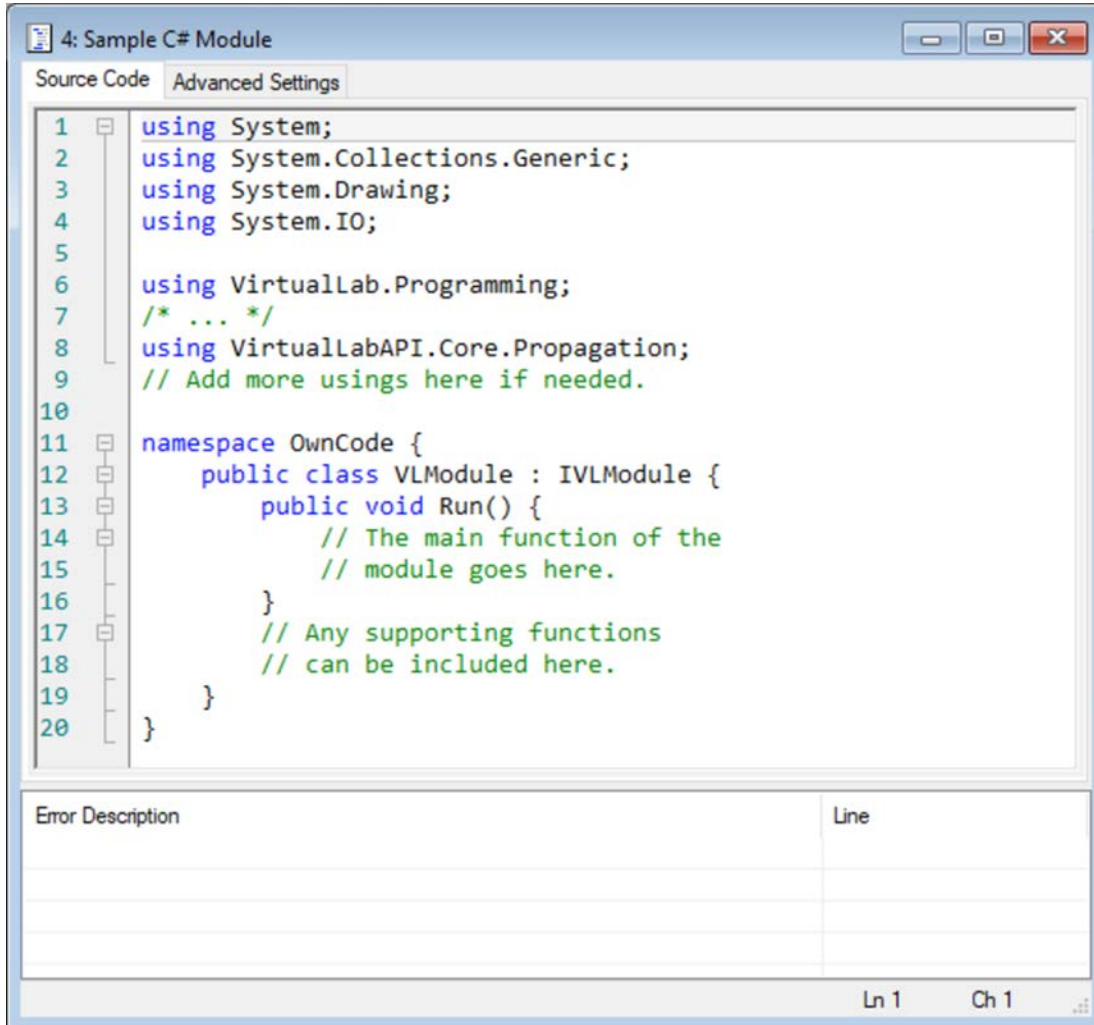


## **How to Work with the C# Module and Example (Computing the Deviation Between Two Fields)**

# Abstract



```
1 using System;
2 using System.Collections.Generic;
3 using System.Drawing;
4 using System.IO;
5
6 using VirtualLab.Programming;
7 /* ... */
8 using VirtualLabAPI.Core.Propagation;
9 // Add more usings here if needed.
10
11 namespace OwnCode {
12     public class VLModule : IVLModule {
13         public void Run() {
14             // The main function of the
15             // module goes here.
16         }
17         // Any supporting functions
18         // can be included here.
19     }
20 }
```

Error Description	Line

Ln 1 Ch 1

Offering maximum versatility for your optical simulations is one of our most central aims. Nowhere is this versatility more apparent than in the Module: while other programmable elements in VirtualLab Fusion (sources, detectors, components, etc.) have a predetermined input and output type, the Module gives the user total freedom of implementation. One reason for that is the fact that it functions outside the Optical Setup document, so it is up to the user's discretion to decide on the input and output of the code: this also means that reading in and delivering the different file types is fundamental.

# Where to Find the Module

The screenshot displays the main interface of Wyrowski VirtualLab Fusion. The 'File' menu is highlighted with a red box and a red arrow pointing to a zoomed-in view. In this view, the 'File' menu is open, and the 'C# Module' option is highlighted with a red arrow and a red box. A red callout box with the text 'Hint: You can also use the keyboard shortcut Ctrl. + M' points to the 'C# Module' option. A second zoomed-in view shows the 'C# Module' dialog box with the 'C# Module' option selected. A red arrow points from this dialog to a code editor window titled '2: Module - C#'. The code editor shows the following source code:

```
1 using System;  
2 using System.Collections.Generic;  
3 using System.Drawing;  
4 using System.IO;  
5  
6 using VirtualLab.Programming;  
7 using VirtualLabAPI.Core.BasicFunctions;  
8 using VirtualLabAPI.Core.Common;  
9 using VirtualLabAPI.Core.DataVisualization;  
10 using VirtualLabAPI.Core.FieldRepresentations;  
11 using VirtualLabAPI.Core.Functions;  
12 using VirtualLabAPI.Core.GeometryDescription;  
13 using VirtualLabAPI.Core.LightPath;  
14 using VirtualLabAPI.Core.Materials;  
15 using VirtualLabAPI.Core.Modules;
```

The code editor also shows an 'Error Description' table with columns 'Error Description' and 'Line'. The status bar at the bottom indicates 'Number of Used Cores (not for Parameter Run): 4', 'CPU Usage: 0%', and 'Physical Memory: 0/32 GB'.

# Writing the Code

```
1 using System;
2 using System.Collections.Generic;
3 using System.Drawing;
4 using System.IO;
5
6 using VirtualLab.Programming;
7 /* ... */
8 using VirtualLabAPI.Core.Propagation;
9 // Add more usings here if needed.
10
11 namespace OwnCode {
12     public class VLModule : IVLModule {
13         public void Run() {
14             // The main function of the
15             // module goes here.
16         }
17         // Any supporting functions
18         // can be included here.
19     }
20 }
```

Error Description	Line

Ln 1 Ch 1

The header of the module. A list of usings is included by default; add more should the need for them arise.

The main function of the module should be included between the curly brackets for Run(), as indicated.

Any additional supporting functions can be defined in the same class, but outside of the "Run" environment.

After compilation (F6) or an attempt to run the module (F5) any compilation errors are shown here.

# Writing the Code

---

- It is of particular importance for the Module to be familiar with the different data types available in VirtualLab, and how to read them in and display them. Some useful examples:
  - `VL_GUI.AskForDouble()` → Prompts the user to enter a value for a `double` parameter. Also exists for `int` and `Complex`.
  - `VL_GUI.WriteToMessagesTab()` or `WriteLineToMessagesTab()` → Displays a string in the Messages tab. The first variant includes no carriage return. A return can be added manually at any point by the user using the special character `\n` inside the `string`.
  - `VL_GUI.ShowDocument()` → Displays a graph of any class which implements the interface `IDocument`. An example of this would be `ComplexAmplitude` or `HarmonicFieldsSet`.
  - `VL_GUI.SelectOpenField()` → Prompts the user to select an open document of type `ComplexAmplitude`. There are similar options for other document types.

# Writing the Code

---

- `ComplexAmplitude` → Object designed to store a monochromatic, equidistantly sampled complex amplitude (transversal distribution of field at a plane). It stores the `ComplexField` for  $E_x$  and  $E_y$ , whether in globally polarized form (one common field function for both and one Jones vector which is constant in the plane) or in locally polarized form (two different functions for  $E_x$  and  $E_y$ ). All other electromagnetic components can be computed from those two on demand, as per Maxwell's equations.
- `HarmonicFieldsSet` → Object type designed to group several instances of `ComplexAmplitude`. For instance, a polychromatic field, which will contain one `ComplexAmplitude` per spectral sample.
- `DataArray2D` → Contains the discrete values defining one or more generally complex functions on a 2D support. These values can be equidistantly or non-equidistantly sampled. The dimensions of both the function and its support are free for the user to define. There exists also a 1D version of the data array.

# Compiling & Running Your Module

The screenshot displays the Wyrowski VirtualLab Fusion (2nd Generation Technology Update [Build 7.4.0.49]) interface. The main window shows a source code editor for a module named "5: Compute Deviation". The code includes comments and function calls, such as `(ComplexAmplitude) (VL_GUI.SelectOpenField("Select field for analysis"));` and `VL_GUI.WriteLineToDisplay(scalingFactor);`. A red hand icon points to the "Compile" button in the toolbar, and another red hand icon points to the "Run" button. The toolbar also includes buttons for "Go!", "Select All", and "Sources". The interface also shows a "Property Browser" on the right and a "Messages" panel at the bottom. The status bar at the bottom indicates "Number of Used Cores (not for Parameter Run): 4", "CPU Usage: 0%", and "Physical Memory: 0/32 GB".

```
29 (ComplexAmplitude) (VL_GUI.SelectOpenField("Select field for analysis"));
30
31 // Declare the output variables
32 double relativeDeviation;
33 double absoluteDeviation;
34 Complex scalingFactor;
35
36 // Compute deviation
37 ComplexAmplitudeOpenComplexAmplitudeOpenComplexAmplitudeOpen
38 caAnalysis,
39 caReference,
40 false,
41 out relativeDeviation;
42 out absoluteDeviation;
43 out scalingFactor;
44 InterpolationMethod;
45
46 // Display results
47 // support function
48 VL_GUI.WriteLineToDisplay(scalingFactor);
49
50 // Compute deviation
```

# **Programming a C# Module That Computes the Standard Deviation Between Two Fields**



# Standard Deviation

---

Given two sampled, complex functions  $f$  and  $g$ , defined on the  $x, y$  plane, the relative standard deviation of  $g$  with respect to  $f$  is defined as

$$\sigma [f(x_m, y_n), g(x_m, y_n)] = \frac{\sum |f(x_m, y_n) - g(x_m, y_n)|^2 \delta x \delta y}{\sum |f(x_m, y_n)|^2 \delta x \delta y} \quad (1)$$

The computation of the absolute deviation would respond to the same expression but without the normalization constant.

Sometimes it is of interest to allow for a complex constant to be multiplied onto  $g(x, y)$  so that the value of the deviation is minimized. This allows us to compare just the shapes of the two functions, without paying attention to the scale. The function implemented in VirtualLab for the calculation of the deviation, which we shall use throughout this example, allows for both possibilities (with and without scaling). The function delivers automatically the value of the complex constant which minimizes the error.

# Where to Find the Module

The screenshot displays the VirtualLab Fusion software interface. The main window title is "Wyrowski VirtualLab Fusion (2nd Generation Technology Update [Build 7.4.0.49])". The menu bar includes File, Start, Sources, Functions, Catalogs, and Windows. The File menu is expanded, showing options like New, Open, Save, and Shaping. A red box highlights the File menu, and a red arrow points to a zoomed-in view of the File menu. In this zoomed view, a red hand icon with the number "1" points to the File menu, and another red hand icon with the number "2" points to the "C# Module" option, which has the keyboard shortcut "Ctrl+M" next to it. A red arrow points from the "C# Module" option to a code editor window titled "2: Module - C#". The code editor shows the following source code:

```
1 using System;  
2 using System.Collections.Generic;  
3 using System.Drawing;  
4 using System.IO;  
5  
6 using VirtualLab.Programming;  
7 using VirtualLabAPI.Core.BasicFunctions;  
8 using VirtualLabAPI.Core.Common;  
9 using VirtualLabAPI.Core.DataVisualization;  
10 using VirtualLabAPI.Core.FieldRepresentations;  
11 using VirtualLabAPI.Core.Functions;  
12 using VirtualLabAPI.Core.GeometryDescription;  
13 using VirtualLabAPI.Core.LightPath;  
14 using VirtualLabAPI.Core.Materials;  
15 using VirtualLabAPI.Core.Modules;
```

A red callout box with a white background and a red border contains the text: "Hint: You can also use the keyboard shortcut Ctrl. + M". The code editor also has an "Advanced Settings" tab and an "Error Description" table at the bottom.

# Test the Code!

## C# Module (Header)

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;

using VirtualLab.Programming;
using VirtualLabAPI.Core.BasicFunctions;
using VirtualLabAPI.Core.Common;
using VirtualLabAPI.Core.DataVisualization;
using VirtualLabAPI.Core.FieldRepresentations;
using VirtualLabAPI.Core.Functions;
using VirtualLabAPI.Core.GeometryDescription;
using VirtualLabAPI.Core.LightPath;
using VirtualLabAPI.Core.Materials;
using VirtualLabAPI.Core.Modules;
using VirtualLabAPI.Core.Numerics;
using VirtualLabAPI.Core.Numerics.Region2D;
using VirtualLabAPI.Core.OpticalSystems;
using VirtualLabAPI.Core.Propagation;

namespace OwnCode {
    public class VLModule : IVLModule {
        public void Run() {
```

# Test the Code!

## C# Module (Run, I)

```
// Read in fields (reference field and field for analysis):
ComplexAmplitude caReference =
    (ComplexAmplitude) (VL_GUI.SelectOpenField("Select reference field"));
ComplexAmplitude caAnalysis =
    (ComplexAmplitude) (VL_GUI.SelectOpenField("Select field for analysis"));

// Declare the outputs:
double relativeDeviation;
double absoluteDeviation;
Complex scalingFactor;

// Compute deviation without scaling:
ComplexAmplitudeOperations.ComputeDeviation(
    caAnalysis, // Field to be analyzed.
    caReference, // Reference field.
    false, // Whether scaling shall be applied or not.
    out relativeDeviation, // The resulting relative deviation (Eq. 1).
    out absoluteDeviation, // The resulting absolute deviation.
    out scalingFactor, // The scaling factor (= 1 if without scaling).
    InterpolationMethod.Nearest); // Interpolation method employed to match the sampling
// of the two fields.
```

# Test the Code!

## C# Module (Run, II)

```
// Display results in Messages window using custom
// support function (see below):
VL_GUI.WriteLineToMessagesTab("--Calculation WITHOUT scaling--");
Display(scalingFactor, relativeDeviation, absoluteDeviation);

// Compute deviation with scaling:
ComplexAmplitudeOperations.ComputeDeviation(
    caAnalysis,
    caReference,
    true,
    out relativeDeviation,
    out absoluteDeviation,
    out scalingFactor,
    InterpolationMethod.Nearest);

// Display results in Messages window using
// custom support function (see below):
VL_GUI.WriteLineToMessagesTab("--Calculation WITH scaling--");
Display(scalingFactor, relativeDeviation, absoluteDeviation);

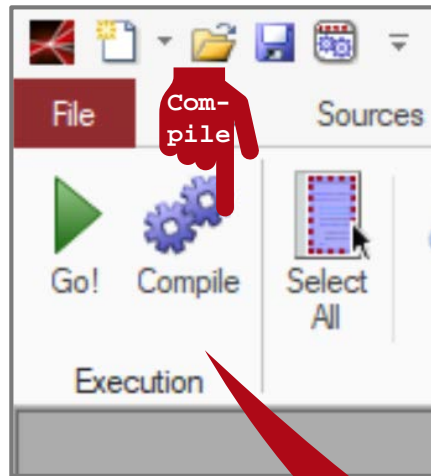
}
```

# Test the Code!

## C# Module (Support Functions)

```
// For illustration purposes, we define a support function
// that delivers the values returned by the calculation to
// the Messages window
public void Display(Complex ScalingFactor, double RelativeDeviation, double AbsoluteDeviation)
{
    VL_GUI.WriteToMessagesTab("Scaling factor: " + ScalingFactor.ToString() +
        "\nRelative deviation: " + RelativeDeviation.ToString() +
        "\nAbsolute deviation: " + AbsoluteDeviation.ToString());
}
}
```

# Compile & Run Your Module



A screenshot of the IDE source code editor window titled '6: Compute Deviation'. The 'Source Code' tab is active, showing the following C# code:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Drawing;
4 using System.IO;
5
6 using VirtualLab.Programming;
7 using VirtualLabAPI.Core.BasicFunctions;
8 using VirtualLabAPI.Core.Common;
9 using VirtualLabAPI.Core.DataVisualization;
10 using VirtualLabAPI.Core.FieldRepresentations;
11 using VirtualLabAPI.Core.Functions;
12 using VirtualLabAPI.Core.GeometryDescription;
13 using VirtualLabAPI.Core.LightPath;
14 using VirtualLabAPI.Core.Materials;
```

Below the code editor is an 'Error Description' table with the following content:

Error Description	Line
Compile successful	

The status bar at the bottom right shows 'Ln 1 Ch 1'.

Attempting to run your module will also automatically compile it!

# Compile & Run Your Module

1

2

3

4

5

Make sure to have two ComplexAmplitude documents open before you run the module!

Messages

```
[11/08/2018 15:23:57] Compile successful
[11/08/2018 15:23:57] Module started
--Calculation WITHOUT scaling--
Scaling factor: 1
Relative deviation: 0.000405459385663326
Absolute deviation: 6.36894111770052E-12--Calculation WITH scaling--
Scaling factor: 0.99
Relative deviation: 0.00030241216857258
Absolute deviation: 4.75027922158937E-12[11/08/2018 15:24:01] Thread finished normally
```

Messages | Detector Results



# Document Information

title	How to Work with the C# Module and Example (Computing the Deviation Between Two Fields)
document code	CZT.0101
version	1.0
toolbox(es)	Starter Toolbox
VL version used for simulations	7.4.0.49
category	Feature Use Case
further reading	<ul style="list-style-type: none"><li>- <a href="#"><u>Programming a Module That Smooths the Edge of a Structure</u></a></li><li>- <a href="#"><u>Programming a Module That Computes the Standard Deviation between Two Harmonic Fields</u></a></li></ul>