
  - Snippets
- Snippets for Programmable Components
  - The concept
  - Input and output of the Programmable Component
  - Examples

Contents

- Programming Environments
  - Using the VirtualLab editor
  - Using Visual Studio Projects
  - How to setup a project
  - How to include C++ DLLs
  - MATLAB® and VirtualLab
  - Example of DLLs within Snippets
- Programmable Detector
  - Concept
  - Overview of output types

Introduction

Goal of Programming in VirtualLab

- VirtualLab is the first Field Tracing software on the market.
- The requirements on modern optical simulation are getting more and more complex.
- LightTrans offers you a lot of simulation techniques, but you are enabled to run your own algorithms in VirtualLab as well.
- The programming interface within VirtualLab allows the customization of building blocks as well of components and detectors.
- So this allows full flexibility in system simulation.

Programming Language

- This overview is not a programming course. It is a starting point for “learning by doing”. Own exercises are recommended.
- Used Programming language: C#
  - Object oriented language similar to C++.
- .NET Framework
  - Release 2002
  - Large library of functions, classes, especially GUI elements that can be used.

Programming Language

- C# - some references
  - Beginning C# 3.0: An Introduction to Object Oriented Programming (Jack Purdum, Wiley Publishing Inc., 2008)
  - C# 3.0: A Beginner's Guide (Herbert Schildt, McGraw Hill, 2009)
  - Professional C# 2008 (Christian Nagel, Bill Evjen, Jay Glynn, Morgan Skinner, Karli Watson, Wiley Publishing Inc., 2008)
  - C# 3.0 The Complete Reference (Herbert Schildt, McGraw Hill, 2009)

Programming Language

- .NET Functions
  - .NET data type (double, int, etc)
  - e.g. Math.Sin(), GUI, etc.
  - http://msdn.microsoft.com/en-us/library/67ef8sbd(VS.80).aspx
- VirtualLab™ Programming
  - Access to classes and functions of the VirtualLab library (VirtualLabAPI).
  - Programming Reference available in the Help menu of VirtualLab.

Some C# Syntax

- Comments:
```
/*  comment (multiple lines) */
// comment (1 line)
```
- Variable types
```
int i; double x; float y;
```
- Math-function:
```
y=Math.Sin(x); y=Math.Abs(x);
```
- use “`;`” at end of a line
- to return a value use
```
return x;
```

Levels for Programming in VirtualLab

Programming within VirtualLabAPI.dll

- Programming of the VirtualLabAPI is being done by LightTrans and is restricted to LightTrans.
- Currently about 3000 files and 800.000 lines of code.
- Highest flexibility, but design rules are to be applied.
- Code is NOT available to externals, but classes and methods can be used in snippets, modules.
- Many third party DLLs are used from VirtualLabAPI, development licenses for using these DLLs are restricted to LightTrans.

VirtualLab Modules

- VirtualLab Modules can be written in C# or in VB .NET.
- 100-1000 lines of code are typically within a module.
- Classes and functions of .NET and VirtualLabAPI.dll and other external references can be used.
- This allows the user a very flexible way to solve a custom task.

VirtualLab Snippets

- VirtualLab introduced the option to define different parts of the simulation by the specification of *snippets*:
  - Source code block which represents just the body of some function.
  - Typically 10-100 lines of code are written.
  - Within a snippet no classes or functions can be specified.
  - The usage of function will be added in 2013.

VirtualLab Snippets

- It is also allowed to add external references to snippets. This allows to call functions from external DLLs. Those DLLs can be
  - .NET DLLs
  - C++ DLLs
  - DLLs originating from MATLAB® code. For this we refer to the Tutorial 501 “Using MATLAB® Functions from VirtualLab Snippets and Modules“

Using external DLLs and user rights

- Using external DLLs requires the following Windows user rights:
  - During the execution a folder “DLLCache” is created and manipulated. The folder is placed into the installation directory of VirtualLab.
  - If VirtualLab has been installed at the system drive (typically C:), then Administrative rights are required. That is VirtualLab has to be started with “Run as Administrator”.
  - Alternatively, VirtualLab can be installed at another drive (not C:) where ordinary user rights are sufficient to create and manipulate folders.

Restrictions

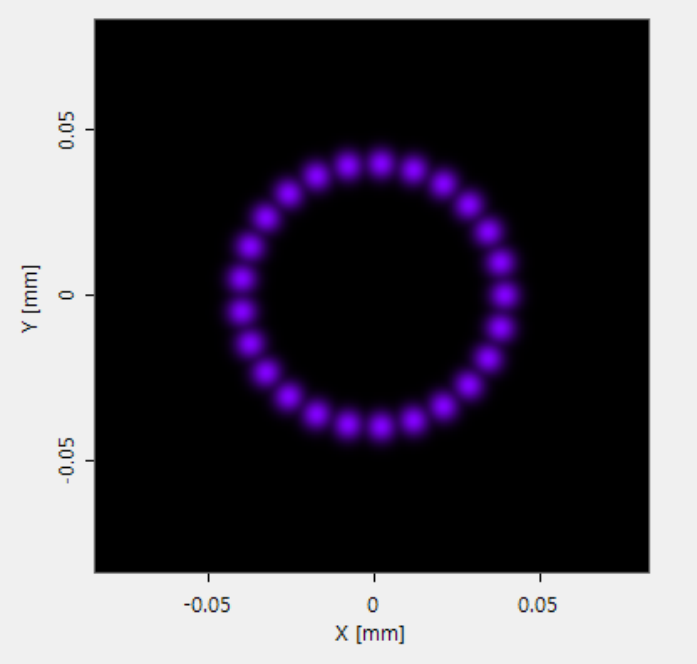
- VirtualLab Advanced
  - No restriction with respect to the programming features.
- VirtualLab (Standard)
  - Programmable Components and Programmable Detectors are read only, i.e. the source code cannot be modified.
- VirtualLab Trial
  - Cannot run modules.
  - All programmable building blocks based on snippets are read only.

Snippet-based Elements in VirtualLab

Programmable (Mode Planar) Light Source

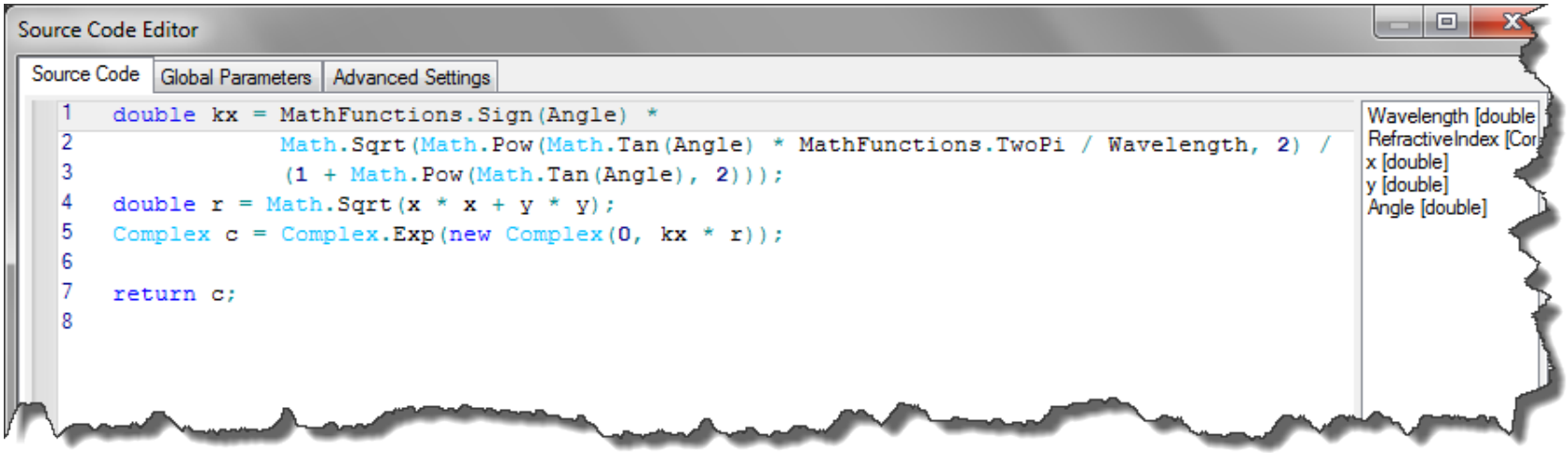
- Light source allows programming of:
  - Lateral distribution
  - Spectral distribution (by spectrum generator)
  - Mode locations
  - Weights for modes

Source Catalog:
Modes on a Circle
(λ = 405 nm).



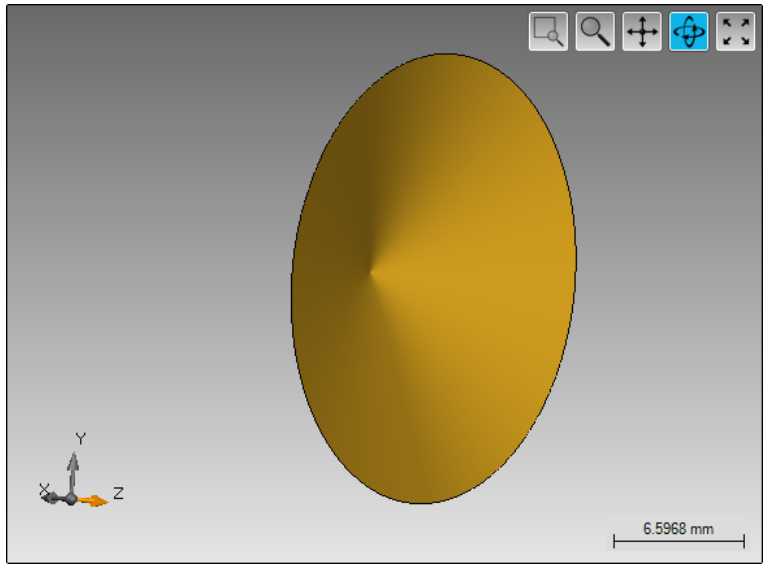
Programmable Transmission

- Idealized effects that can be described by a transmission function can be formulated via snippets.
- The user specifies the transmission as a function of the position (x,y).



Programmable Interface

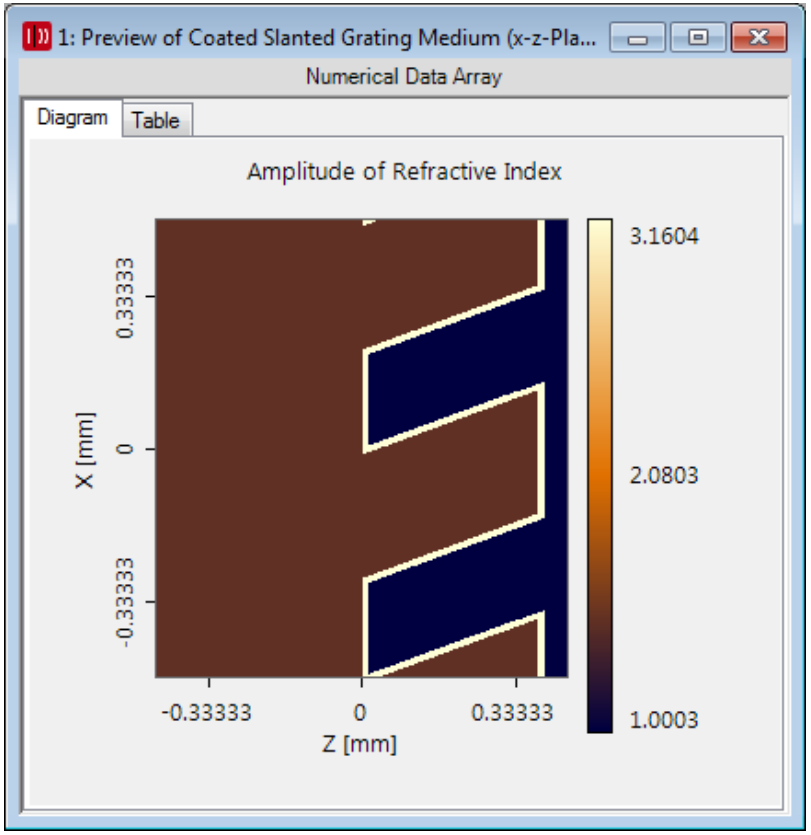
- The user can define the height function of an optical interface $h(x, y)$.
- Additionally, the partial derivatives $\frac{\partial h}{\partial x}(x, y)$ and $\frac{\partial h}{\partial y}(x, y)$ can be specified.



Interface Catalog:
Axicon
Interface 20deg.

Programmable Medium

- Within a programmable medium the user can define a 3D index modulation.



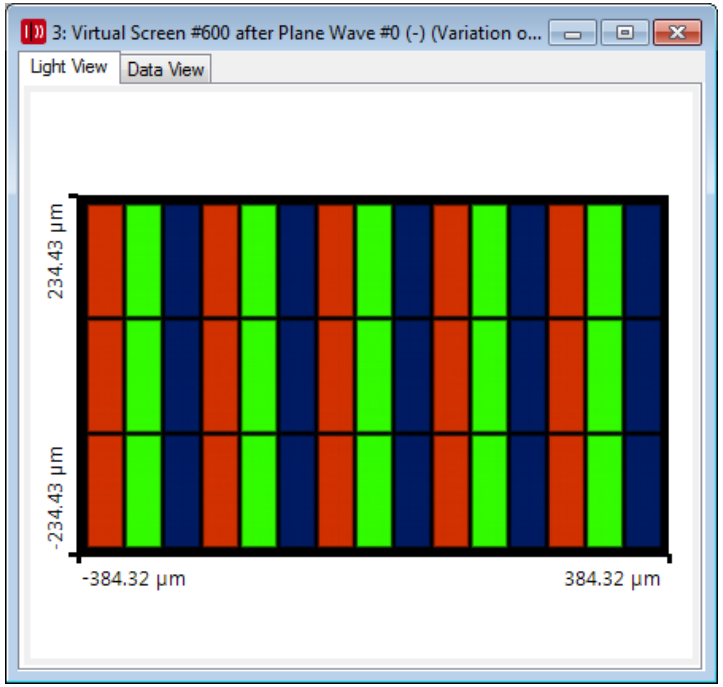
Media Catalog:
Coated Slanted Grating Medium.

Parameter Run: Programmable Mode

- The Parameter Run within VirtualLab allows the variation of parameters in a user defined mode (alternatively scanning, random and standard mode can be used).
- The user defined selection of parameters is important if the parameter to vary shall be dependent from each other or if the user likes to evaluate the output of the Parameter Run in a specific way.

ParameterRun: Programmable Mode

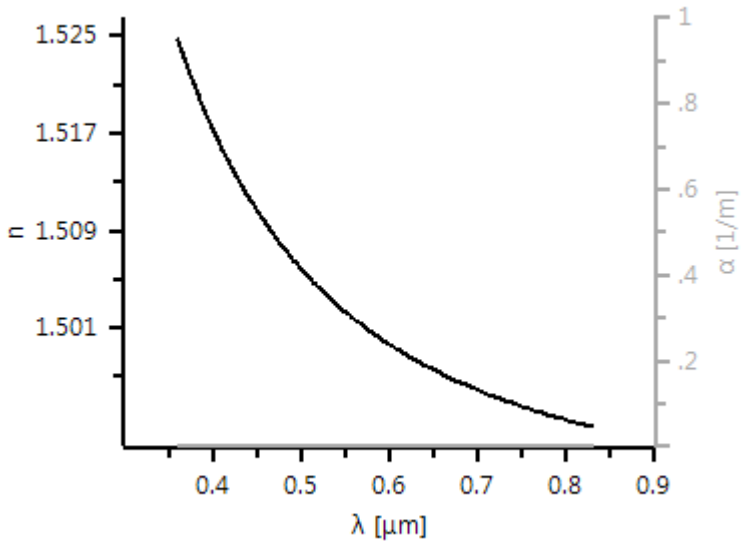
- Example:
  - Scenario 178.01: LCD Source Simulation
  - The parameters Wavelength and Weights are dependent from each other. The positions are varied additionally.



Result of Scenario 178.01:
An array of RGB pixels which can be understood as pixel array within an LCD. The field can be used for further simulations.

Programmable Material

- The user can define the dispersion formula in a snippet.



Materials Catalog: Abbe Number V_d Material.

Programmable Component

- The programmable component allows the definition of a complete component and its propagation function.
- The developer of the snippet can specify the output of the component dependent on the incident field as well as on structural parameters.
- The programmable component has no extension, so its effect is only described in one plane.
- Further information are given below, in the Manual of VirtualLab and the Programming Reference (Help menu in VirtualLab).

Programmable Detector

- The programmable detector allows the generation of *DetectorResult* objects dependent on the input light field object.
- The user can extract information from the incident field and out put them via:
  - Physical Value (numbers + physical property)
  - Harmonic fields (*ComplexAmplitude* objects)
  - Data Arrays
- Output of physical values can further be used for parametric optimization.
- Further information are given below, in the Manual of VirtualLab and the Programming Reference (Help menu in VirtualLab).

Programmable Components

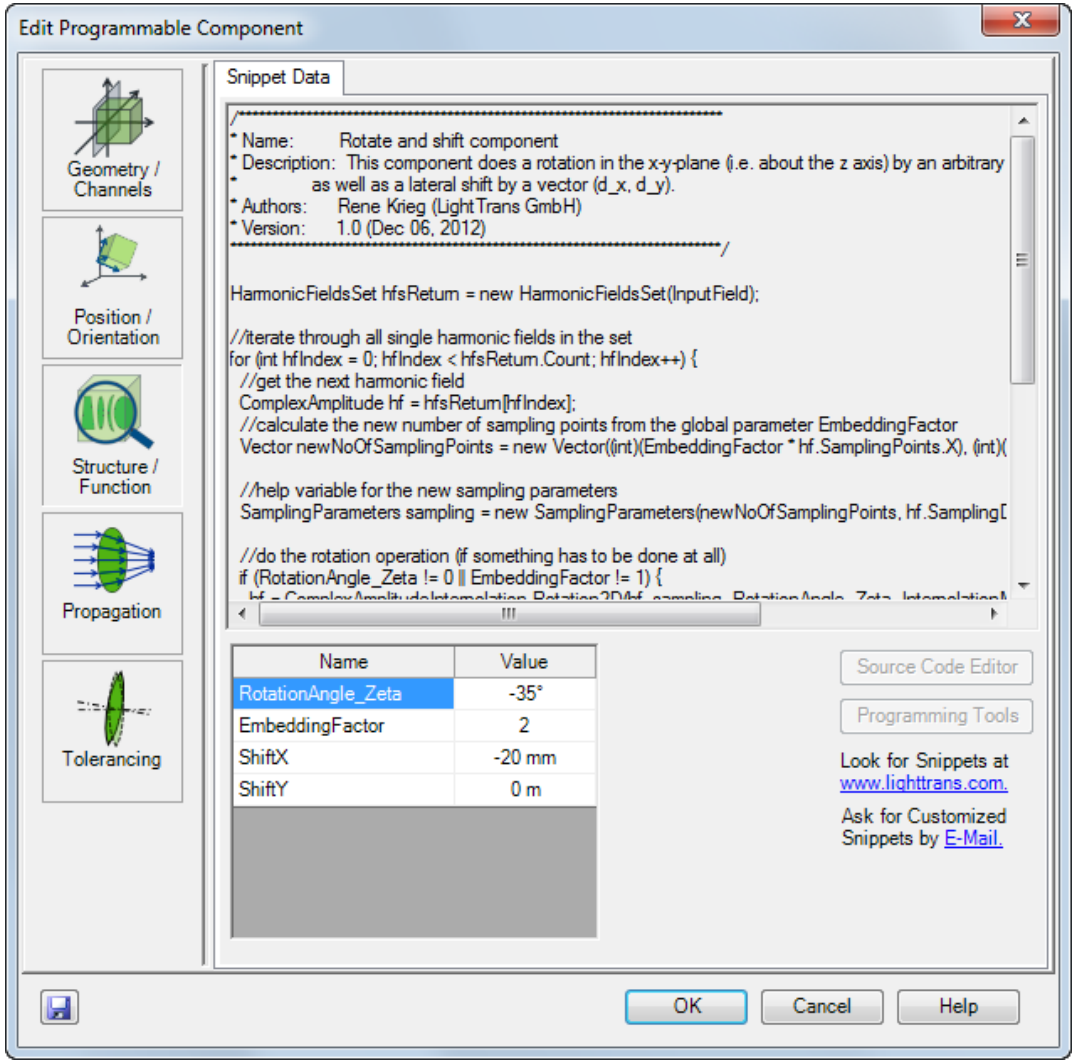
Concept of Programmable Component

- The programmable component allows to change the incoming field in an arbitrary way by the specification of a snippet.
- Within the Light Path Diagram the geometry of the programmable component is only a plane. So the output field of the programmable component has the same position as the input field in the light path. In case of reflection, the orientation is changed as for the reflection at a single interface.
- This concept will be extended later in 2013 to allow tilts and to define an extension of the component.

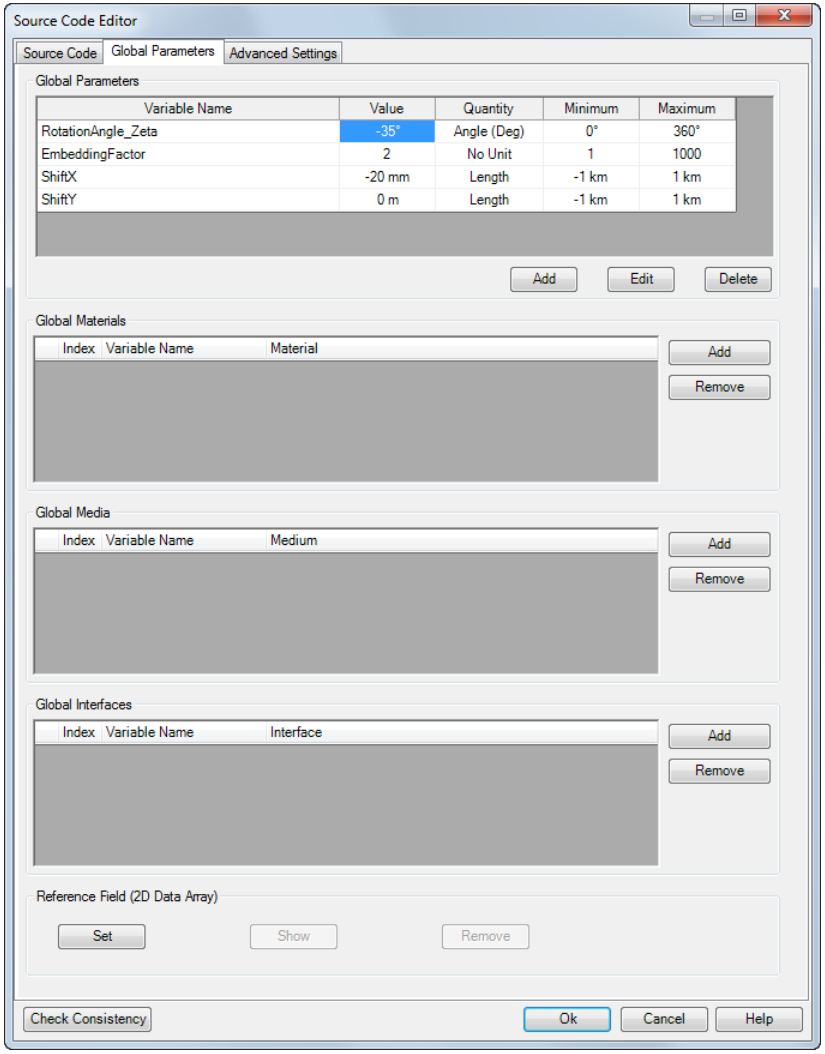
Concept of Programmable Component

- For the internal geometry specification, the programmable component allows to specify a list of optical interfaces, optical materials and optical media and a reference data array.
- These system building blocks can be used within the snippet which calculates the output field from the input field.
- If the programmable component is used within the Parameter Run, the global parameters of the snippet and also the parameters of the optical interfaces, media and materials can be varied.

Edit Dialog of the Programmable Component



Dialog of the Source Code Editor



- Global Parameters:
  - Values
  - Materials
  - Media
  - Interfaces
  - Data Array

Input/Output: The HarmonicFieldsSet Class

- The type of the light field object which is used for input and output within the programmable component is **HarmonicFieldsSet**.
- A **HarmonicFieldsSet** object consists of a number of harmonic fields which can be accessed by an indexer.
  - HarmonicFieldSet[int i] can be used to set or get the harmonic field at index i.
- Generation of **HarmonicFieldsSet**:
  - Copy-Constructor: new HarmonicFieldsSet(hfsOld).
  - Empty Constructor: new HarmonicFieldsSet().

Input/Output: The HarmonicFieldsSet Class

- Additionally the **HarmonicFieldsSet** (HFS) offers a variety of support methods which can be used to modify the object:
  - Add(ComplexAmplitude ca) (adds a complex amplitude to the HFS).
  - Remove(int i) (removes the complex amplitude at index i).
  - Insert(ComplexAmplitude ca, int i) (inserts the complex amplitude at the index i).

Input/Output: The HarmonicFieldsSet Class

- Typical loop over all members of the **HarmonicFieldsSet**:

```
HarmonicFieldsSet hfsReturn = new HarmonicFieldsSet();

for(int runMember = 0; runMember < InputField.Count; runMember++){
  ComplexAmplitude caCurrent = (ComplexAmplitude)InputField[runMember].Clone();

  /************************************
   * Do something the ComplexAmplitude *
   ************************************/

  hfsReturn.Add(caCurrent);
}

return hfsReturn;
```

Input/Output: The ComplexAmplitude Class

- The **ComplexAmplitude** class within the VirtualLabAPI.dll is used to represent a harmonic field.
- Within a **ComplexAmplitude** object the most important properties are:
  - ca.Wavelength (vacuum wavelength of the ca).
  - ca.IsLocallyPolarized (flag whether locally polarized).
  - ca.Field (matrix of FieldValues in case of global polarization).
  - ca.FieldX, ca.FieldY (matrices of FieldValues in case of local polarization).

Input/Output: The ComplexAmplitude Class

  - ca.JonesVector (Jones vector of the ca, only accessible if ca is globally polarized).
  - ca.EmbeddingMedium (the medium in which the ca is defined).
  - ca.LFO_CoordinateSystem (position and orientation of the ca).
- Additionally the **ComplexAmplitde** class supports an indexer. By calling ca[x,y,false] the data matrix for Ex at the pixel coordinate (x,y) can be easily accessed.
- The **ComplexAmplitude** class also offers a lot of methods to vary or evaluate the harmonic field.

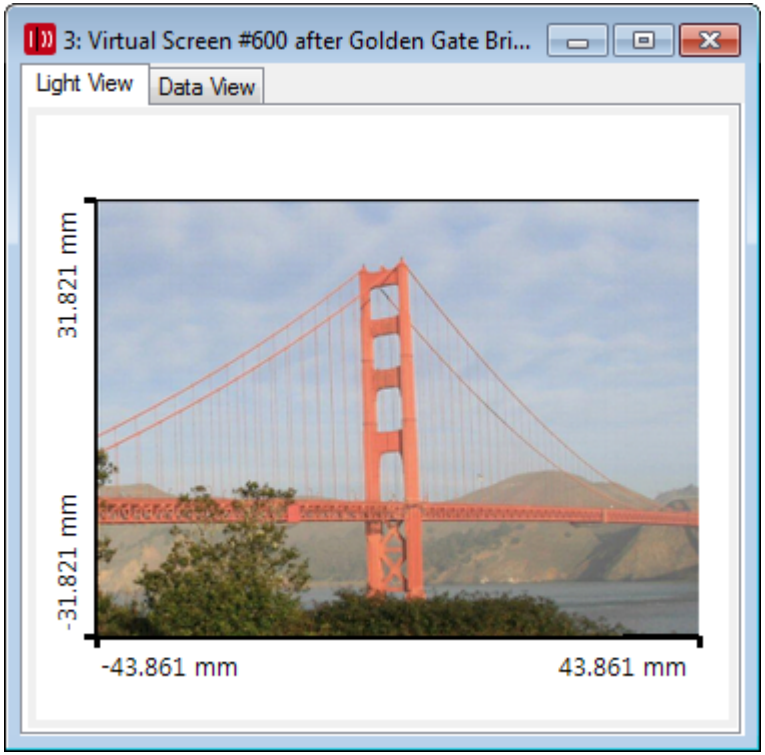
Programmable Component: Rotate and Shift

```
8
9   HarmonicFieldsSet hfsReturn = new HarmonicFieldsSet(InputField);
10
11  //iterate through all single harmonic fields in the set
12  for (int hfIndex = 0; hfIndex < hfsReturn.Count; hfIndex++) {
13    //get the next harmonic field
14    ComplexAmplitude hf = hfsReturn[hfIndex];
15    //calculate the new number of sampling points from the global parameter EmbeddingFactor
16    Vector newNoOfSamplingPoints = new Vector((int)(EmbeddingFactor * hf.SamplingPoints.X), (int)(EmbeddingFactor * hf.SamplingPoints.Y));
17
18    //help variable for the new sampling parameters
19    SamplingParameters sampling = new SamplingParameters(newNoOfSamplingPoints, hf.SamplingDistance);
20
21    //do the rotation operation (if something has to be done at all)
22    if (RotationAngle_Zeta != 0 || EmbeddingFactor != 1) {
23      hf = ComplexAmplitudeInterpolation.Rotation2D(hf, sampling, RotationAngle_Zeta, InterpolationMethod.Cubic6P);
24    }
25
26    if (ShiftX != 0 || ShiftY != 0) {
27      //determine new zero point that considers the shift
28      VectorD newZero = new VectorD(-ShiftX, -ShiftY);
29
30      //do the shift operation via Interpolation method
31      hf = ComplexAmplitudeInterpolation.Interpolation(hf, sampling, newZero, InterpolationMethod.Cubic6P);
32    }
33
34    //set the rotated and shifted harmonic field back to the set
35    hfsReturn[hfIndex] = hf;
36  }
37
38
39  return hfsReturn;
```

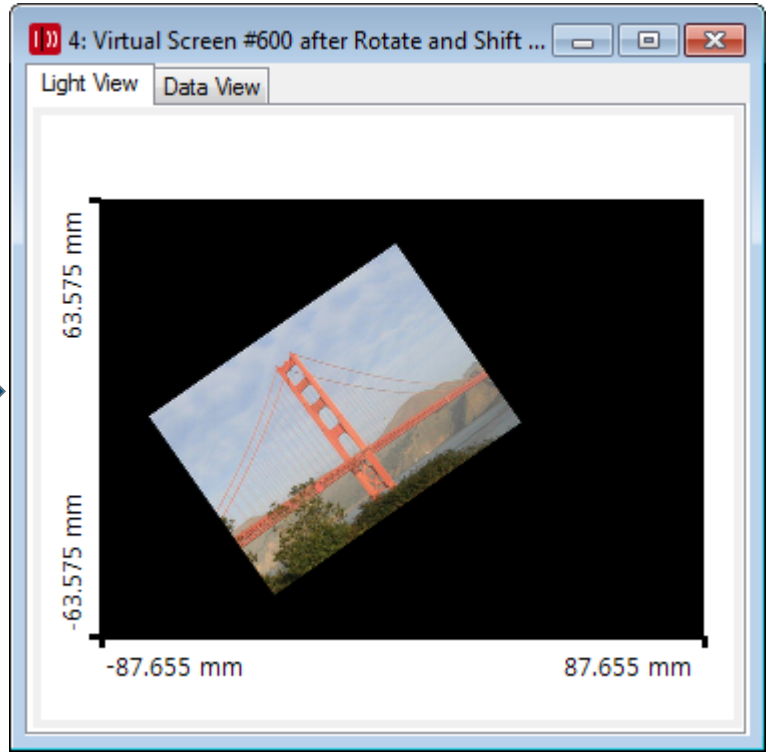
Components Catalog:
Rotate and Shift Component.

Rotate and Shift: Result

Input



Output



Development of Snippets and Modules

VirtualLab Source Code Editor vs. Visual Studio

- The development of snippets can be done by the usage of the source code editor which is integrated in VirtualLab.
- This source code editor supports:
  - Syntax highlighting.
  - (Un)Collapsing (regions).
  - Comment/Uncomment consecutive code lines.
- Alternatively, the development of snippets can be done within Microsoft Visual Studio, which provides additionally the **syntax completion**.

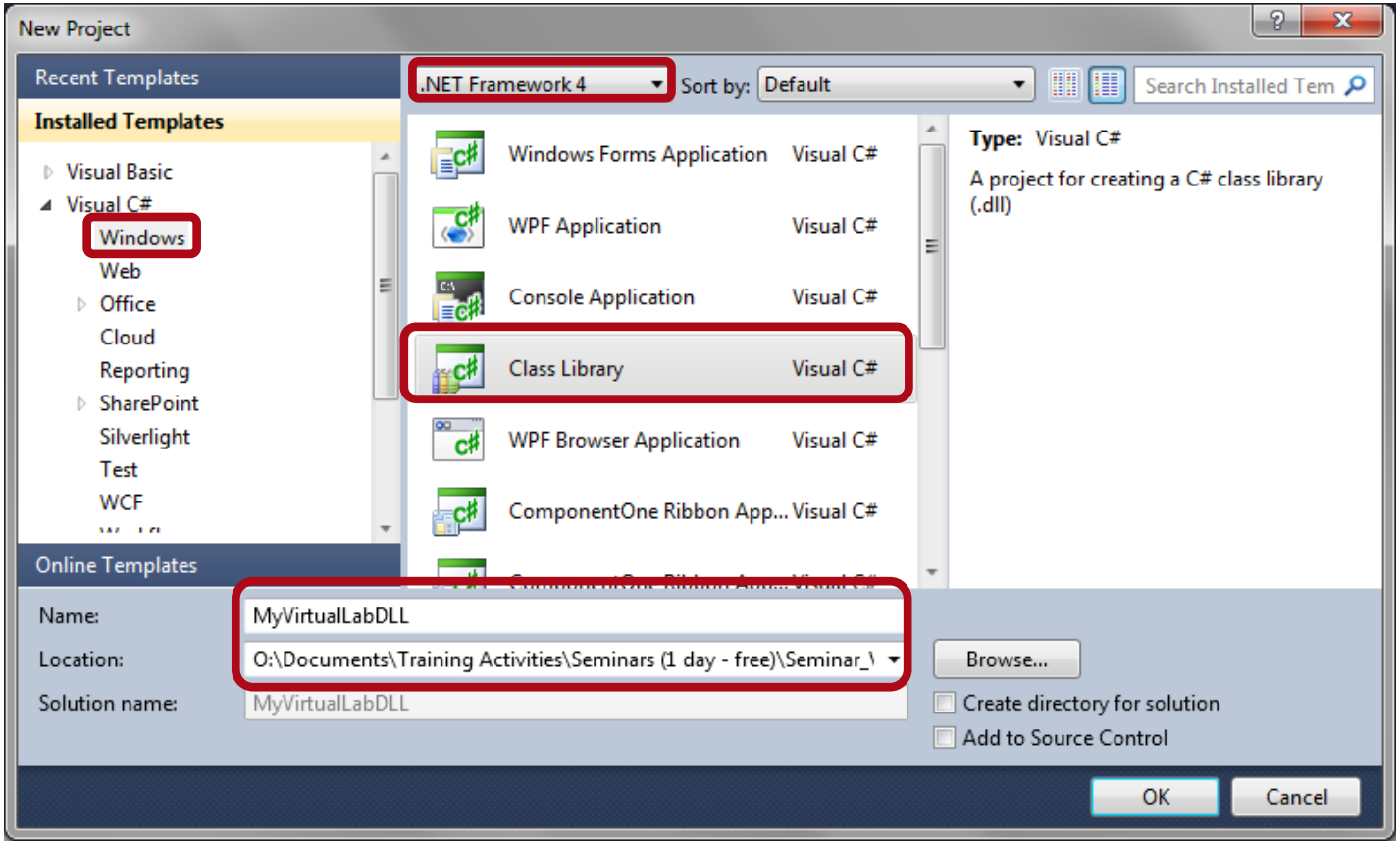
Download and Install Visual Studio

- The Visual Studio Express version can be downloaded for free. (http://www.microsoft.com/germany/express/download/default.aspx)
- The main differences between Professional and Express version are:
  - No support of SQL Server.
  - No usage of Team Foundation Manager.
  - Development of application in the language F#.

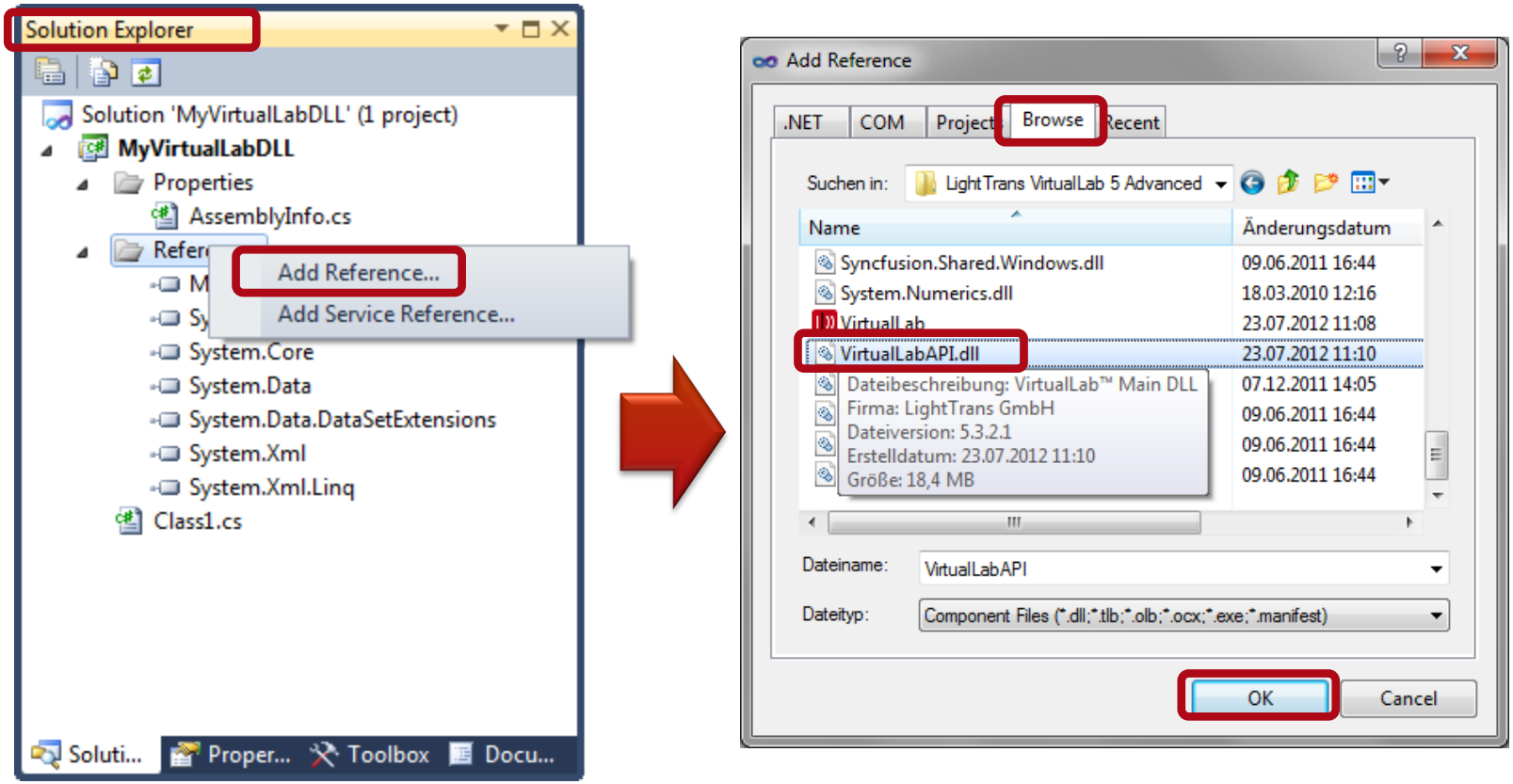
Visual Studio 2010 Express helps you to develop Snippets in VirtualLab.

How to Setup Visual Studio Project

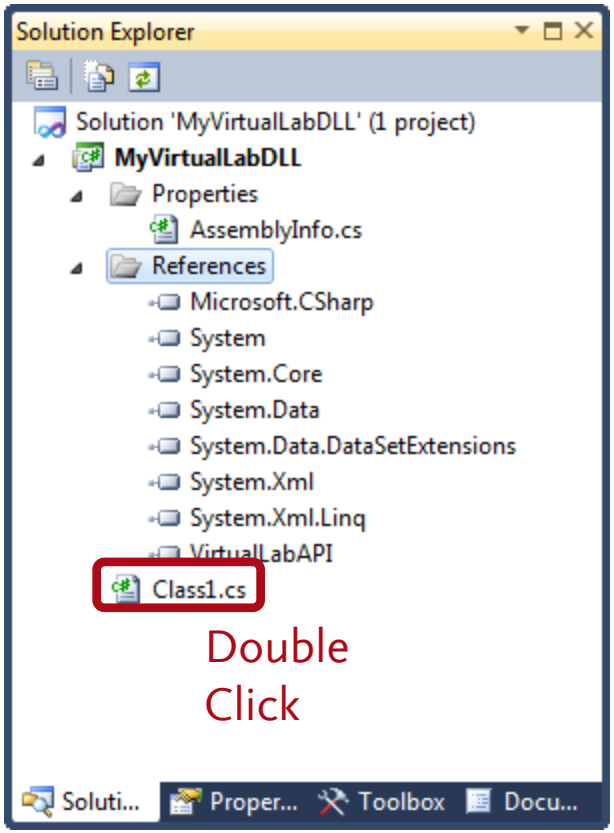
- Open Visual Studio 2010 and select New Project



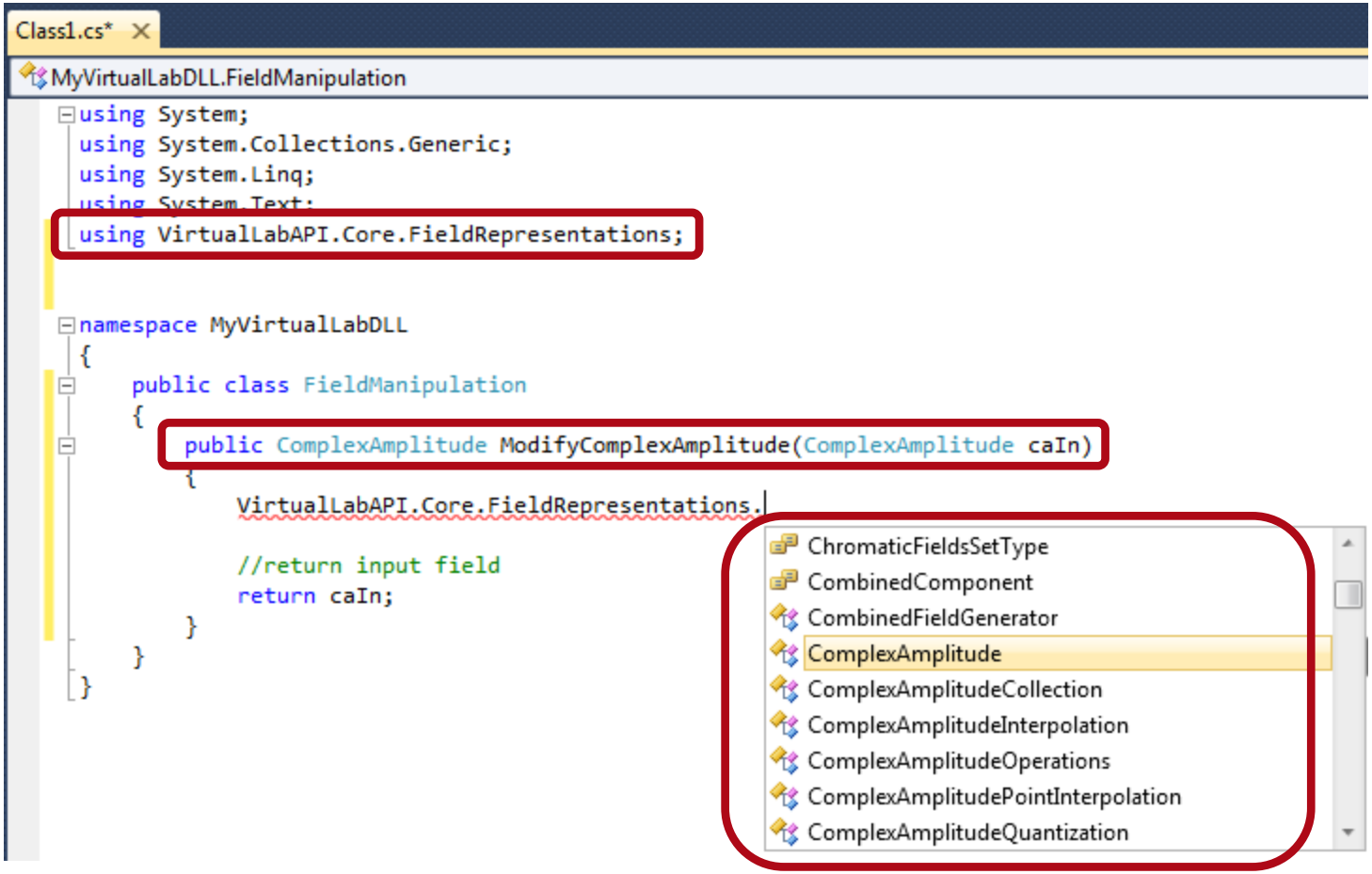
How to Setup Visual Studio Project



How to Setup Visual Studio Project



Example: Syntax Completion

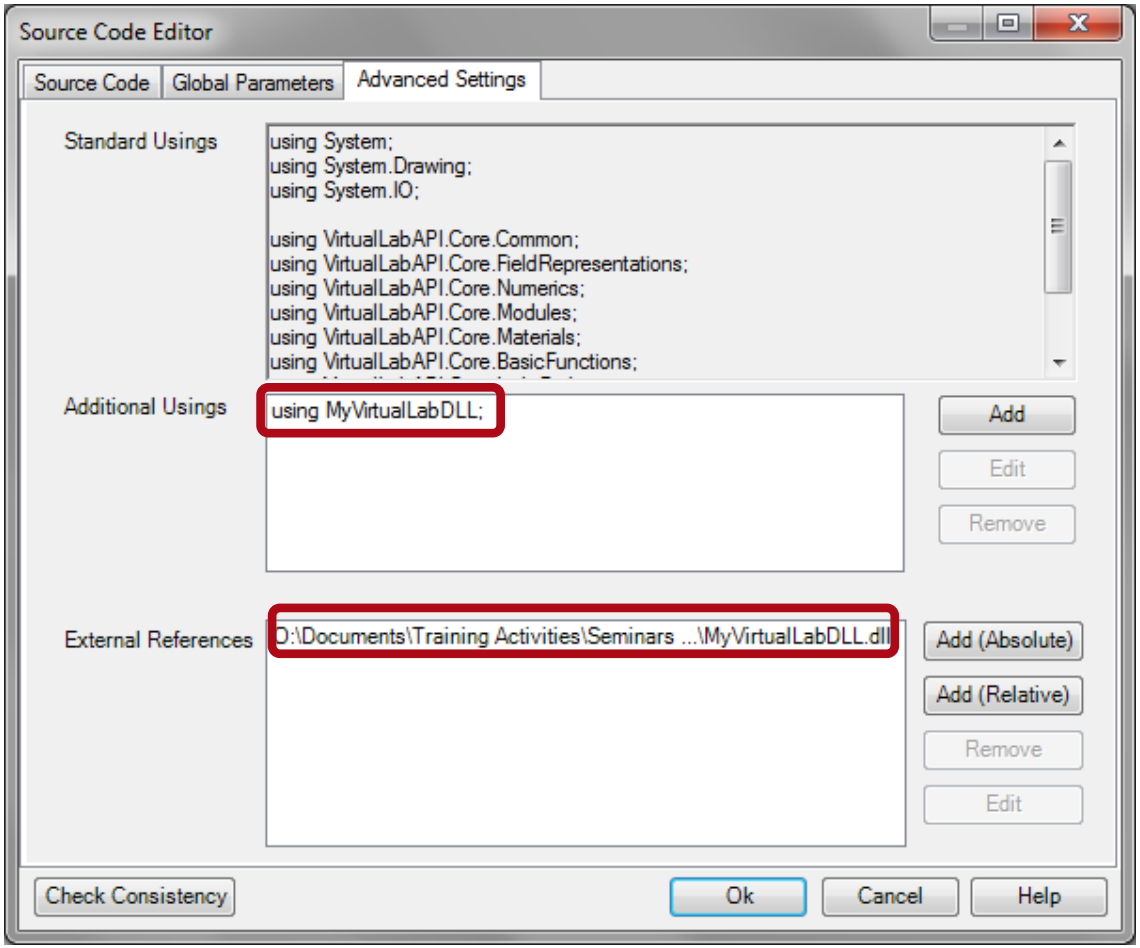


Snippets and DLLs

- Using a Visual Studio project simplifies the development of snippets and modules since syntax completion is available.
- So far the entire code was written in snippets.
  - The source code written in Visual Studio can be copied into the snippet.
- From the snippet, it is also possible to call functions from external DLLs.
  - Visual Studio can be used to write such DLLs. Then also functions and class definitions can be used for more complex applications.
  - The DLL is to be added as external reference.

Advanced Settings

- Advanced Settings of Snippet:



Example: Using a DLL

- Code within DLL:

```
namespace MyVirtualLabDLL
{
    public class FieldManipulation
    {
        /// <summary> ...
        public ComplexAmplitude MulitplyComplexAmplitudeWithFactor(ComplexAmplitude caIn,
                                                                   double factor)
        {
            ComplexAmplitude caOut = (ComplexAmplitude)caIn.Clone();
            //run through all sampling points in x and y direction
            for (int runSamplingY = 0; runSamplingY < caOut.SamplingParameters.SamplingPoints.Y; runSamplingY++){
                for (int runSamplingX = 0; runSamplingX < caOut.SamplingParameters.SamplingPoints.X; runSamplingX++){
                    //check whether locally or globally polarization
                    if (caOut.IsGloballyPolarized){
                        //use Field property to multiply factor on data
                        caOut.Field[runSamplingX, runSamplingY] *= factor;
                    }
                    else{
                        //use FieldX and FieldY property to multiply factor on data
                        caOut.FieldX[runSamplingX, runSamplingY] *= factor;
                        caOut.FieldY[runSamplingX, runSamplingY] *= factor;
                    }
                }
            }

            //return input field
            return caOut;
        }
    }
}
```

Example: Using a DLL

- Code within Snippet:

```
1   //create output HarmonicFieldsSet
2   HarmonicFieldsSet hfsReturn = new HarmonicFieldsSet();
3
4   //run through all members of the harmonic fields set
5   for(int runMember = 0; runMember < InputField.Count; runMember++){
6     //generate instance of the class which is defined within DLL
7     FieldManipulation manipulation = new FieldManipulation();
8     //call method of the class that multiplies the factor on the input field
9     ComplexAmplitude caMulitply = manipulation.MulitplyComplexAmplitudeWithFactor(InputField[runMember],
10                                                                                  FactorToMultiply);
11
12    //add the manipulated ComplexAmplitude to the output HarmonicFieldsSet
13    hfsReturn.Add(caMulitply);
14  }
15  //return the output harmonic fields set
16  return hfsReturn;
```

Usage of C++ DLLs

- Also C++-DLLs can be used.
  - In the C++ DLL use the “extern” statement:

```
#ifdef __cplusplus
extern "C"
#endif
HEADER_USER_DLL void SetFrequency(double frequency);
```

  - Write a C# DLL and use “DllImport” statement to provide access to C++ function from C#.

```
[DllImport(@".\User.dll")]
public static extern void SetFrequency(double frequency);
```

  - Use imported functions e.g. as follows:

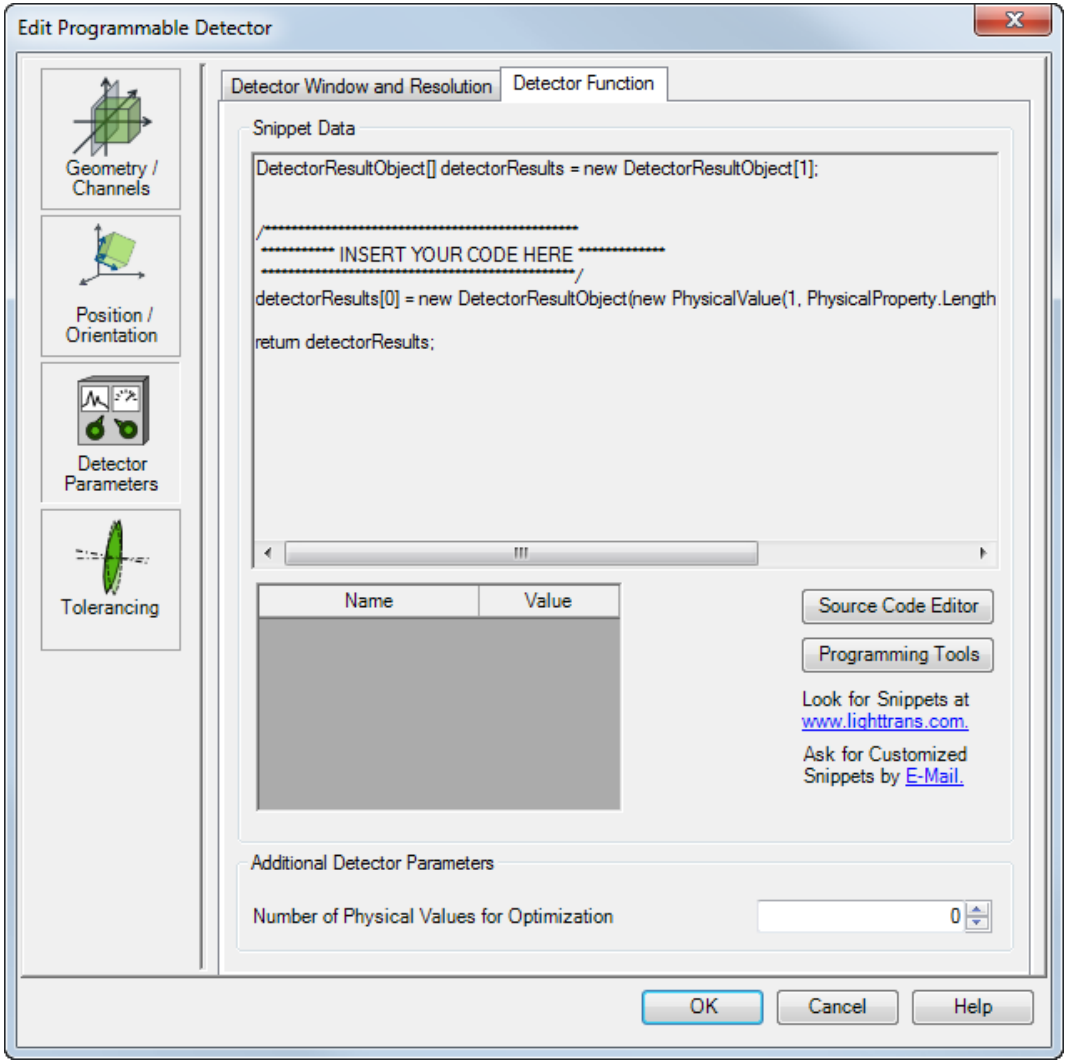
```
UserDLL.SetFrequency(2.0 * Math.PI / wavelength);
```

Programmable Detector

Concept of Programmable Detector

- The programmable detector of Virtual Lab enables the user to specify his own merit function in his system analysis.
- The programmable detector is defined by a snippet that evaluates the incident field (type: **HarmonicFieldsSet**) and generates an arbitrary number of detector result objects.
- Additionally the user can specify the number of numerical values that shall be used within the parametric optimization. This setting is done within the edit dialog of the detector.

Edit Dialog of the Programmable Detector



Detector Results

- The programmable detector allows a snippet that generates an array of DetectorResultObjects dependent on the incident field.
- Example:

```
//declare array of DetectorResultObject of dimension 1
DetectorResultObject[] detectorResults = new DetectorResultObject[1];
//specify first detector result to be a PhysicalValue with the detectorname "My Detector"
detectorResults[0] = new DetectorResultObject(new PhysicalValue(1, PhysicalProperty.Length, "My Detector Result"),
                                              "My Detector");
```

- Typical data types within DetectorResultObjects:
  - List<PhysicalValue>
  - DataArray1D
  - DataArray2D

Output: List<Physical Value>

- For the output of numerical data the **List<PhysicalValues>** has to be used.
- A **PhysicalValue** object consists of three different information (value, physical property, comment).
- The **PhysicalValue** class is located within the namespace `VirtualLabAPI.Core.Numerics`.
- Example:

```
//generate new  list of physical values
List<PhysicalValue> listResults = new List<PhysicalValue>();
//add a new physical value [1m, Comment: Length) to the list
listResults.Add(new PhysicalValue(1, PhysicalProperty.Length, "Distance"));
```

Output: DataArray1D

- To generate DataArray1D several constructors are available.
- Visual Studio supports with completion.
- Namespace: `VirtualLabAPI.Core.Numerics`
- Example:

```
//the data within the data array
double[,] dataAndCoordsOfDataArray = new double[4, 2]{{1,2},
                                                      {2,3},
                                                      {3,4},
                                                      {4,5}};

DataArray1D dArray = new DataArray1D(dataAndCoordsOfDataArray, //the data
                                     PhysicalProperty.ElectricalField, //physical property of data
                                     "Electric Field Values", //comment of data
                                     PhysicalProperty.Length, //physical property of coordinates
                                     "Z"); //comment of coordinates
```

Output: DataArray2D

- To generate DataArray2D several constructors are available.
- Visual Studio supports with completion.
- Namespace: `VirtualLabAPI.Core.Numerics`
- Example:

```
//generate data field of dimension 20x20
ComplexField cfData = new ComplexField(new Vector(20,20));
//fill with 2 (only for demonstration purpose)
cfData.Fill(2);
//generate new data array
DataArray2D dArray2D = new DataArray2D(new ComplexFieldArray(cfData), //the data within the dataarray
                                       new PhysicalProperty[] { PhysicalProperty.AngleDeg }, //physical property of data
                                       new string[] { "My Angles" }, //comment of data
                                       0.2, //sampling distance x-coordinate
                                       -10, //start coordinate x-coordinate
                                       PhysicalProperty.Length, //physical property of x-coordinate
                                       "X", //comment of x-coordinate
                                       1,//sampling distance y-coordinate
                                       0,//start coordinate y-coordinate
                                       PhysicalProperty.Percentage,//physical property of y-coordinate
                                       "Power"); //comment of y-coordinate
```

Remarks on Data Arrays

- DataArray1D and DataArray2D are complex data types which are used for standard output within VirtualLab.
- The sampling of Data Arrays can be equidistant as well as non-equidistant.
- To specify whether an equidistant or an non-equidistant data array shall be generated, the appropriate constructor has to be used.

Examples

- Examples of Programmable Detectors
  - Snippet 028: Diffractive Optics Merit Functions for Harmonic Fields Set Detector
  - Snippet 027: Coherence Detector

Summary

- VirtualLab allows to include user defined simulation techniques enhancing the field tracing methods.
- Tutorials, Manual and Programming Reference give useful information for the programming.
- Further support (to be paid, please contact LightTrans for an offer) can be obtained:
  - Courses on programming given by LightTrans.
  - Support for software solution developed by customers.
  - Customized software solutions developed by LightTrans.