Tutorial 501 (2.0)

Using MATLAB® Functions from VirtualLab™ Snippets and Modules

Author: Christian Hellmann (LightTrans)

Related Tutorials: Tutorial 507

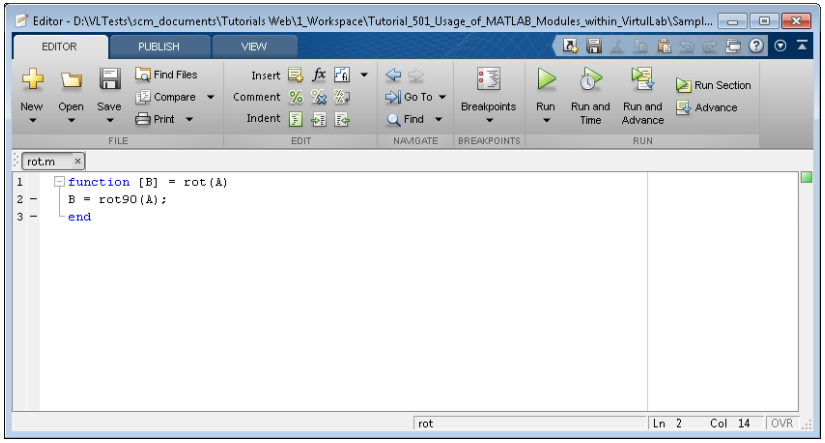
Requirements: VirtualLab™ 5.5 – Starter Toolbox

LIGHTTRANS

Introduction

- This tutorial gives an introduction how MATLAB® functions can be accessed from VirtualLab™ snippets and modules.
- This functionality is demonstrated by the programmable component that performs a simple rotation of the input field by the usage of MATLAB.
- For technical hints and more general information on programming in VirtualLab we refer to Tutorial 507: “Programmable Building Blocks, Components and Detectors of VirtualLab”.

The MATLAB Module



- It is possible to define MATLAB functions within the MATLAB .m-file.
- These functions are used to enter the MATLAB code.
- In this example we define a function rot that rotates the input matrix A and returns the rotated matrix B.

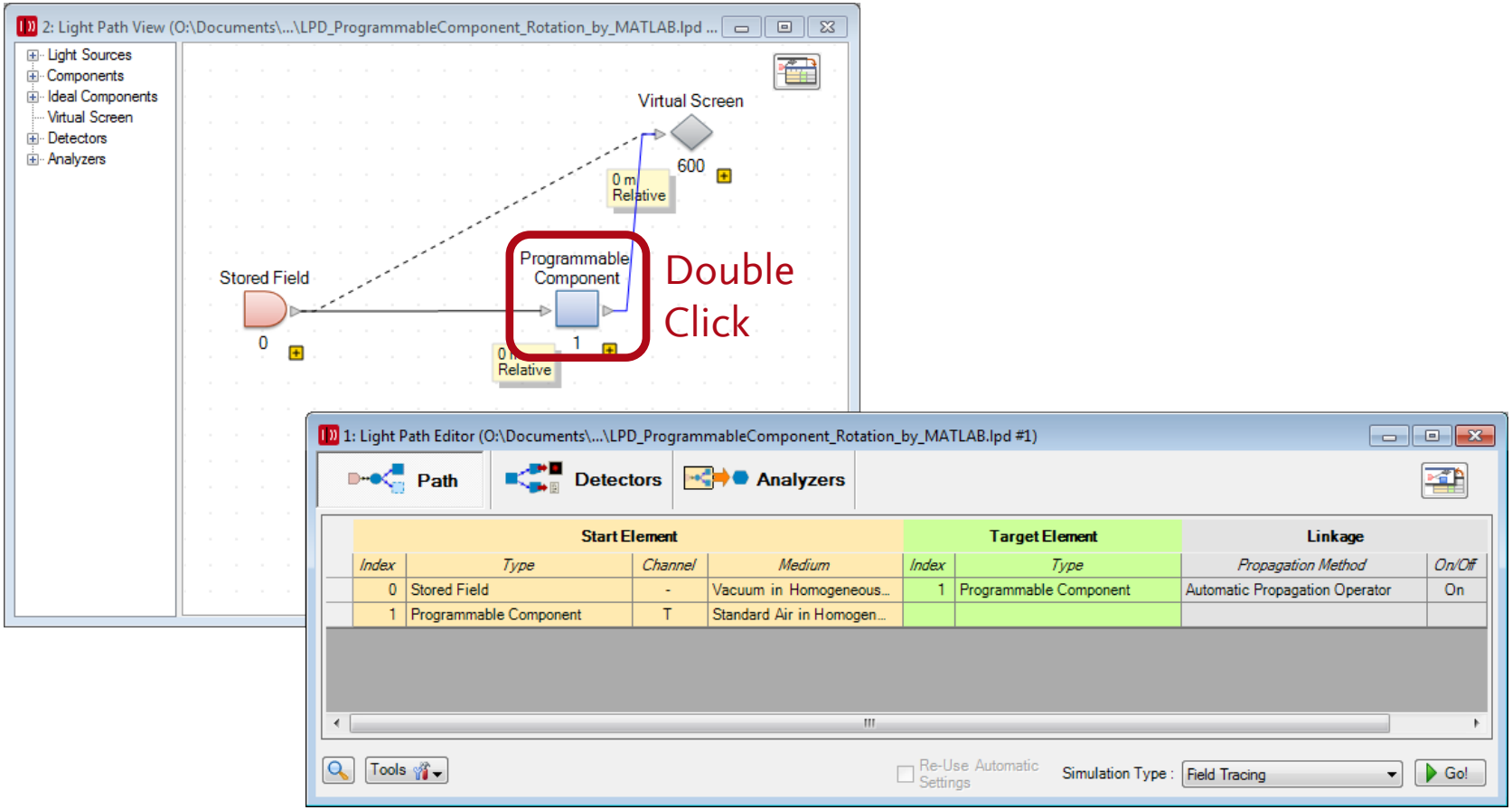
Build .NET DLL from MATLAB

- MATLAB supports the compilation of *.m files to generate a .NET DLL.
- The .NET compilation can be triggered by the MATLAB command “deploytool”.
- This tutorial will not explain the concrete usage of the MATLAB compiler. These information can be found at:

http://www.mathworks.com/products/demos/compiler/deploytool/index.html

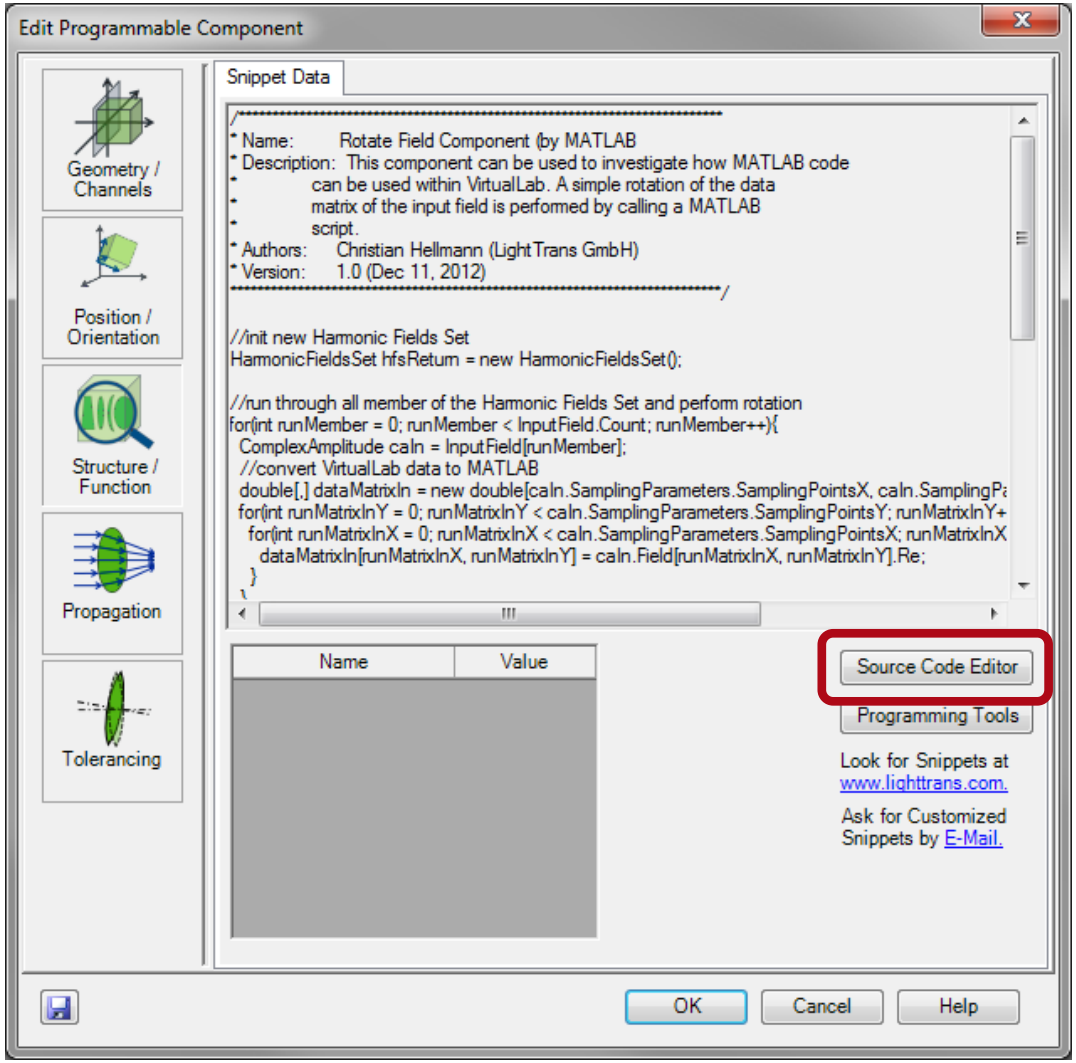
- The result of the compilation is a .NET DLL that can be used within VirtualLab.

Sample Light Path Diagram



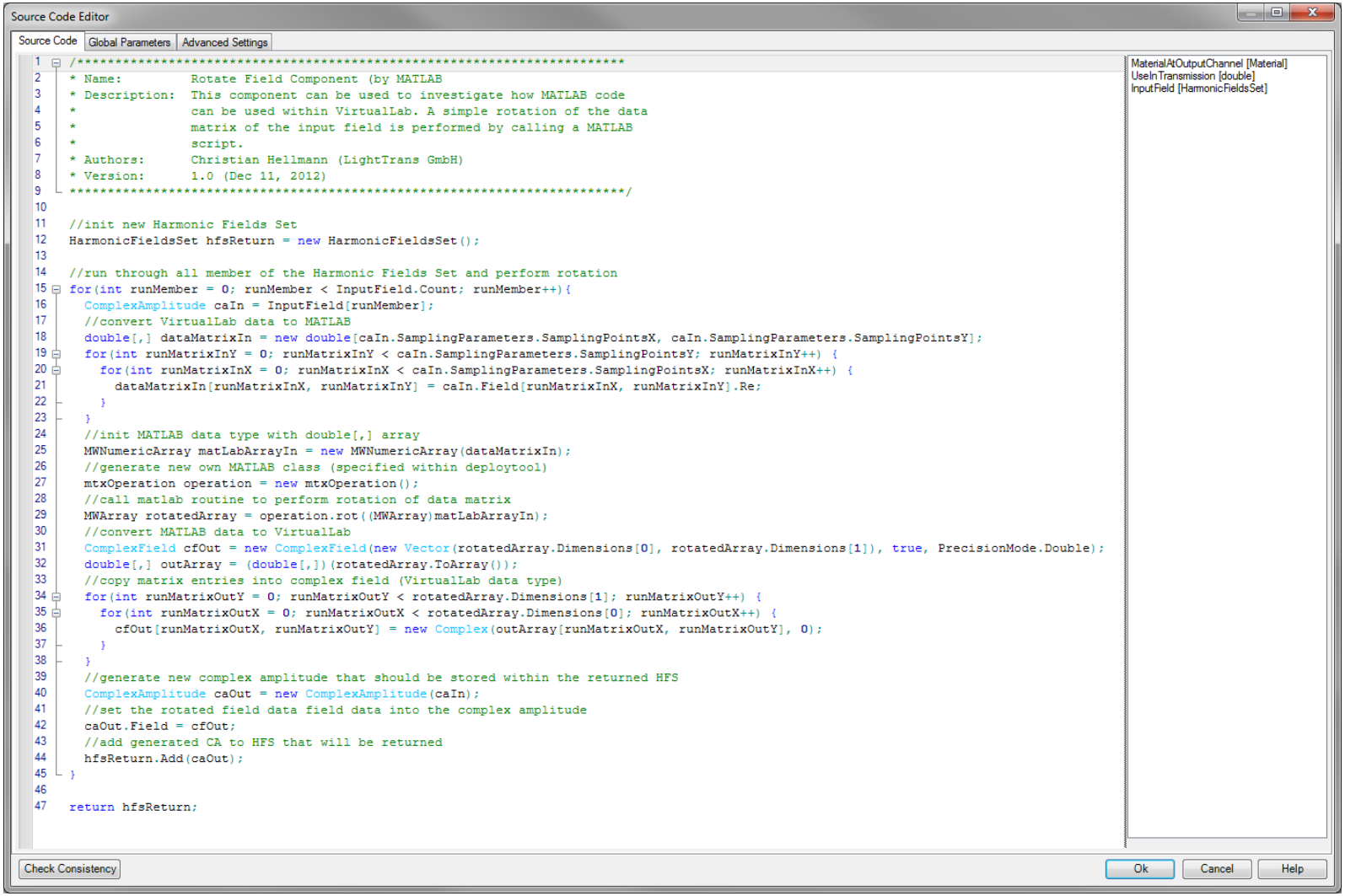
Results in

Edit Dialog of Programmable Component



Results in

Source Code Editor




```
/*************************************************************************
 * Name:         Rotate Field Component (by MATLAB
 * Description:  This component can be used to investigate how MATLAB code
 *               can be used within VirtualLab. A simple rotation of the data
 *               matrix of the input field is performed by calling a MATLAB
 *               script.
 * Authors:      Christian Hellmann (LightTrans GmbH)
 * Version:      1.0 (Dec 11, 2012)
 *************************************************************************/

//init new Harmonic Fields Set
HarmonicFieldsSet hfsReturn = new HarmonicFieldsSet();

//run through all member of the Harmonic Fields Set and perform rotation
for(int runMember = 0; runMember < InputField.Count; runMember++){
  ComplexAmplitude caIn = InputField[runMember];
  //convert VirtualLab data to MATLAB
  double[,] dataMatrixIn = new double[caIn.SamplingParameters.SamplingPointsX, caIn.SamplingParameters.SamplingPointsY];
  for(int runMatrixInY = 0; runMatrixInY < caIn.SamplingParameters.SamplingPointsY; runMatrixInY++) {
    for(int runMatrixInX = 0; runMatrixInX < caIn.SamplingParameters.SamplingPointsX; runMatrixInX++) {
      dataMatrixIn[runMatrixInX, runMatrixInY] = caIn.Field[runMatrixInX, runMatrixInY].Re;
    }
  }
  //init MATLAB data type with double[,] array
  MWNumericArray matLabArrayIn = new MWNumericArray(dataMatrixIn);
  //generate new own MATLAB class (specified within deploytool)
  mtxOperation operation = new mtxOperation();
  //call matlab routine to perform rotation of data matrix
  MWArray rotatedArray = operation.rot((MWArray)matLabArrayIn);
  //convert MATLAB data to VirtualLab
  ComplexField cfOut = new ComplexField(new Vector(rotatedArray.Dimensions[0], rotatedArray.Dimensions[1]), true, PrecisionMode.Double);
  double[,] outArray = (double[,])(rotatedArray.ToArray());
  //copy matrix entries into complex field (VirtualLab data type)
  for(int runMatrixOutY = 0; runMatrixOutY < rotatedArray.Dimensions[1]; runMatrixOutY++) {
    for(int runMatrixOutX = 0; runMatrixOutX < rotatedArray.Dimensions[0]; runMatrixOutX++) {
      cfOut[runMatrixOutX, runMatrixOutY] = new Complex(outArray[runMatrixOutX, runMatrixOutY], 0);
    }
  }
  //generate new complex amplitude that should be stored within the returned HFS
  ComplexAmplitude caOut = new ComplexAmplitude(caIn);
  //set the rotated field data field data into the complex amplitude
  caOut.Field = cfOut;
  //add generated CA to HFS that will be returned
  hfsReturn.Add(caOut);
}

return hfsReturn;
```

Explanation: Source Code

- For each member of the input field (Harmonic Fields Set) the rotation is performed using MATLAB.
- In order to communicate with MATLAB the data types defined in the MCR (MATLAB Compiler Runtime) have to be generated from the VirtualLab field data.

```
//init MATLAB data type with double[,] array
MWNumericArray matLabArrayIn = new MWNumericArray(dataMatrixIn);
//generate new own MATLAB class (specified within deploytool)
mtxOperation operation = new mtxOperation();
//call matlab routine to perform rotation of data matrix
MWArray rotatedArray = operation.rot((MWArray)matLabArrayIn);
```

Explanation: Source Code

- Typical MATLAB data types are:
  - MathWorks.MATLAB.NET.Arrays.MWArray;
  - MathWorks.MATLAB.NET.Arrays.MWNumericArray;
- For detailed information please check the help/ documentation of the MCR.

Explanation: Source Code

- Action with MATLAB: an instance of the .NET class within the built .NET DLL has to be generated. The name of the class that is to be generated can be specified within the deployment tool of MATLAB. (mtxOperation)
- The generated instance of the .NET class allows to call the function defined within the original m-file (rot).

```
//init MATLAB data type with double[,] array
MWNumericArray matLabArrayIn = new MWNumericArray(dataMatrixIn);
//generate new own MATLAB class (specified within deploytool)
mtxOperation operation = new mtxOperation();
//call matlab routine to perform rotation of data matrix
MWArray rotatedArray = operation.rot((MWArray)matLabArrayIn);
```

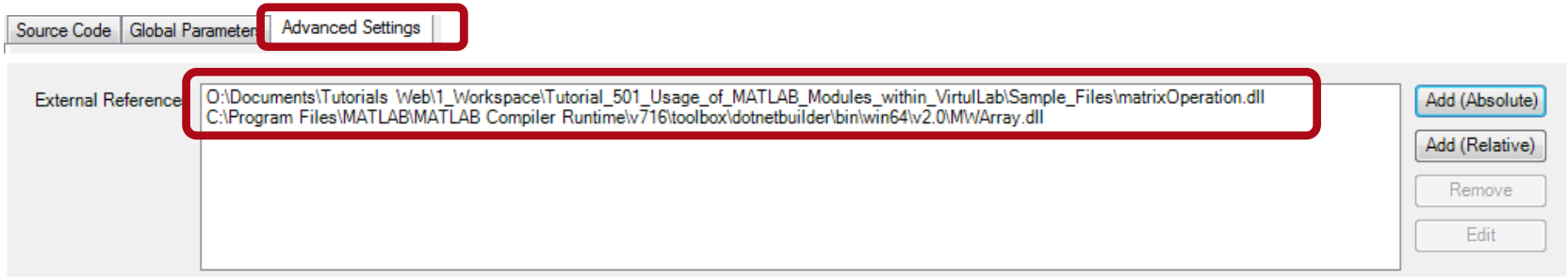
Explanation: Source Code

- After calling the rotate function the MATLAB data have to be re-converted to the VirtualLab data types for field data.

```
//convert MATLAB data to VirtualLab
ComplexField cfOut = new ComplexField(new Vector(rotatedArray.Dimensions[0], rotatedArray.Dimensions[1]),
double[,] outArray = (double[,])(rotatedArray.ToArray());
//copy matrix entries into complex field (VirtualLab data type)
for(int runMatrixOutY = 0; runMatrixOutY < rotatedArray.Dimensions[1]; runMatrixOutY++) {
  for(int runMatrixOutX = 0; runMatrixOutX < rotatedArray.Dimensions[0]; runMatrixOutX++) {
    cfOut[runMatrixOutX, runMatrixOutY] = new Complex(outArray[runMatrixOutX, runMatrixOutY], 0);
  }
}
//generate new complex amplitude that should be stored within the returned HFS
ComplexAmplitude caOut = new ComplexAmplitude(caIn);
//set the rotated field data field data into the complex amplitude
caOut.Field = cfOut;
```

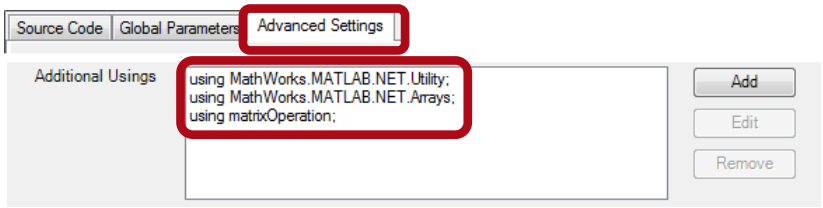
- Finally the generated field data are added to the Harmonic Fields Set which is the return value.

External References



- The user has to specify the following external references to ensure the consistency of the snippet:
  - MATLAB – DLL (generated by the deployment tool of MATLAB) (matrixOperation.dll)
  - MCR – DLL that contains the data types that are necessary to communicate with MATLAB.

Additional Usings

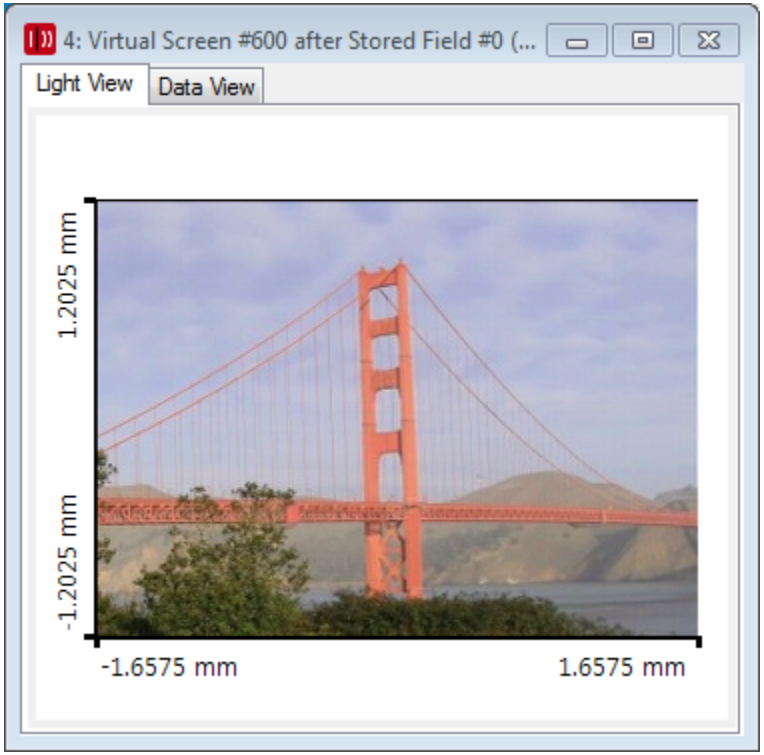


- The user has to specify the namespace of his MATLAB DLL as a using. (in our case this is matrixoperation).
- Additionally the name-spaces of the MCR DLL have to be specified. (see figure on the left side).

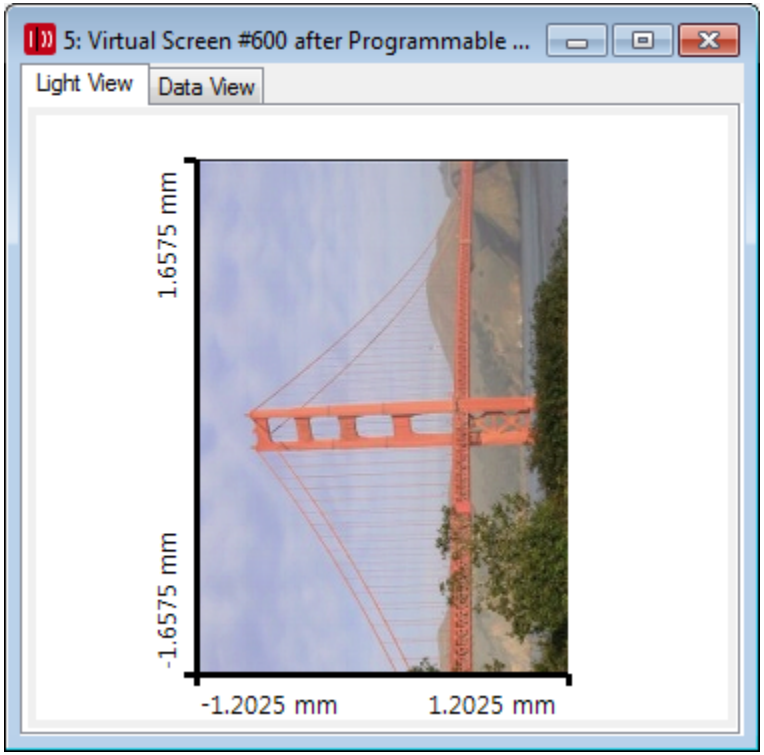
Using external DLLs and user rights

- 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.

Results of the System Propagation



Field generated by the light source.



Field after rotation by the programmable component using MATLAB.

Remarks on the MATLAB Compiler Runtime

- The MCR (MATLAB Compiler Runtime) is free for download from the MATLAB server:

http://www.mathworks.de/products/compiler/mcr/index.html

- It is important to install the correct MCR. The MCR version that is required depends on the MATLAB version used for compiling the MATLAB DLL.
- In the example, version 7.16 of the MCR is required.
- Important: To compile MATLAB code to a .NET DLL a license of MATLAB is required. The use of the .NET DLL built from MATLAB is free. (only the correct version of the MCR has to be installed).

Overview Versions MATLAB - MCR

MATLAB Release	MATLAB Compiler Runtime (MCR) version	MATLAB Compiler Version
R14 (7.0)	7.0	4.0
R14SP1 (7.0.1)	7.1	4.1
R14SP2 (7.0.4)	7.2	4.2
R14SP3 (7.1)	7.3	4.3
R2006a (7.2)	7.4	4.4
R2006b (7.3)	7.5	4.5
R2007a (7.4)	7.6	4.6
R2007b (7.5)	7.7	4.7
R2008a (7.6)	7.8	4.8
R2008b (7.7)	7.9	4.9
R2009a (7.8)	7.10	4.10
R2009b (7.9)	7.11	4.11
R2009bSP1 (7.9.1)	7.12	4.12
R2010a (7.10)	7.13	4.13
R2010b (7.11)	7.14	4.14
R2010bSP1 (7.11.1)	7.14.1	4.14.1
R2011a (7.12)	7.15	4.15
R2011b (7.13)	7.16	4.16
R2012a (7.14)	7.17	4.17
R2012b (8.0)	8.0	4.18

Conclusion

- VirtualLab supports the usage of .NET DLLs within snippets and modules.
- It is possible to use MATLAB code, compiled as .NET DLL, within the system simulation of VirtualLab.
- The programmable component and the option to use MATLAB code allow user defined optical modeling.