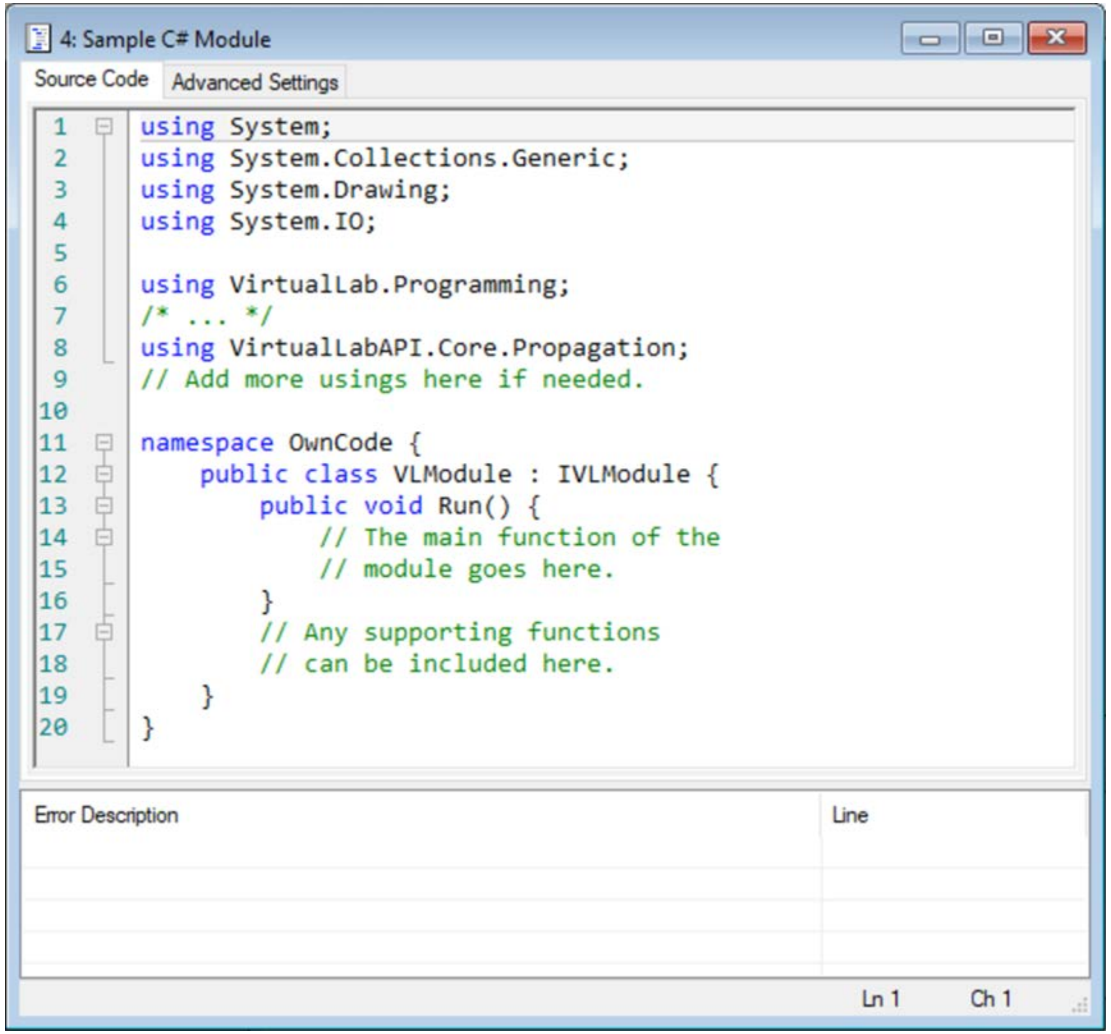
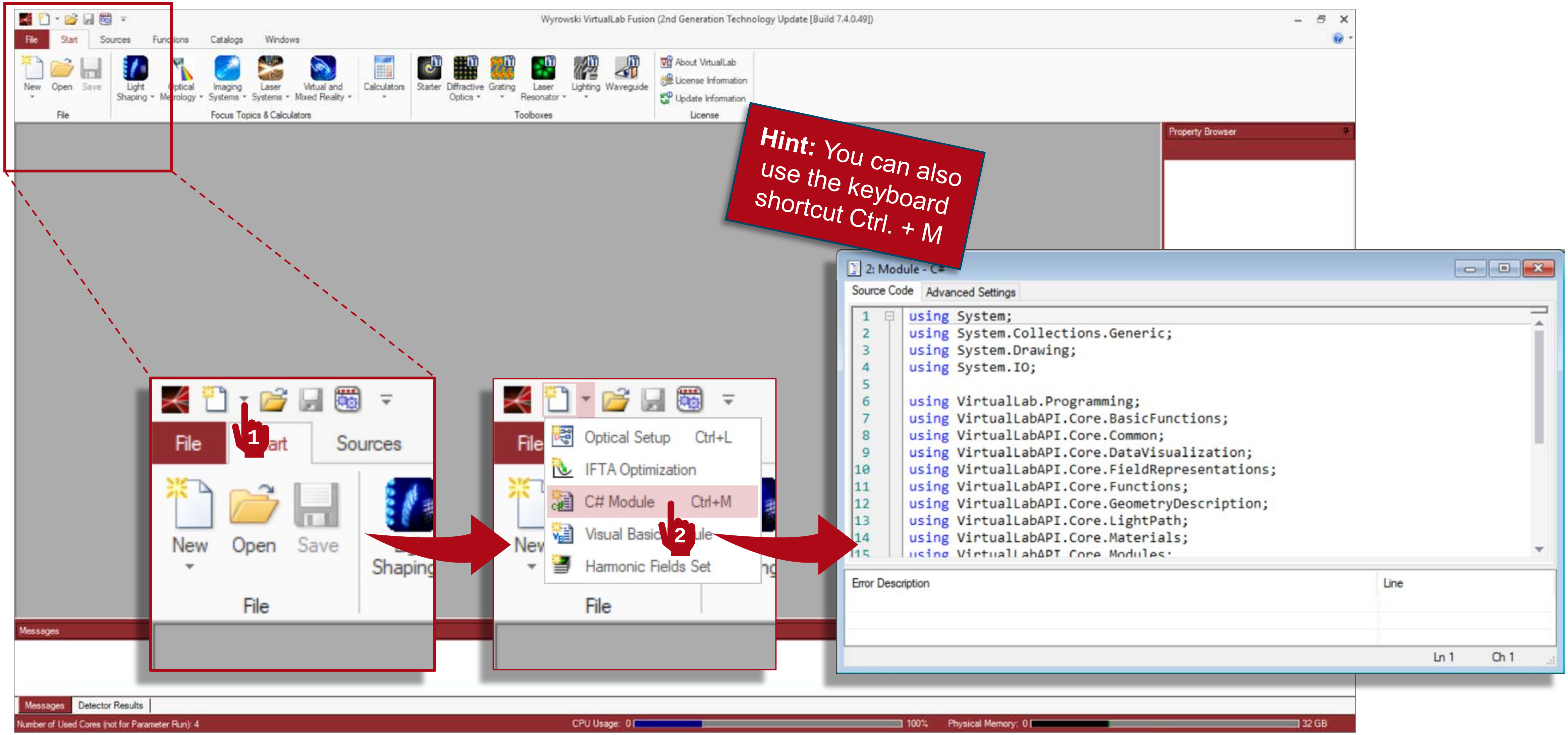
How to Work with the C# Module and Example (Computing the Deviation Between Two Fields)

Abstract



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.

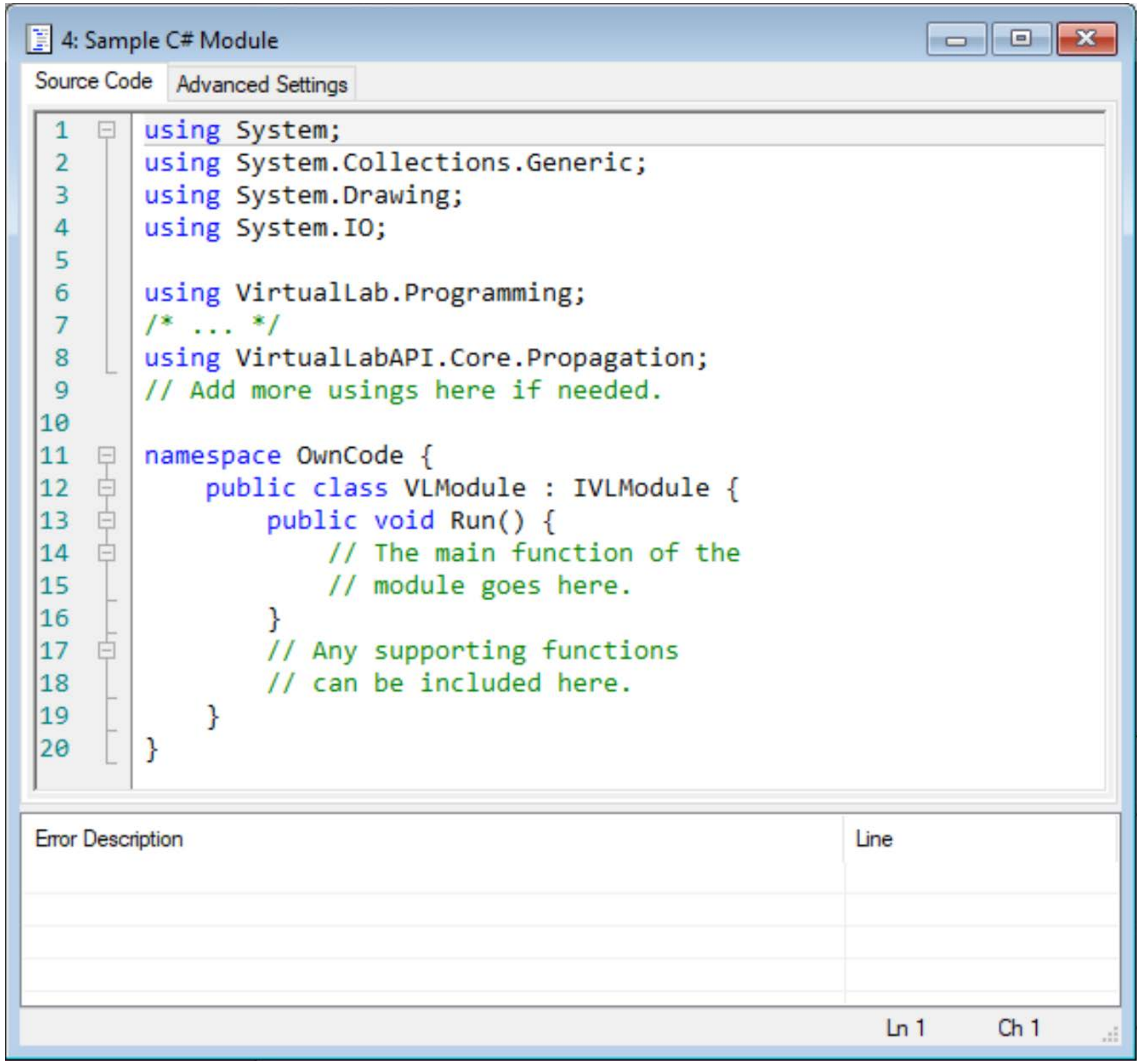
Where to Find the Module



Writing the Code

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

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



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

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

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.

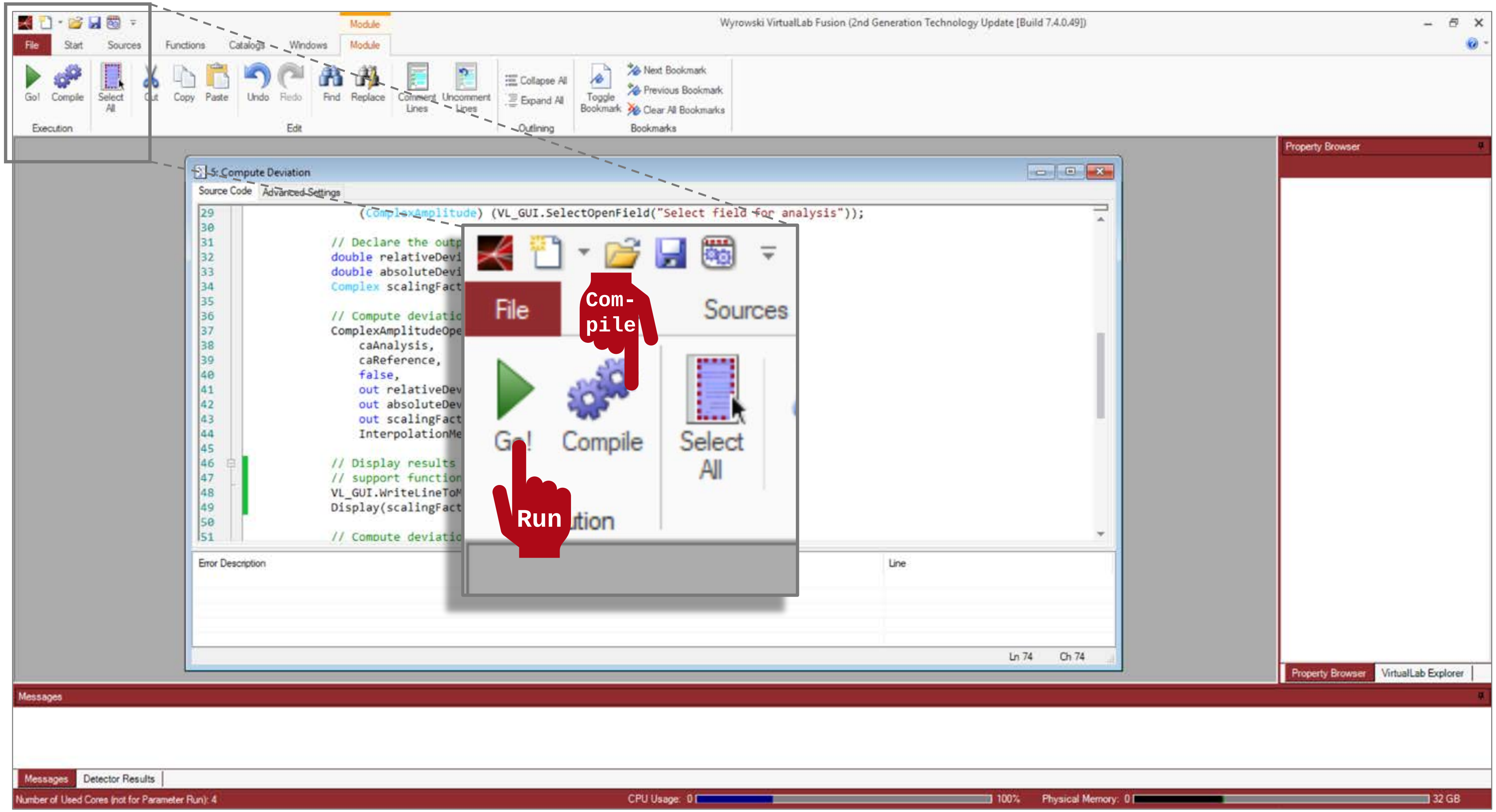
Writing the Code

- `ComplexAmplitude` → Object designed to store a monochromatic, equidistantly sampled complex amplitude (transversal distribution of field at a plane). It stores the `ComplexField` for Ex and Ey, 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 Ex and Ey). 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 `ComplexAmpltiude`. For instance, a polychromatic field, which will contain one `ComplexAmpltiude` 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



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

Standard Deviation

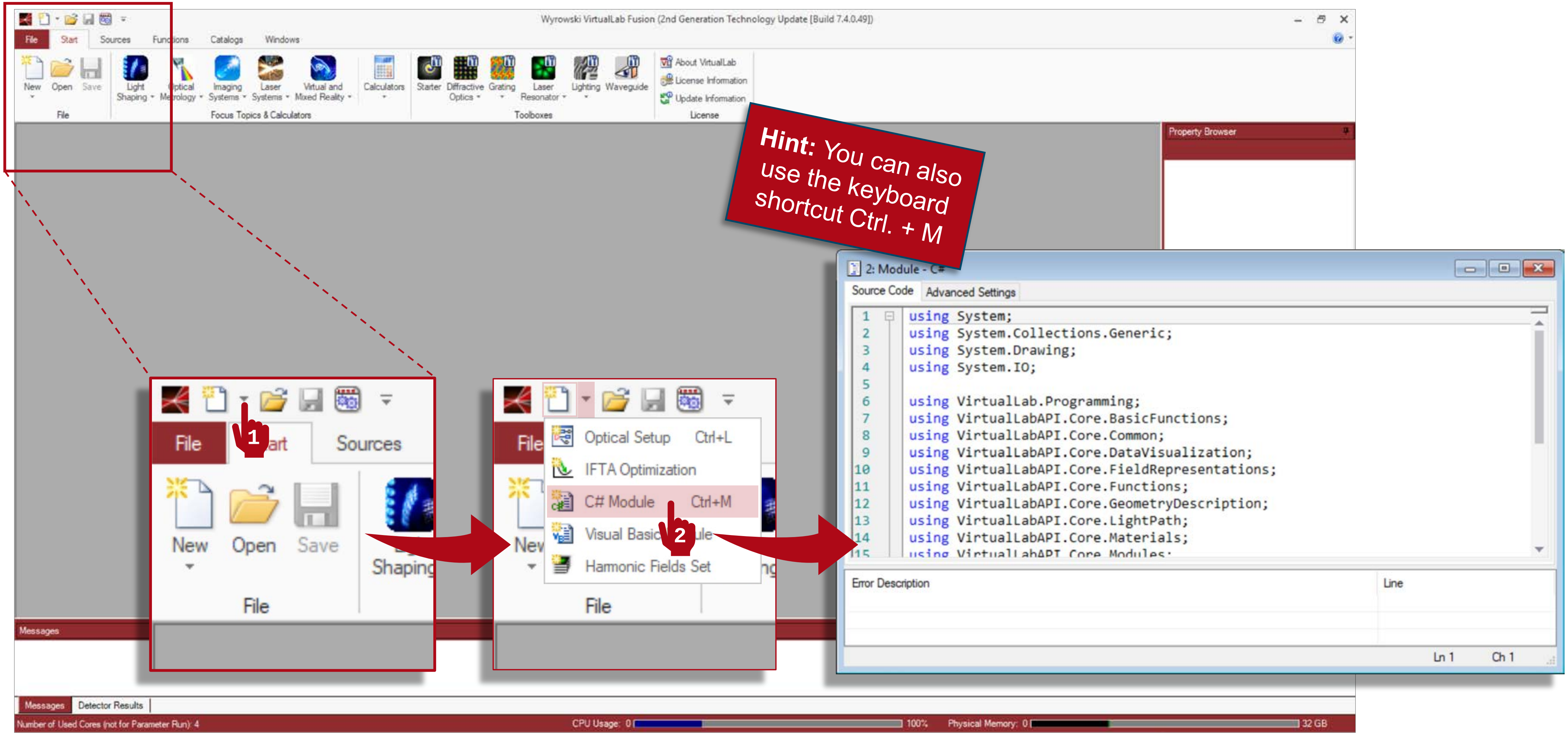
Given two sampled, complex functions *f* and *g*, defined on the *x, y* plane, the relative standard deviation of *g* with respect to *f* is defined as

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

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



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)

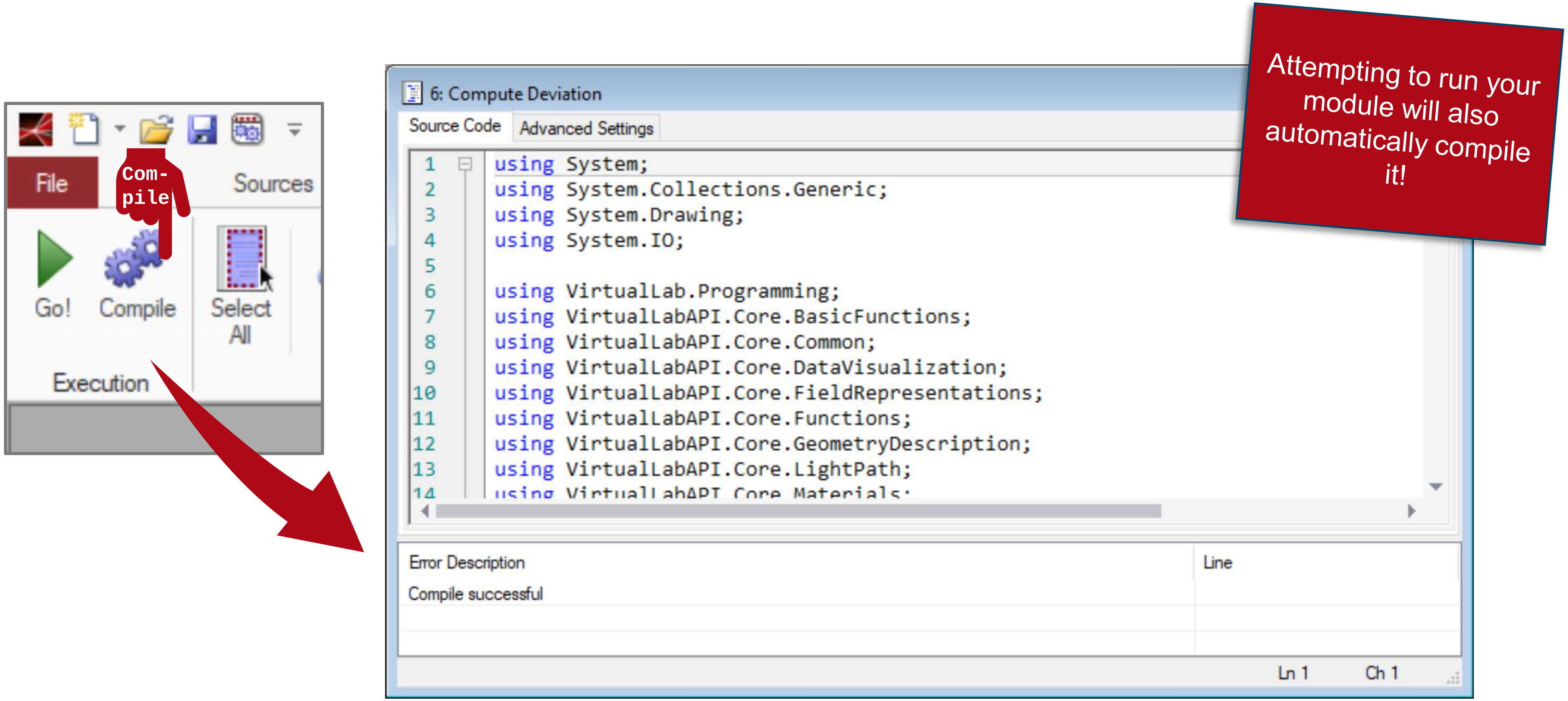
```
            // Display results in Messages window using custom
            // support function (see below):
            VL_GUI.WriteLineToMessagesTab("--Calculation WITHOUT scaling--");
            Display(scalingFactor, relativeDeviation, absoluteDeviation);

            // 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!

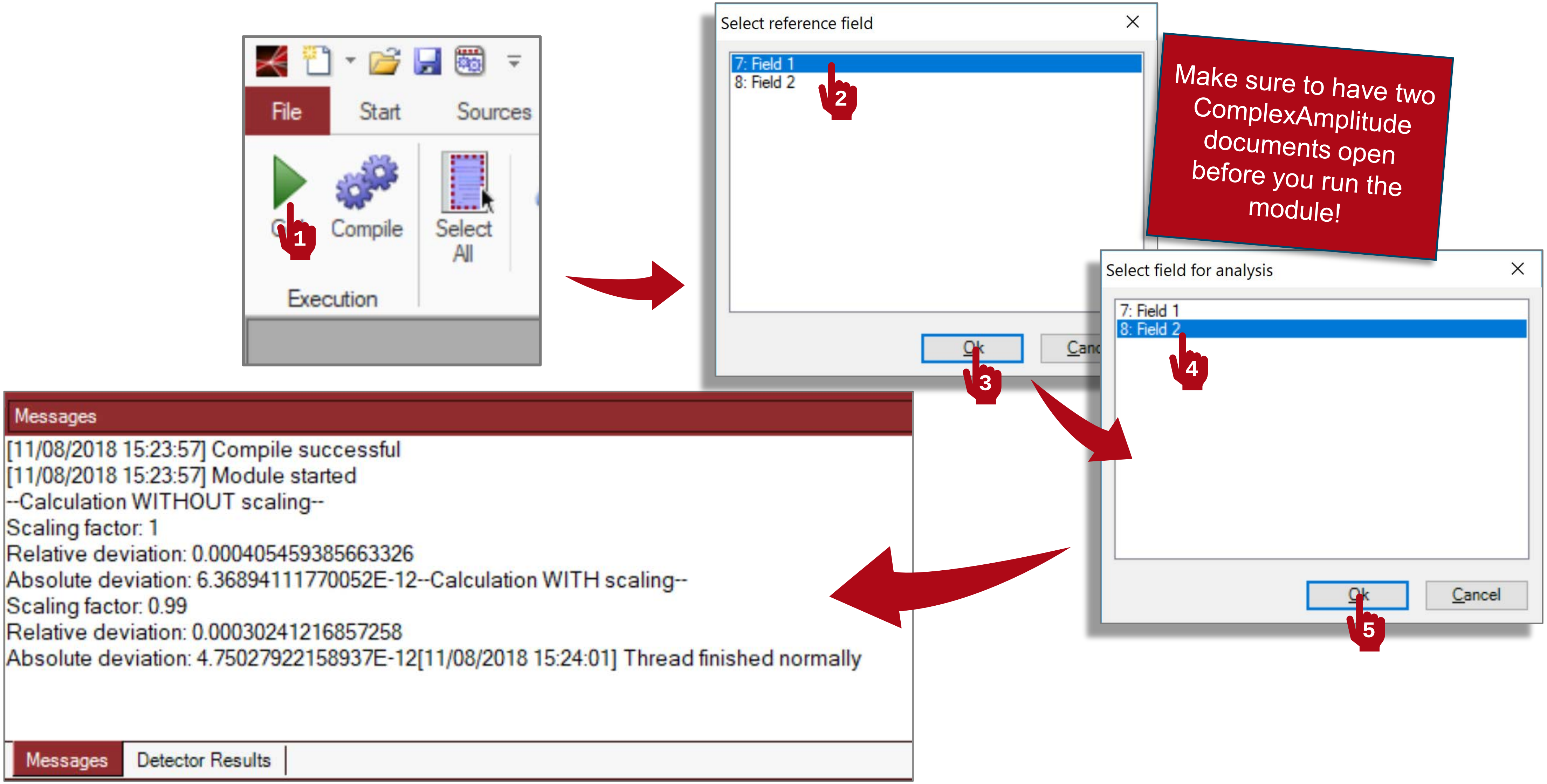
C# Module (Support Functions)

```
        // 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



Compile & Run Your Module



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	- Programming a Module That Smooths the Edge of a Structure - Programming a Module That Computes the Standard Deviation between Two Harmonic Fields