


# Source Code Editor

# Abstract

```
double height = 0.0;
/*****
***** INSERT YOUR CODE HERE *****/
/*****/

/* begin of sample code (can be removed)
height = Math.Sin(x / 5e-3) * y / 10;
end of sample code */

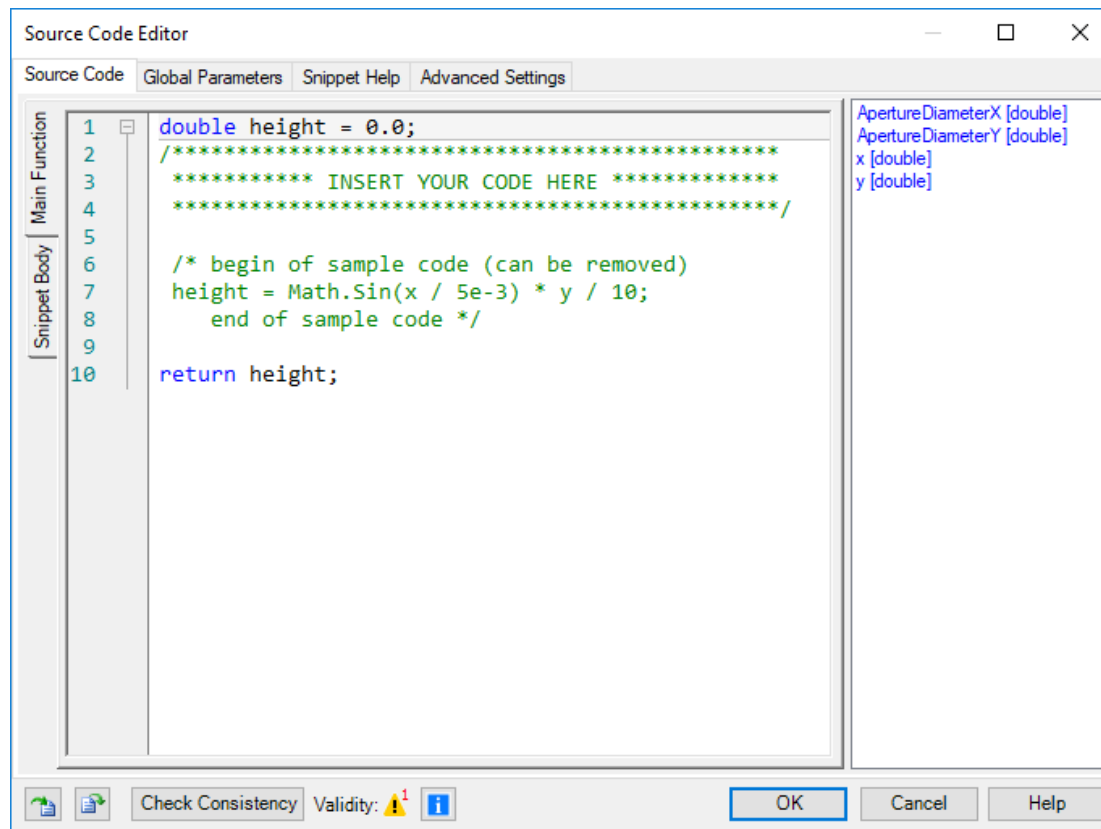
return height;
```



For the optical elements, which cannot be found in the catalogs of VirtualLab Fusion, users can create them by using programmable objects, e.g., programmable source, interface, medium, and detector. Also a programmable component is available, which allows users to develop own algorithm, with the access of all optical elements, as well as the implemented functions of VirtualLab. The programming language is C#. The source code editor is the most important structure of all programmable objects. This use case introduces the general structure of the source code editor.

# This Use Case Shows...

- the general structure of the *Source Code Editor* of programmable objects.



# Tabs: Source Code

- *Source Code* tab :
  - left panel: source code in C# syntax
    - *Main Function*: snippet function to calculate the values this programmable object need to return. For example, in case of programmable interface, the snippet function calculate and return the height profile of this interface  $h(x, y)$ .
    - *Snippet Body*: specify additional methods, properties or variables, which were initialized once-only by calling the snippet.
  - right panel: global parameters, which can be used within the snippet.

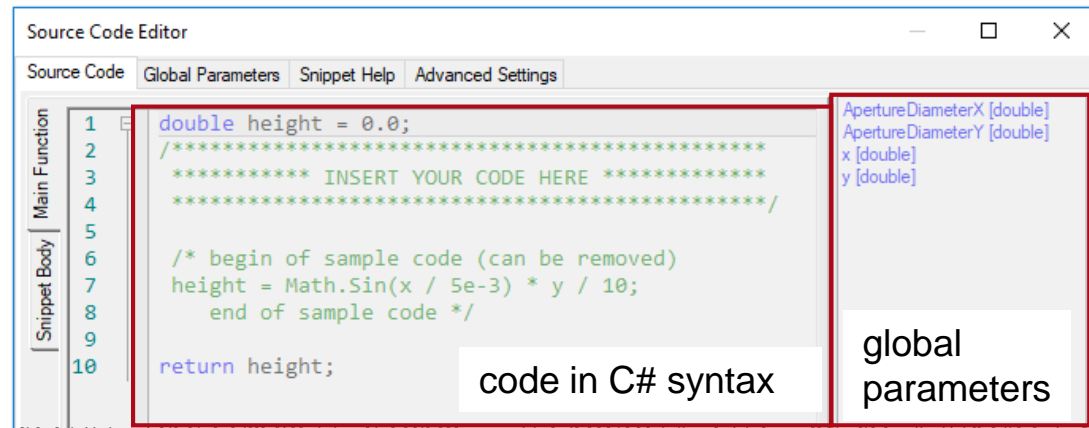





Fig: source code editor of Programmable Interface

# Tabs: Source Code

- *Source Code* tab :
  - left panel: source code in C# syntax
  - right panel: global parameters, which can be used in snippet.
  - bottom:
    - *Import Snippet*  : import a snippet from a .snp-file.
    - *Export Snippet*  : export a snippet into a .snp-file. This file can be used to import the snippet to other programmable objects.
    - *Check Consistency*: check if the snippet is in correct C# syntax. If there is an inconsistency, users will find details by clicking  .

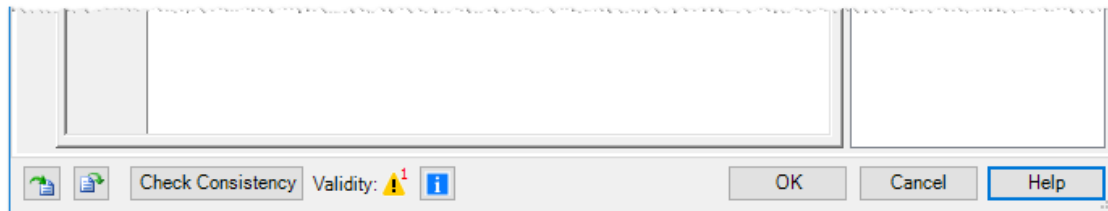
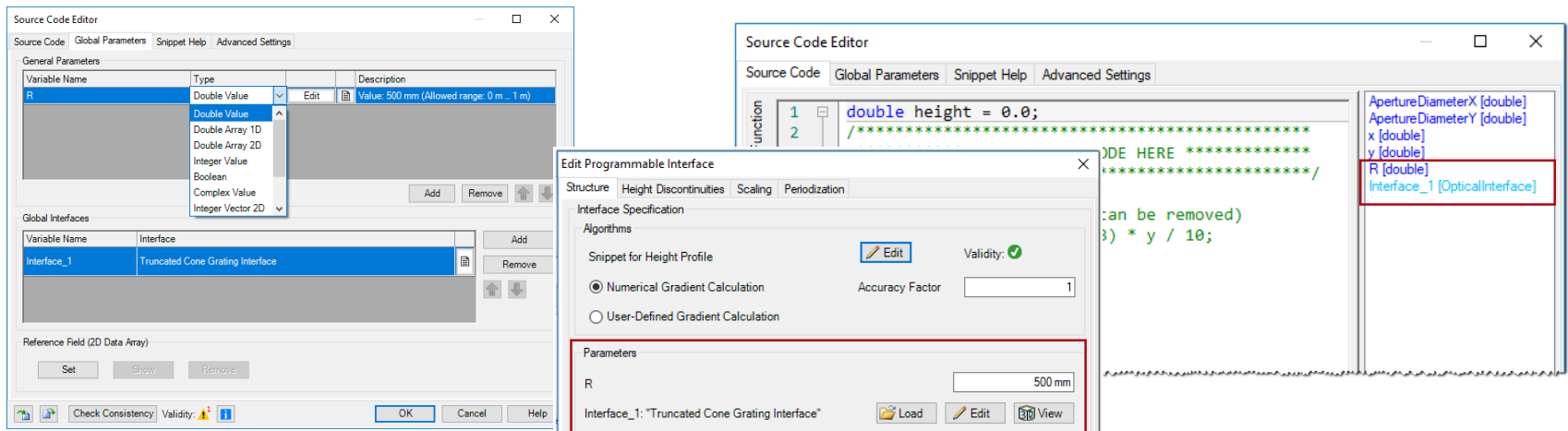


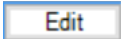
Fig: source code editor of Programmable Interface

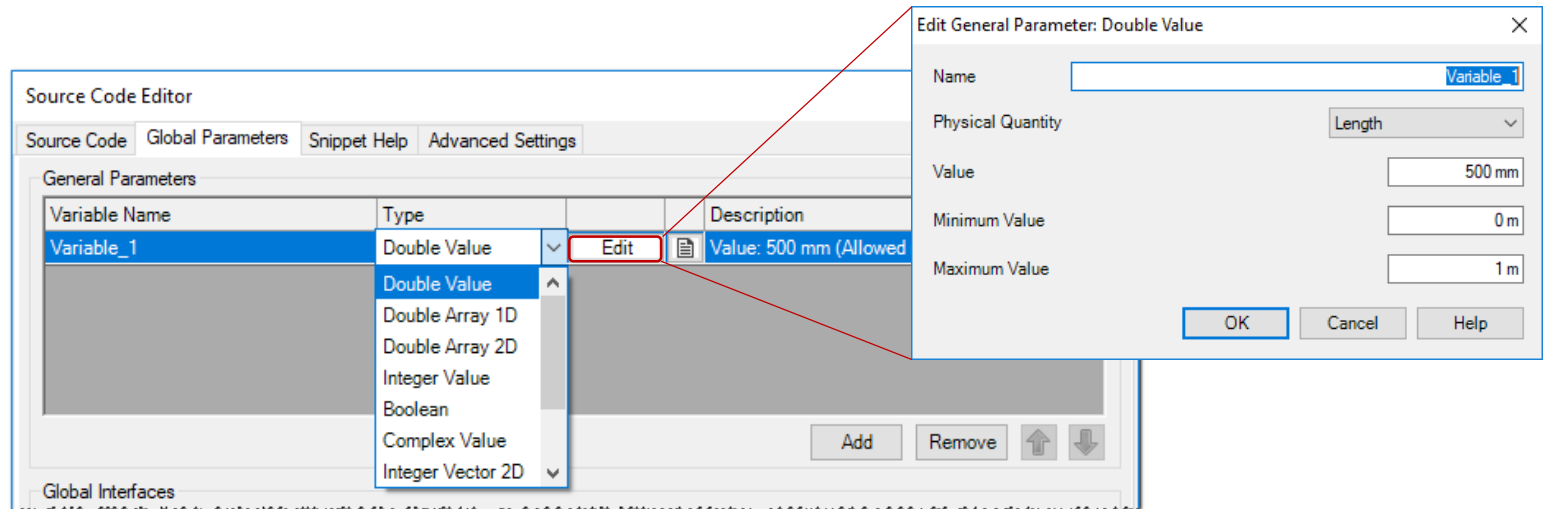
# Tabs: Global Parameters

- *Global Parameters* tab allows users to define different types of global parameters, even predefined interfaces, materials and much more.
  - Global parameters can be manually defined within the programmable object, outside the Source Code Editor.
  - Global parameters can be used in Parameter Run and Parametric Optimization.


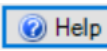


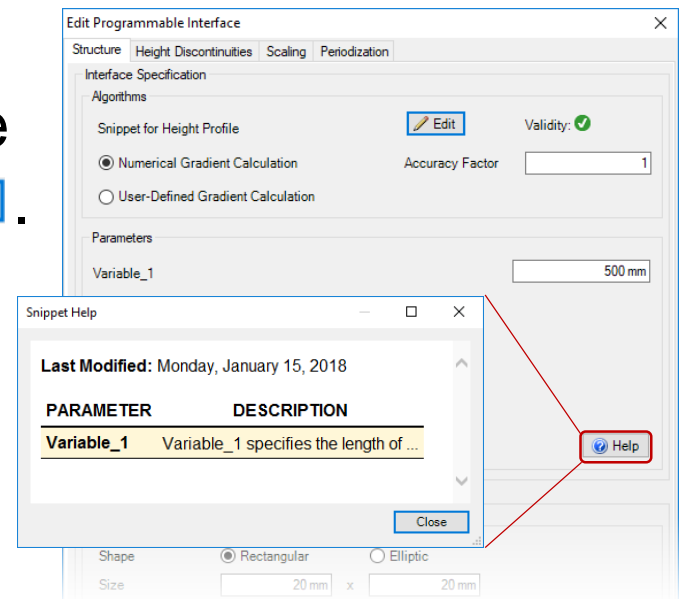
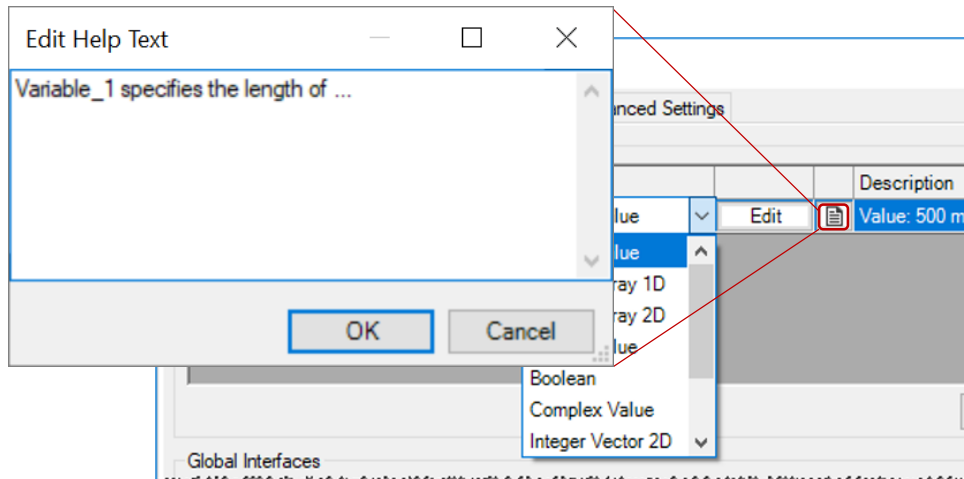
# Global Parameters: General Parameters

- *General Parameters* support several data types like Double, Integer, Boolean, Complex, Double Vector 2D and much more.
- Users are able to define additional properties for the general parameters like physical unit, minimum and maximum value and the valid range by clicking .



# Global Parameters: General Parameters

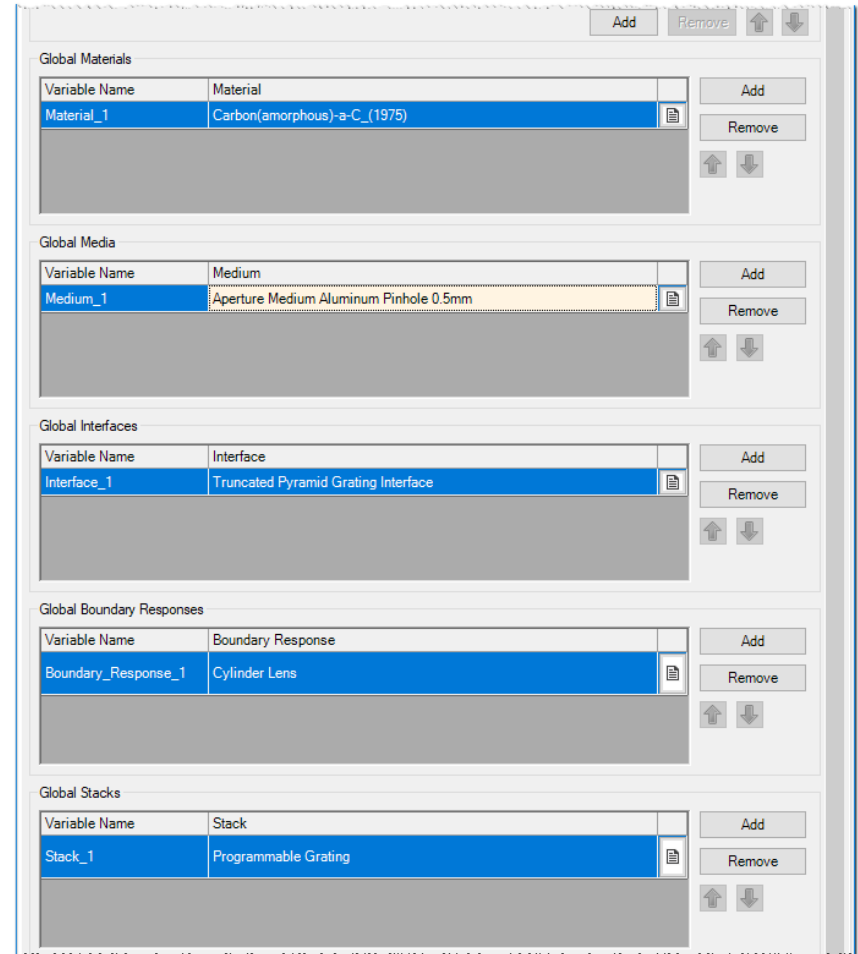
- It is highly recommended to specify a help text for each General Parameter by clicking on  .
- This help text is shown outside the source code editor by clicking  .





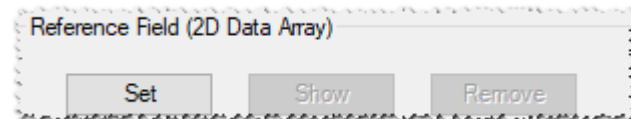
# Global Parameters: System Building Blocks

- Any predefined system building blocks in catalogs can be specified as global parameters.
  - materials
  - media
  - interfaces
  - boundary responses, i.e. transmission functions.
  - stacks



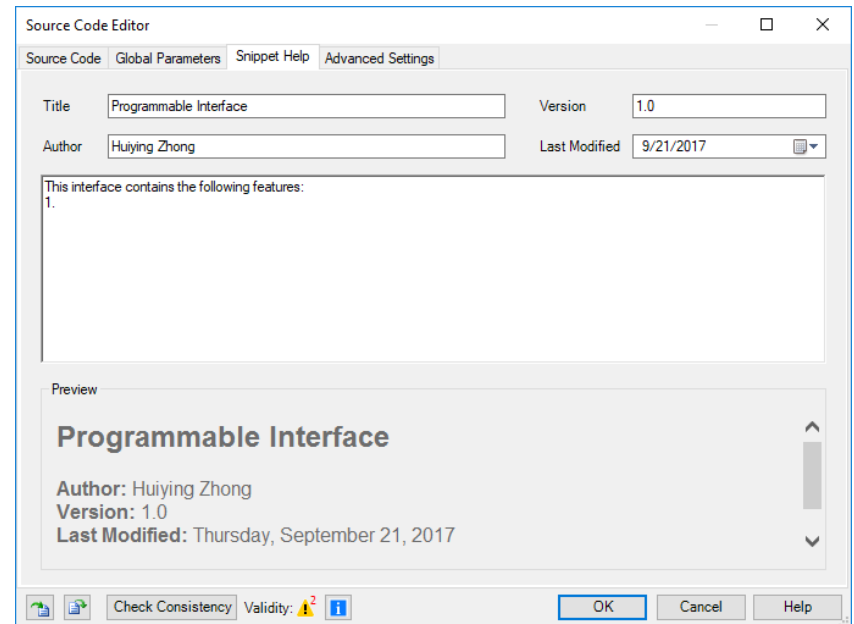
# Global Parameters: Reference Field

- Other harmonic field can be used as *Reference Field*, as global parameter.
- It is super useful when users need information of both fields. E.g. In programmable detector, the reference field is defined as the desired field pattern. Values of merit functions can be calculated by comparing the detected field and *Reference Field*.



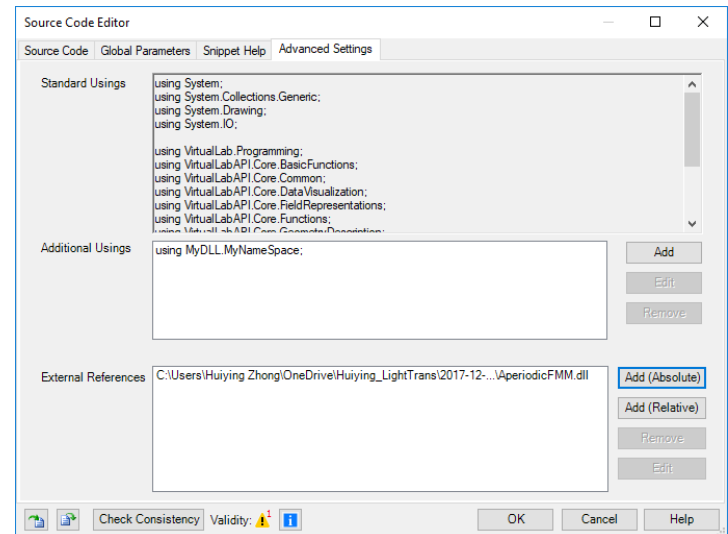
# Tab: Snippet Help

- *Snippets Help* tab helps user to record information about the programmable object and specify different parameters.
- Please read the use case *Customizable Help for Programmable Element* for more details.



# Tab: Advanced Settings

- *Advanced Settings* tab allows you to define additional DLLs and namespaces, which can be used in the snippet.
  - *Standard Usings*: listing standard namespaces, which are used in the snippet
  - *Additional Usings*: user defined additional namespaces of the VirtualLabAPI DLL and the VirtualLab.Programming DLL
  - *External Reference*: importing additional DLLs



# Document Information

---

title	Source Code Editor
version	1.0
VL version used for simulations	7.0.3.4
category	Feature Use Case

---