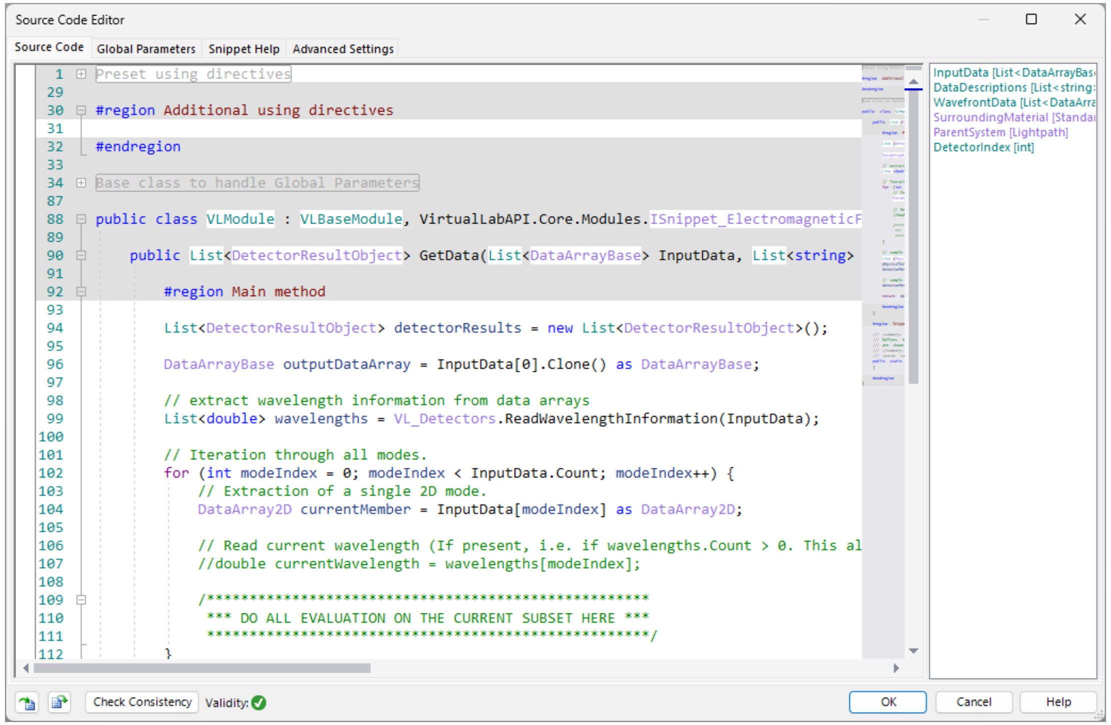
Programming Detector Add-ons in VirtualLab Fusion

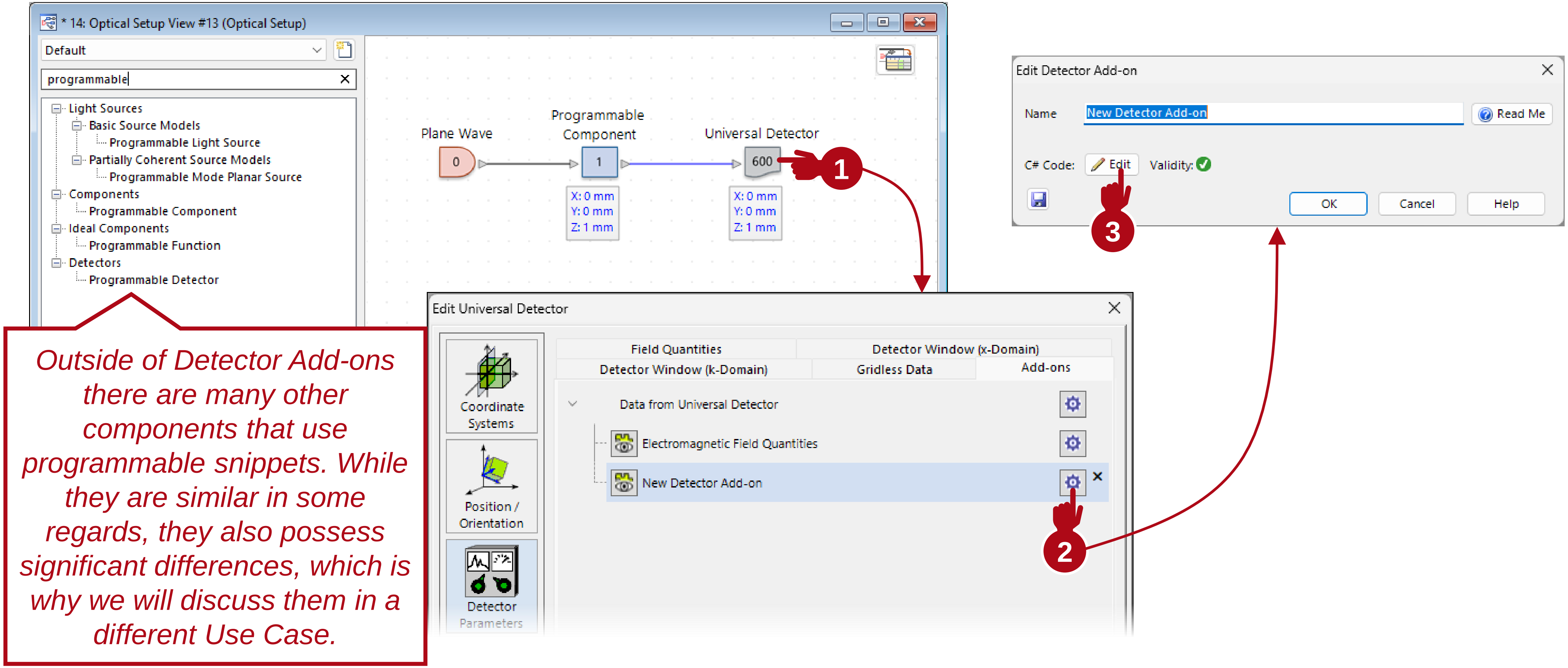
Abstract



```
Preset using directives

#region Additional using directives

#endregion

Base class to handle Global Parameters

public class VLModule : VLBaseModule, VirtualLabAPI.Core.Modules.ISnippet_ElectromagneticF

    public List<DetectorResultObject> GetData(List<DataArrayBase> InputData, List<string>

        #region Main method

        List<DetectorResultObject> detectorResults = new List<DetectorResultObject>();

        DataArrayBase outputDataArray = InputData[0].Clone() as DataArrayBase;

        // extract wavelength information from data arrays
        List<double> wavelengths = VL_Detectors.ReadWavelengthInformation(InputData);

        // Iteration through all modes.
        for (int modeIndex = 0; modeIndex < InputData.Count; modeIndex++) {
            // Extraction of a single 2D mode.
            DataArray2D currentMember = InputData[modeIndex] as DataArray2D;

            // Read current wavelength (If present, i.e. if wavelengths.Count > 0. This al
            //double currentWavelength = wavelengths[modeIndex];

            /*****************************************************
             *** DO ALL EVALUATION ON THE CURRENT SUBSET HERE ***
             *****************************************************/
        }
```

VirtualLab Fusions programmable tools offer maximum flexibility in the definition of physical behavior. Especially the customizable detector add-ons allow for a free definition of the detected physical quantity calculated from the electromagnetic field. In this Use Case we want to shortly introduce how to work with programmable detector add-ons and give two easy examples as references.

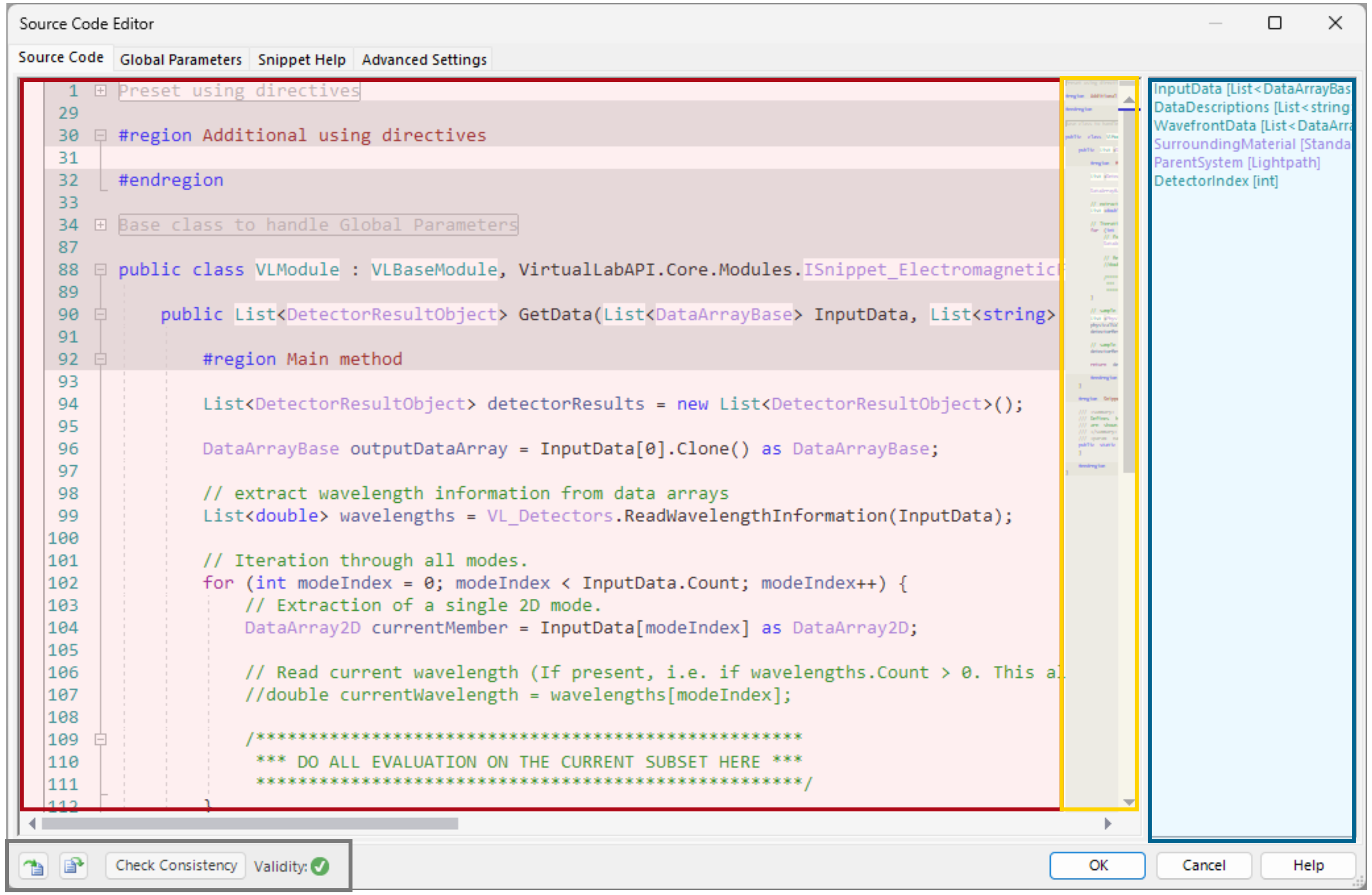
Programmable Snippets for Detector Add-ons



Outside of Detector Add-ons there are many other components that use programmable snippets. While they are similar in some regards, they also possess significant differences, which is why we will discuss them in a different Use Case.

Overview of the Programmable Snippet – Source Code Tab

The actual code is written in the main window on the left side. VirtualLab Fusion uses C# as a programming language. A tutorial on C# by itself is not included in this Use Case.



```
Preset using directives

#region Additional using directives

#endregion

Base class to handle Global Parameters

public class VLModule : VLBaseModule, VirtualLabAPI.Core.Modules.ISnippet_Electromagnetic

    public List<DetectorResultObject> GetData(List<DataArrayBase> InputData, List<string>

        #region Main method

        List<DetectorResultObject> detectorResults = new List<DetectorResultObject>();

        DataArrayBase outputDataArray = InputData[0].Clone() as DataArrayBase;

        // extract wavelength information from data arrays
        List<double> wavelengths = VL_Detectors.ReadWavelengthInformation(InputData);

        // Iteration through all modes.
        for (int modeIndex = 0; modeIndex < InputData.Count; modeIndex++) {
            // Extraction of a single 2D mode.
            DataArray2D currentMember = InputData[modeIndex] as DataArray2D;

            // Read current wavelength (If present, i.e. if wavelengths.Count > 0. This a
            //double currentWavelength = wavelengths[modeIndex];

            /*****************************************************
             *** DO ALL EVALUATION ON THE CURRENT SUBSET HERE ***
             *****************************************************/
        }
```

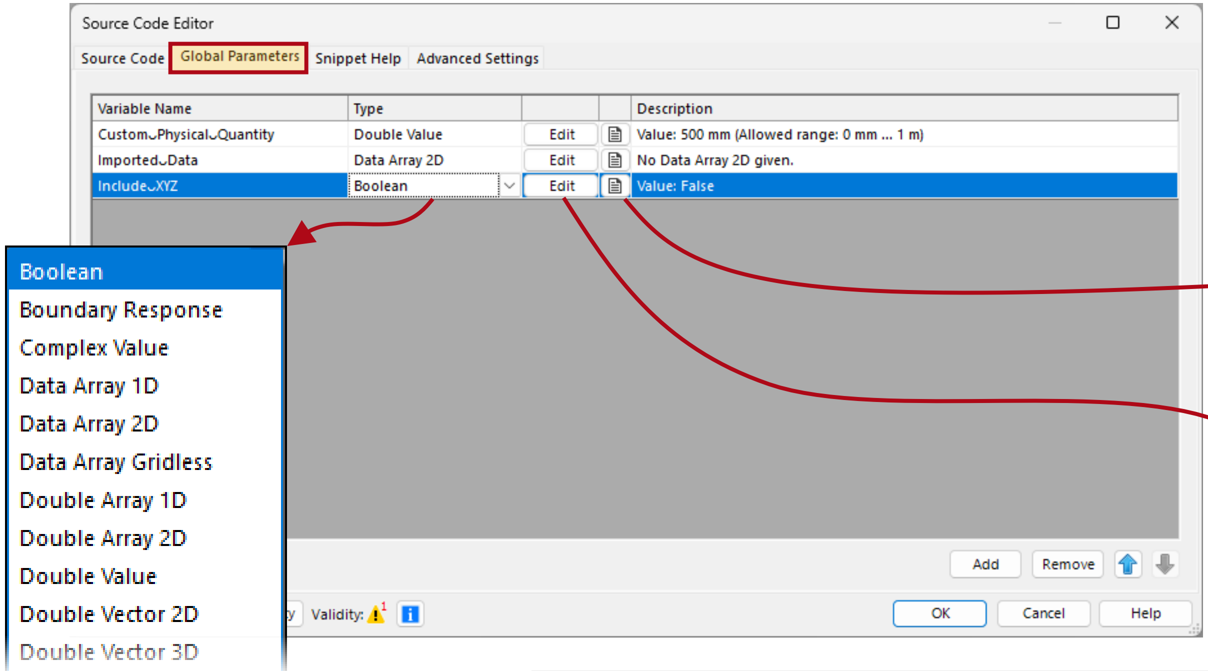
InputData [List<DataArrayBas
DataDescriptions [List<string
WavefrontData [List<DataArra
SurroundingMaterial [Standa
ParentSystem [Lightpath]
DetectorIndex [int]

On the right side all global parameters are listed with the respective classes. This includes parameters defined by the user (see following pages) as well as pre-defined parameters depending on where the snippet is found. For Detector Add-ons the input field (with wavefront) and the surrounding material are e.g. pre-defined global parameters.

At the bottom of the window, tools for importing, exporting and checking the validity of the code can be found.

The scroll bar includes a preview of the code for an easier navigation for long snippets.

Inclusion of Custom Parameters



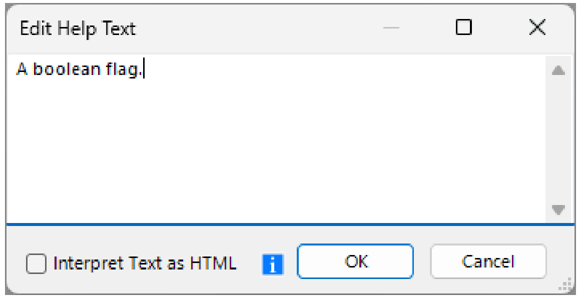
Source Code Editor

Source Code | Global Parameters | Snippet Help | Advanced Settings

Variable Name	Type			Description
Custom␣Physical␣Quantity	Double Value	Edit	🗎	Value: 500 mm (Allowed range: 0 mm ... 1 m)
Imported␣Data	Data Array 2D	Edit	🗎	No Data Array 2D given.
Include␣XYZ	Boolean	Edit	🗎	Value: False

- Boolean
- Boundary Response
- Complex Value
- Data Array 1D
- Data Array 2D
- Data Array Gridless
- Double Array 1D
- Double Array 2D
- Double Value
- Double Vector 2D
- Double Vector 3D

VirtualLab Fusion offers a comprehensive selection of various parameter types, from Booleans over complex data values to data arrays and materials.

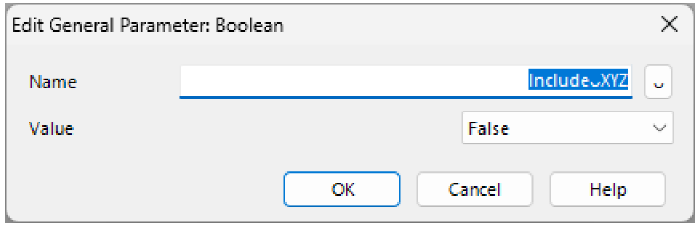


Edit Help Text

A boolean flag.

□ Interpret Text as HTML

Each parameter has a small text box assigned to it, which can be filled by the creator, to remind customers, collogues or oneself of what the parameter shall represent.

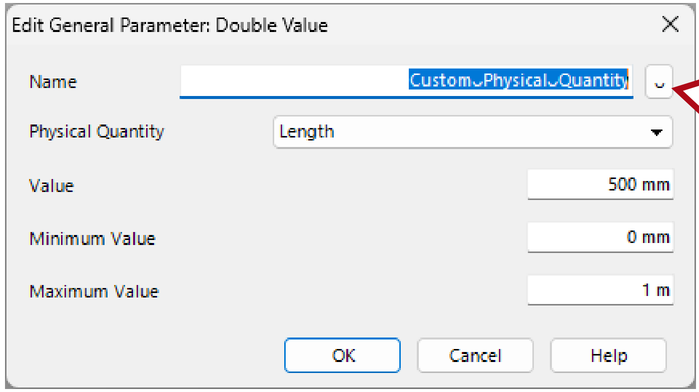


Edit General Parameter: Boolean

Name: Include␣XYZ

Value: False

The edit window will automatically adjust depending on the type of parameter to include e.g., the unit.



Edit General Parameter: Double Value

Name: Custom␣Physical␣Quantity

Physical Quantity: Length

Value: 500 mm

Minimum Value: 0 mm

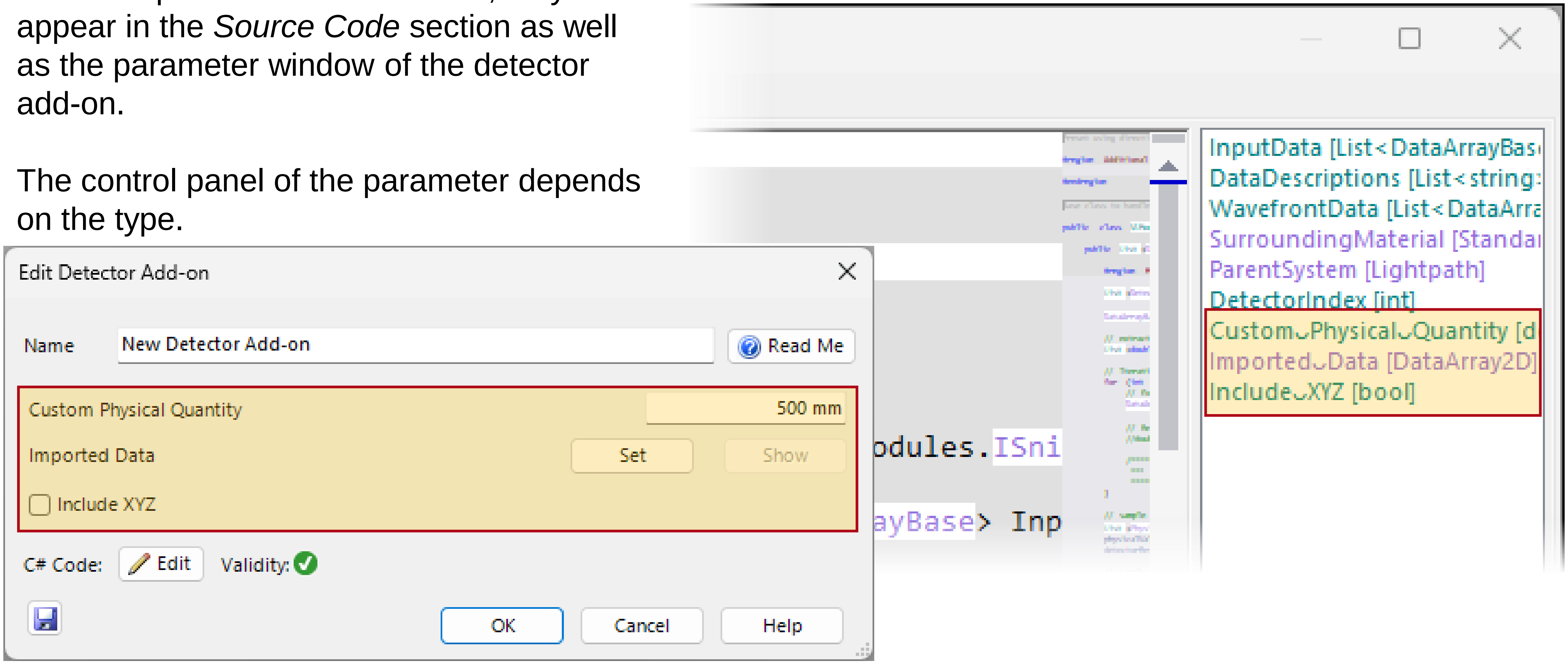
Maximum Value: 1 m

Tipp: This button allows for the inclusion of spaces in the parameter title.

Include Custom Parameters

Once the parameters are defined, they will appear in the *Source Code* section as well as the parameter window of the detector add-on.

The control panel of the parameter depends on the type.



```
odules.ISni
ayBase> Inp
```

InputData [List<DataArrayBas
DataDescriptions [List<string
WavefrontData [List<DataArra
SurroundingMaterial [Standa
ParentSystem [Lightpath]
DetectorIndex [int]
Custom◡Physical◡Quantity [d
Imported◡Data [DataArray2D]
Include◡XYZ [bool]

Edit Detector Add-on

Name: New Detector Add-on

Custom Physical Quantity: 500 mm

Imported Data

□ Include XYZ

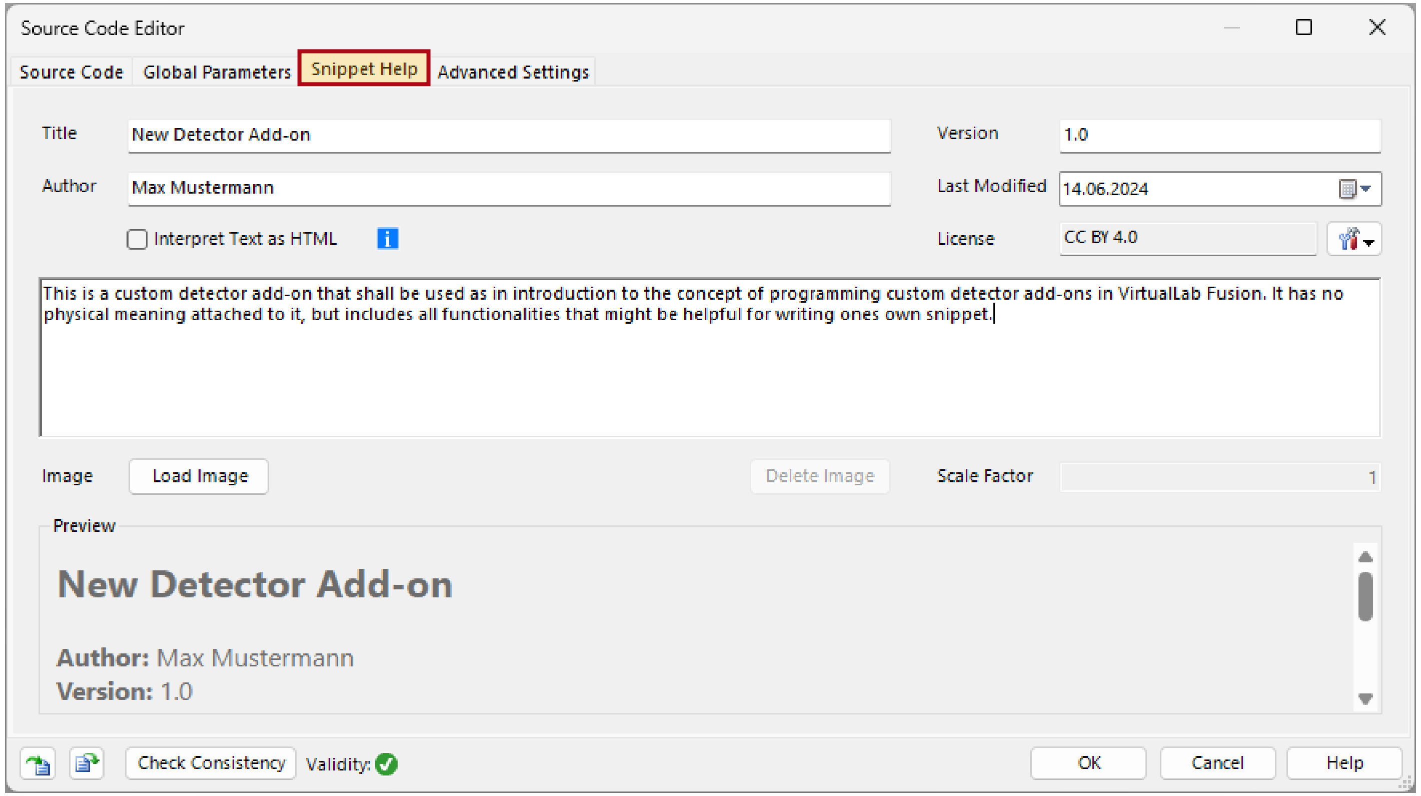
C# Code:

Validity: ✓

Snippet Help

Snippets created by LightTrans International GmbH normally come equipped with a Read Me , that contains useful information, such as a short description of what the snippet does.

For custom snippets, in the Snippet Help section such a document can also be generated by the user. Once this page is filled with content, a Read Me button will automatically appear in the detector add-on edit window.



Title: New Detector Add-on

Version: 1.0

Author: Max Mustermann

Last Modified: 14.06.2024

□ Interpret Text as HTML

License: CC BY 4.0

This is a custom detector add-on that shall be used as in introduction to the concept of programming custom detector add-ons in VirtualLab Fusion. It has no physical meaning attached to it, but includes all functionalities that might be helpful for writing ones own snippet.

Image

Scale Factor: 1

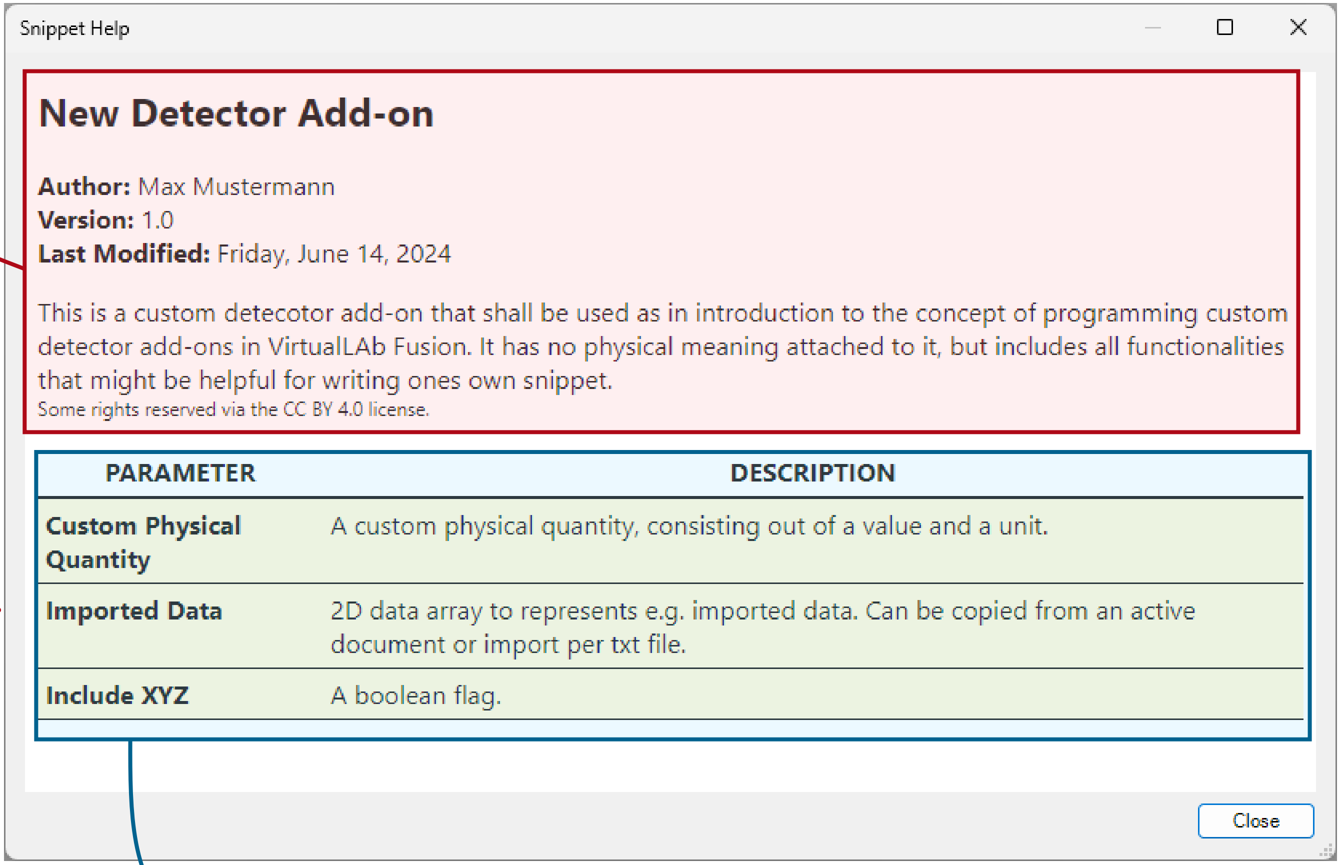
New Detector Add-on

Author: Max Mustermann
Version: 1.0

Validity: ✓

Snippet Help

information from the *Snippet Help* section



New Detector Add-on

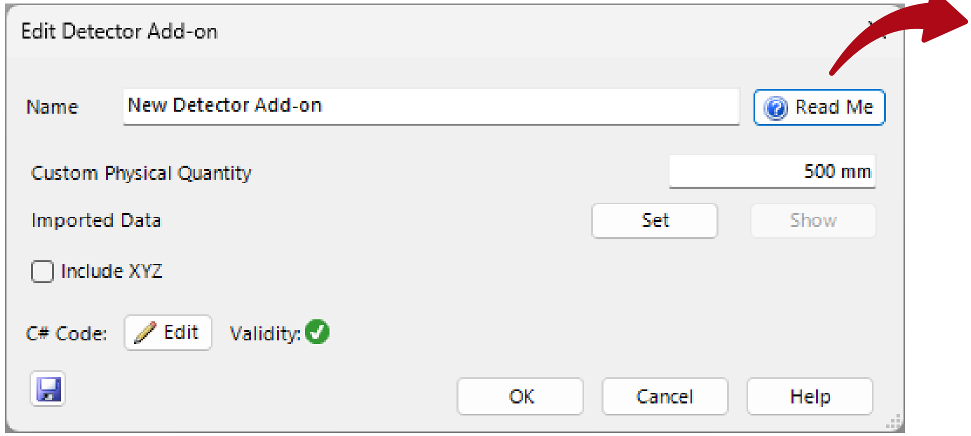
Author: Max Mustermann
Version: 1.0
Last Modified: Friday, June 14, 2024

This is a custom detecotor add-on that shall be used as in introduction to the concept of programming custom detector add-ons in VirtualLAb Fusion. It has no physical meaning attached to it, but includes all functionalities that might be helpful for writing ones own snippet.

Some rights reserved via the CC BY 4.0 license.

PARAMETER	DESCRIPTION
Custom Physical Quantity	A custom physical quantity, consisting out of a value and a unit.
Imported Data	2D data array to represents e.g. imported data. Can be copied from an active document or import per txt file.
Include XYZ	A boolean flag.

information from *Help Text* (🗎) in the *Global Parameters* section



Name: New Detector Add-on

Custom Physical Quantity: 500 mm

Imported Data

☐ Include XYZ

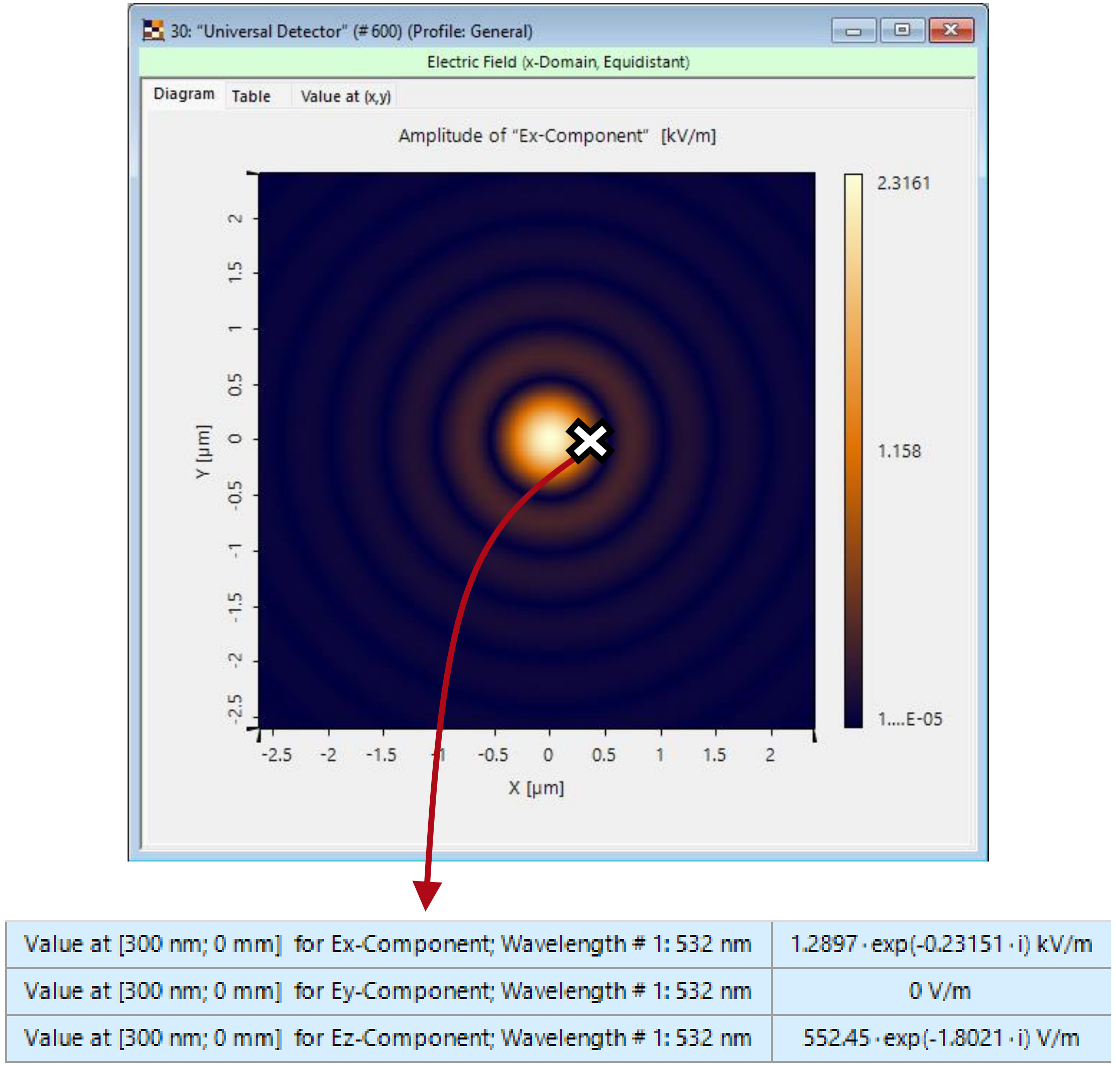
C# Code: Validity: ✓

Example 1: Extract Field Value At Point

Task Description

As an easy example, we want to demonstrate an add-on that detects the amplitude and phase for all field components at a certain point. For the sake of simplicity, the add-on will be restricted to only work for electromagnetic fields that of a 2D gridded input.

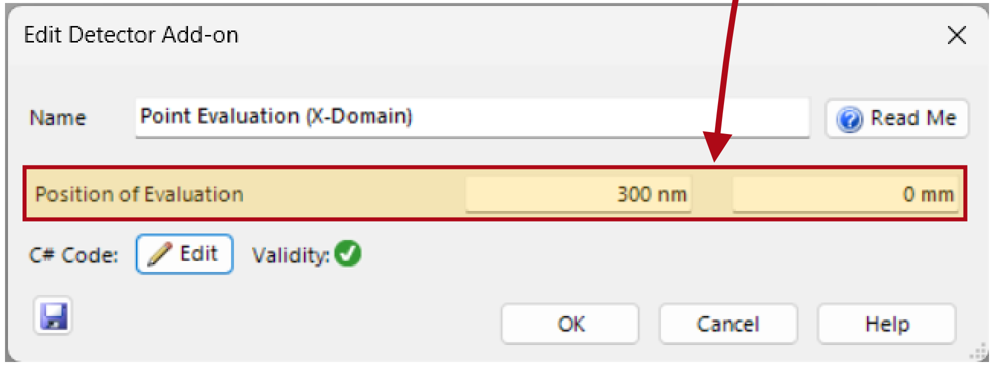
For a more sophisticated approach of this task, i.e. the generalization for also 1D gridded and gridless data and an automatic detection of the unit of the input field, please see the documentation of our *Point Evaluation* add-on.


30: "Universal Detector" (# 600) (Profile: General)
Electric Field (x-Domain, Equidistant)
Diagram
Table
Value at (x,y)
Amplitude of "Ex-Component" [kV/m]
Y [µm]
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
X [µm]
2.3161
1.158
1....E-05

Value at [300 nm; 0 mm] for Ex-Component; Wavelength # 1: 532 nm	1.2897 · exp(-0.23151 · i) kV/m
Value at [300 nm; 0 mm] for Ey-Component; Wavelength # 1: 532 nm	0 V/m
Value at [300 nm; 0 mm] for Ez-Component; Wavelength # 1: 532 nm	552.45 · exp(-1.8021 · i) V/m

Parameters

For the task we define a *Double Vector 2D* parameter with *Length* as the *Physical Quantity*, as this allows the definition of the evaluation point in units (such as nm)



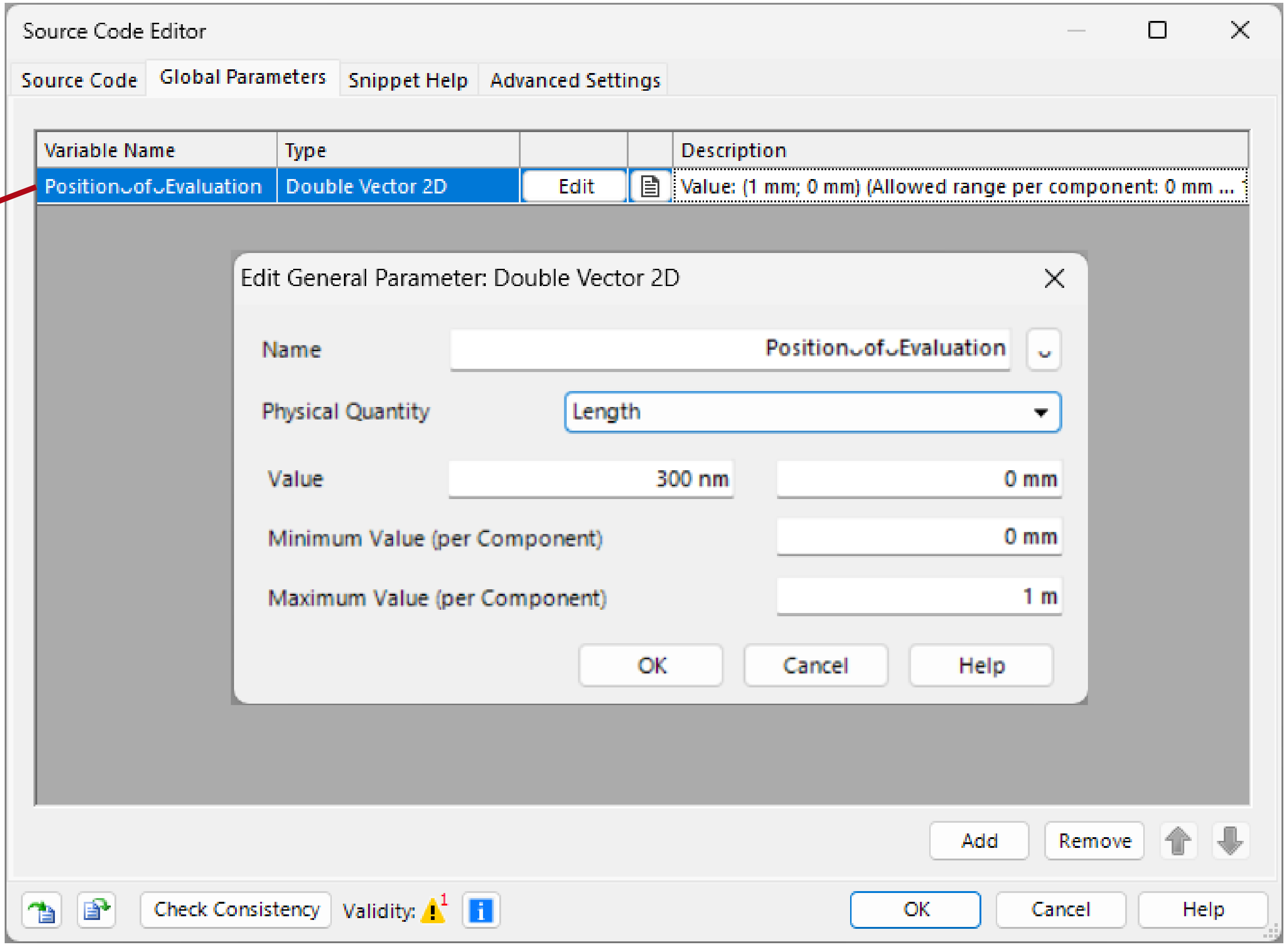
Edit Detector Add-on

Name: Point Evaluation (X-Domain)

Position of Evaluation	300 nm	0 mm

C# Code: Edit

Validity: ✓



Source Code Editor

Variable Name	Type			Description
Position␣of␣Evaluation	Double Vector 2D	Edit	🗎	Value: (1 mm; 0 mm) (Allowed range per component: 0 mm ...

Edit General Parameter: Double Vector 2D

Name: Position␣of␣Evaluation

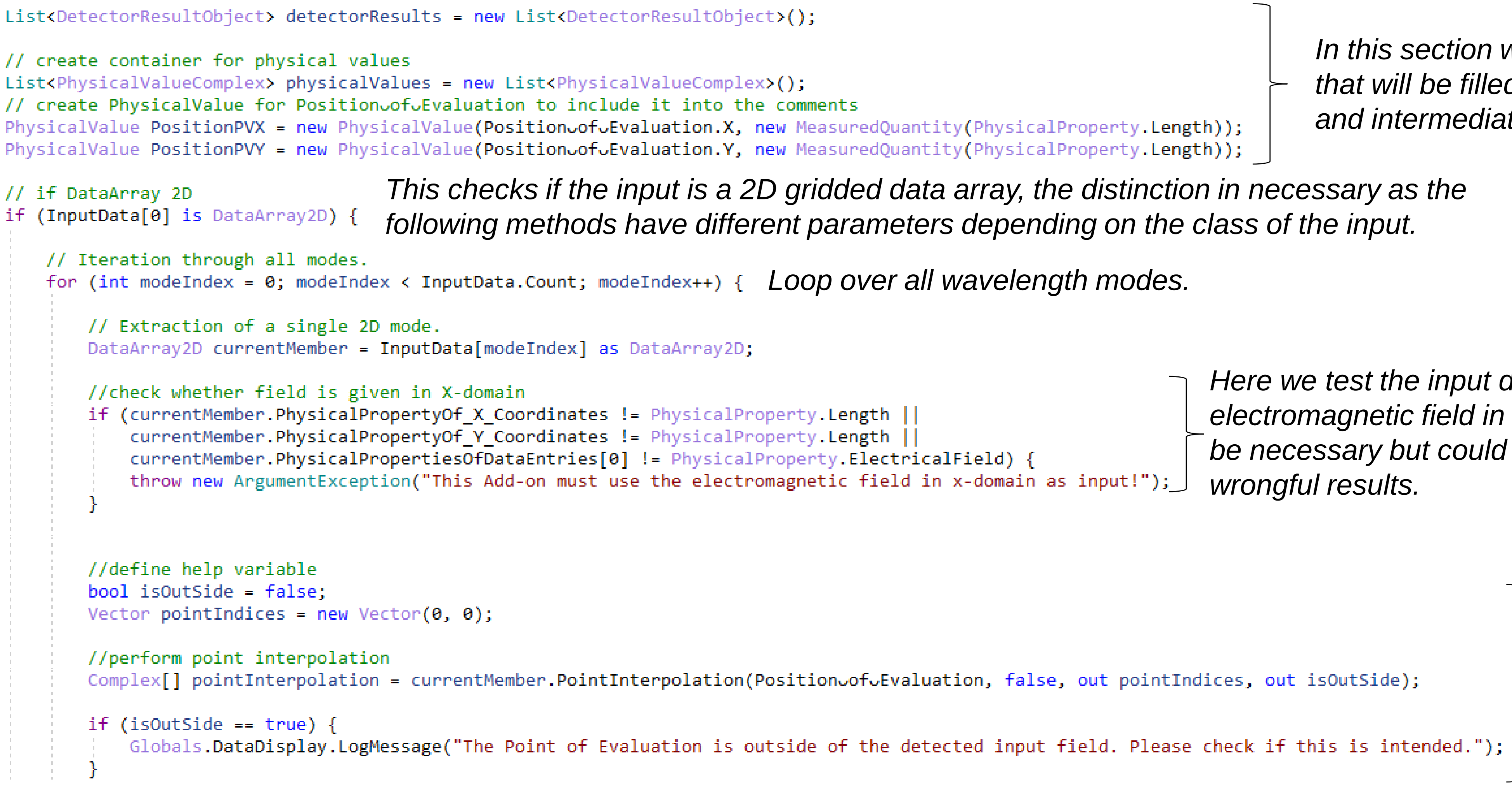
Physical Quantity: Length

Value: 300 nm | 0 mm

Minimum Value (per Component): 0 mm

Maximum Value (per Component): 1 m

Source Code



```
List<DetectorResultObject> detectorResults = new List<DetectorResultObject>();

// create container for physical values
List<PhysicalValueComplex> physicalValues = new List<PhysicalValueComplex>();
// create PhysicalValue for Position‿of‿Evaluation to include it into the comments
PhysicalValue PositionPVX = new PhysicalValue(Position‿of‿Evaluation.X, new MeasuredQuantity(PhysicalProperty.Length));
PhysicalValue PositionPVY = new PhysicalValue(Position‿of‿Evaluation.Y, new MeasuredQuantity(PhysicalProperty.Length));

// if DataArray 2D
if (InputData[0] is DataArray2D) {

    // Iteration through all modes.
    for (int modeIndex = 0; modeIndex < InputData.Count; modeIndex++) {

        // Extraction of a single 2D mode.
        DataArray2D currentMember = InputData[modeIndex] as DataArray2D;

        //check whether field is given in X-domain
        if (currentMember.PhysicalPropertyOf_X_Coordinates != PhysicalProperty.Length ||
            currentMember.PhysicalPropertyOf_Y_Coordinates != PhysicalProperty.Length ||
            currentMember.PhysicalPropertiesOfDataEntries[0] != PhysicalProperty.ElectricalField) {
            throw new ArgumentException("This Add-on must use the electromagnetic field in x-domain as input!");
        }


        //define help variable
        bool isOutSide = false;
        Vector pointIndices = new Vector(0, 0);

        //perform point interpolation
        Complex[] pointInterpolation = currentMember.PointInterpolation(Position‿of‿Evaluation, false, out pointIndices, out isOutSide);

        if (isOutSide == true) {
            Globals.DataDisplay.LogMessage("The Point of Evaluation is outside of the detected input field. Please check if this is intended.");
        }
```

In this section we initialize the container that will be filled with the detector output and intermediatory results.

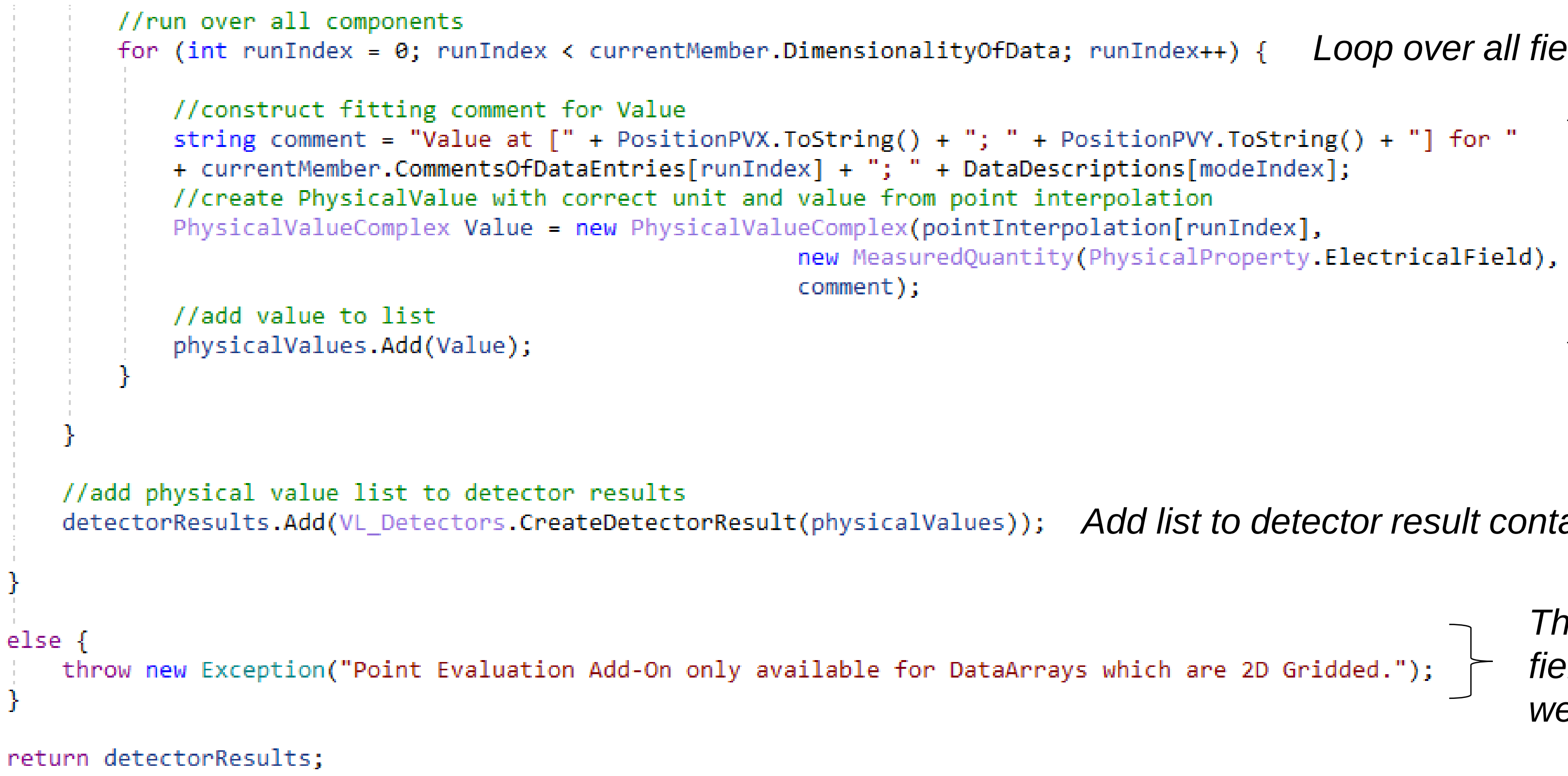
This checks if the input is a 2D gridded data array, the distinction in necessary as the following methods have different parameters depending on the class of the input.

Loop over all wavelength modes.

Here we test the input data if it truly represents an electromagnetic field in x-domain. This may not be necessary but could be helpful to avoid wrongful results.

Here we calculate which data point entry shall be used to extract amplitude and phase on. We also included an optional warning in case the point is outside of the scope of the detected field, as that may make the interpolation unreliable.

Source Code



```
        //run over all components
        for (int runIndex = 0; runIndex < currentMember.DimensionalityOfData; runIndex++) {

            //construct fitting comment for Value
            string comment = "Value at [" + PositionPVX.ToString() + "; " + PositionPVY.ToString() + "] for "
            + currentMember.CommentsOfDataEntries[runIndex] + "; " + DataDescriptions[modeIndex];
            //create PhysicalValue with correct unit and value from point interpolation
            PhysicalValueComplex Value = new PhysicalValueComplex(pointInterpolation[runIndex],
                                                         new MeasuredQuantity(PhysicalProperty.ElectricalField),
                                                         comment);
            //add value to list
            physicalValues.Add(Value);
        }

    }

    //add physical value list to detector results
    detectorResults.Add(VL_Detectors.CreateDetectorResult(physicalValues));

}

else {
    throw new Exception("Point Evaluation Add-On only available for DataArrays which are 2D Gridded.");
}

return detectorResults;
```

Loop over all field components.

Amplitude and Phase are extracted at the calculated data point entry and attached with a fitting unit and comment.
The information is then added to a list as in the case of multiple components and/or wavelength modes we will perform this extraction multiple times.

Add list to detector result container.

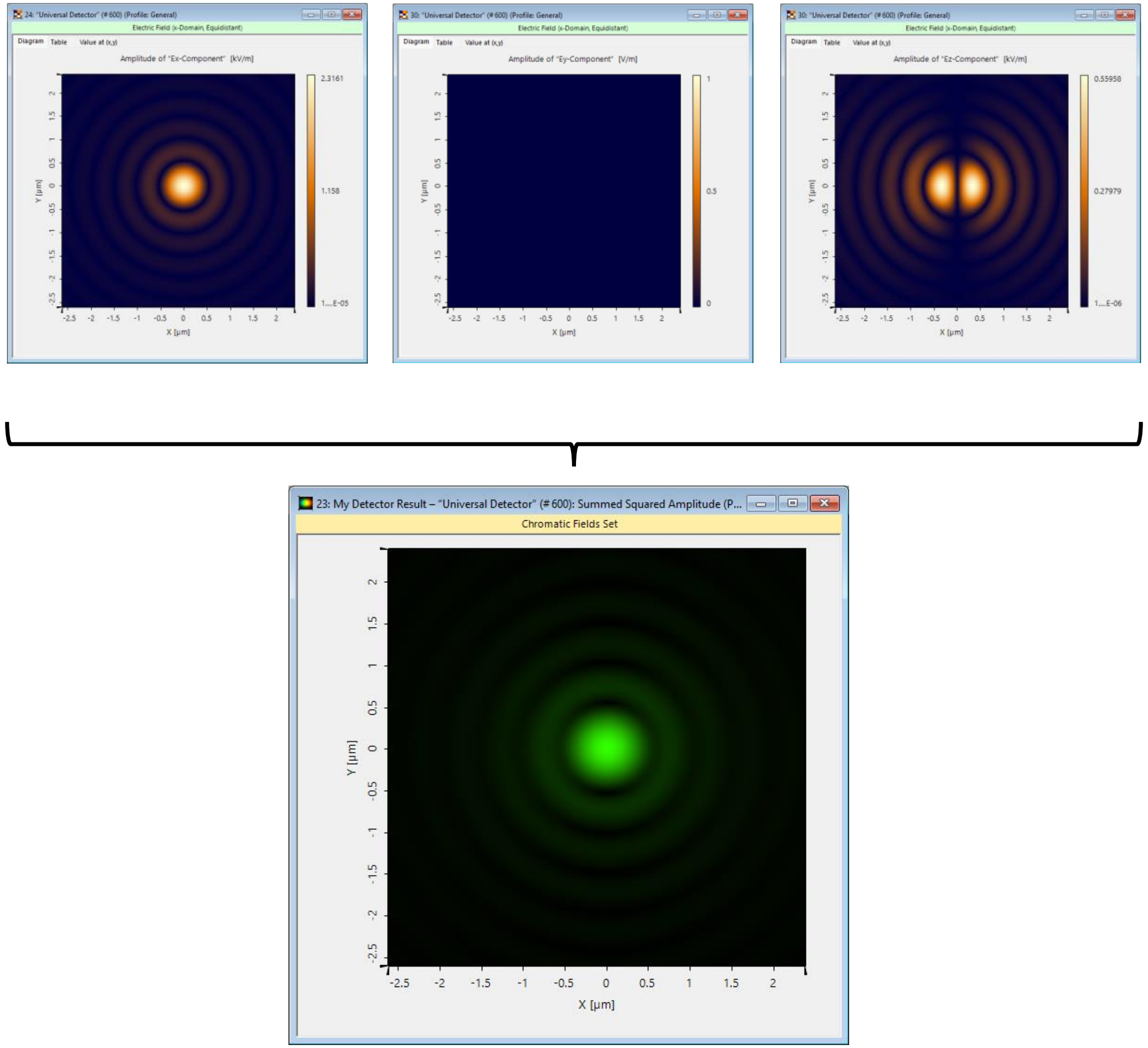
This section determines what happens if the input field is not a 2D gridded data array. In our case we throw an error message.

Example 2: Summed Squared Amplitude

Task Description

For a little more sophisticated add-on, we next want to calculate the summed squared amplitude of an input field. The add-on shall automatically detect if only E-field component are active and shall have a parameter to determine the interpolation method.

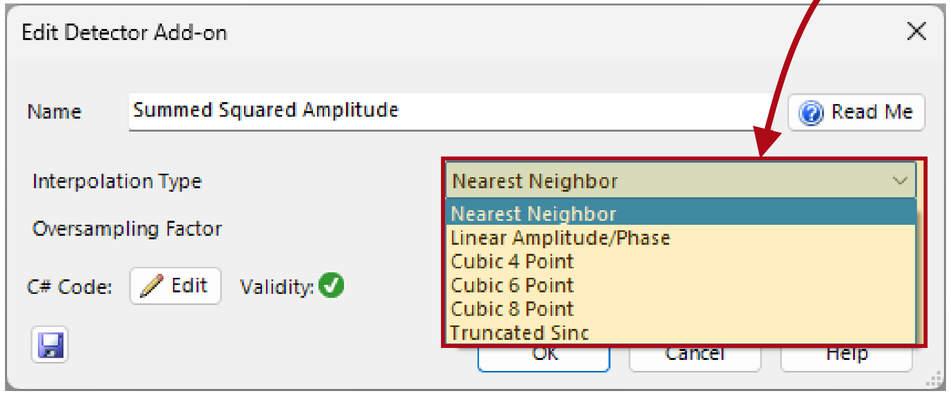
Similar to the first case, we want to restrict the input to 2D gridded data arrays. For a generalization of this concept to any kind of input, please see the documentation of the *Summed Squared Amplitude* add-on.



24: "Universal Detector" (#600) (Profile: General)
Electric Field (x-Domain, Equidistant)
Diagram
Table
Value at (x,y)
Amplitude of "Ex-Component" [kV/m]
2.3161
1.158
1....E-05
30: "Universal Detector" (#600) (Profile: General)
Electric Field (x-Domain, Equidistant)
Diagram
Table
Value at (x,y)
Amplitude of "Ey-Component" [V/m]
1
0.5
0
30: "Universal Detector" (#600) (Profile: General)
Electric Field (x-Domain, Equidistant)
Diagram
Table
Value at (x,y)
Amplitude of "Ez-Component" [kV/m]
0.55958
0.27979
1....E-06
23: My Detector Result – "Universal Detector" (#600): Summed Squared Amplitude (P...
Chromatic Fields Set
Y [µm]
X [µm]
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

Enumeration - Parameters

Enumeration parameter are an easy way to present pre-defined options of a parameter to the user. In the code each option is connected to an index, which can be called when necessary.



Name: Summed Squared Amplitude

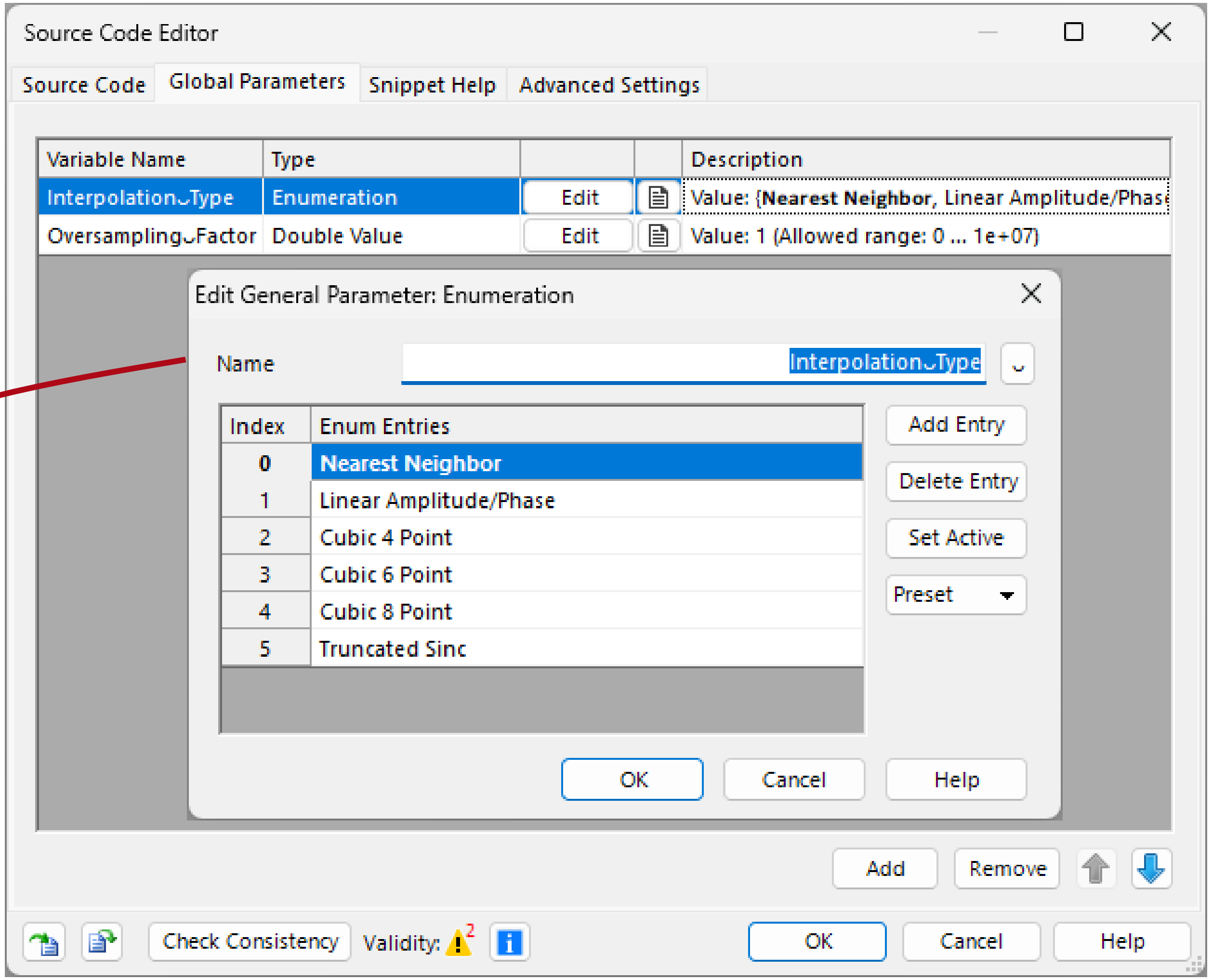
Interpolation Type: Nearest Neighbor

- Nearest Neighbor
- Linear Amplitude/Phase
- Cubic 4 Point
- Cubic 6 Point
- Cubic 8 Point
- Truncated Sinc

Oversampling Factor

C# Code:

Validity: ✓



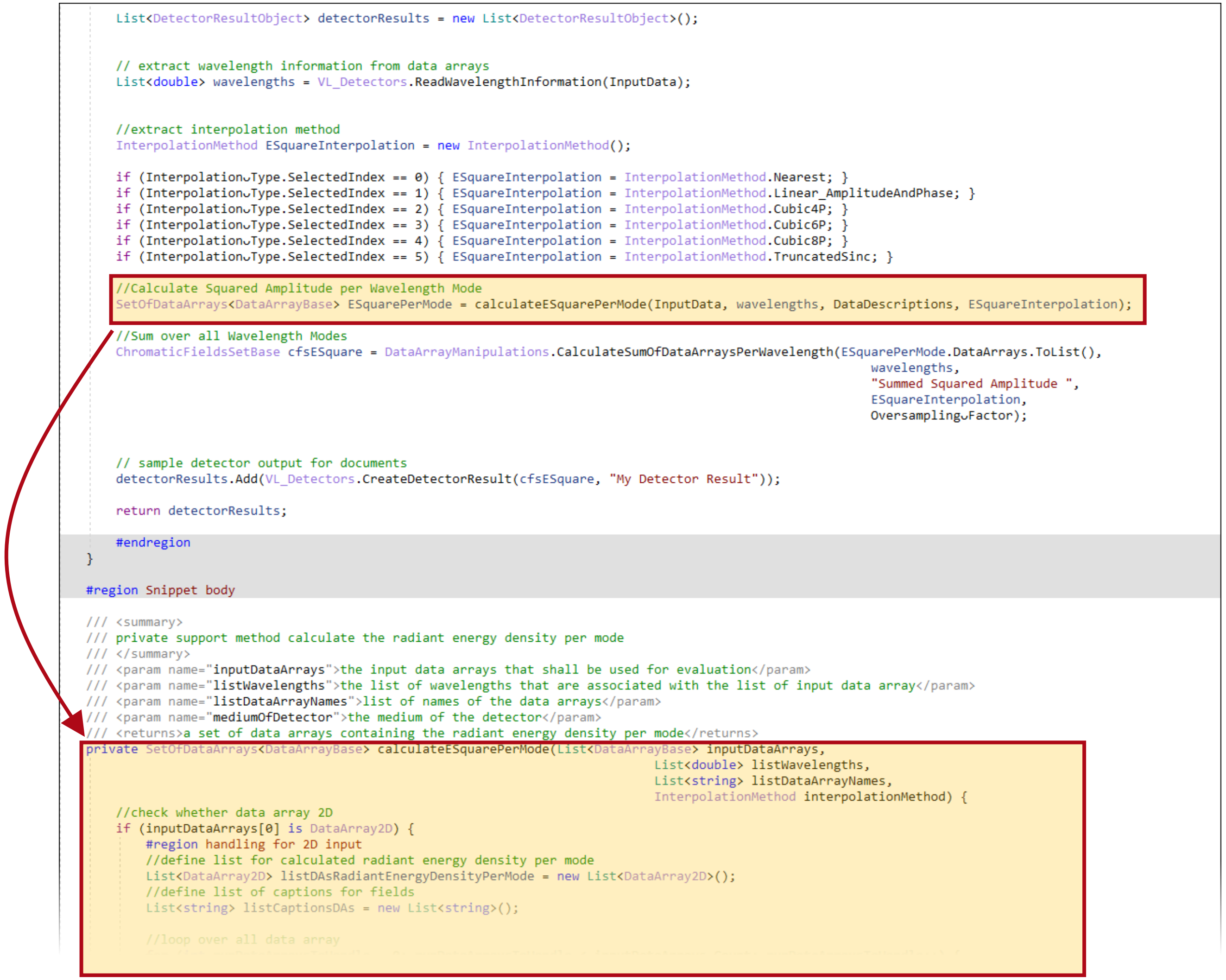
Variable Name	Type			Description
Interpolation␣Type	Enumeration	Edit	🗎	Value: {**Nearest Neighbor**, Linear Amplitude/Phase
Oversampling␣Factor	Double Value	Edit	🗎	Value: 1 (Allowed range: 0 ... 1e+07)

Name: Interpolation␣Type

Index	Enum Entries
0	**Nearest Neighbor**
1	Linear Amplitude/Phase
2	Cubic 4 Point
3	Cubic 6 Point
4	Cubic 8 Point
5	Truncated Sinc

Validity: ⚠ 2

Main & Snippet Body

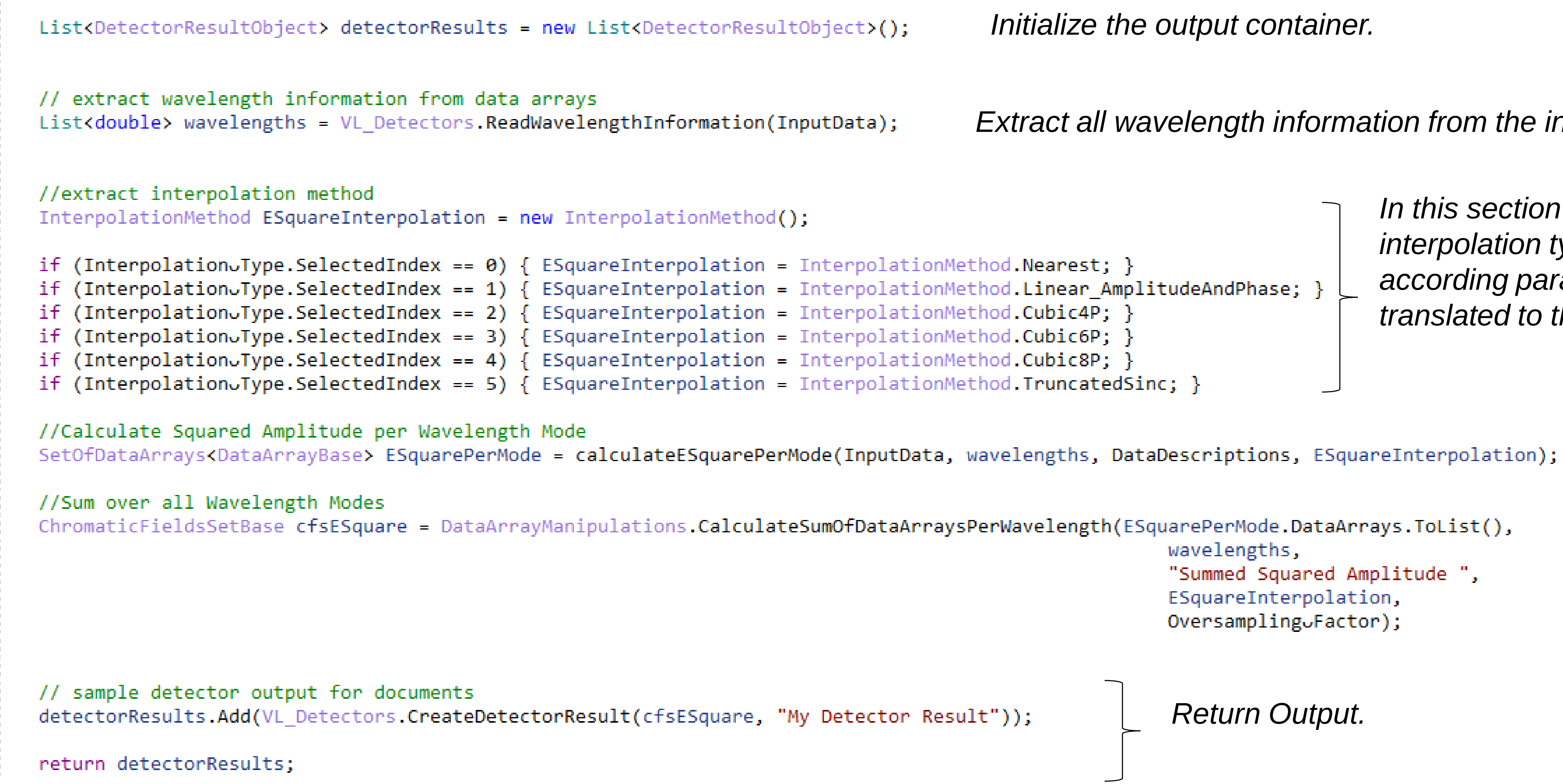


```
    List<DetectorResultObject> detectorResults = new List<DetectorResultObject>();


    // extract wavelength information from data arrays
    List<double> wavelengths = VL_Detectors.ReadWavelengthInformation(InputData);


    //extract interpolation method
    InterpolationMethod ESquareInterpolation = new InterpolationMethod();

    if (Interpolation‿Type.SelectedIndex == 0) { ESquareInterpolation = InterpolationMethod.Nearest; }
    if (Interpolation‿Type.SelectedIndex == 1) { ESquareInterpolation = InterpolationMethod.Linear_AmplitudeAndPhase; }
    if (Interpolation‿Type.SelectedIndex == 2) { ESquareInterpolation = InterpolationMethod.Cubic4P; }
    if (Interpolation‿Type.SelectedIndex == 3) { ESquareInterpolation = InterpolationMethod.Cubic6P; }
    if (Interpolation‿Type.SelectedIndex == 4) { ESquareInterpolation = InterpolationMethod.Cubic8P; }
    if (Interpolation‿Type.SelectedIndex == 5) { ESquareInterpolation = InterpolationMethod.TruncatedSinc; }

    //Calculate Squared Amplitude per Wavelength Mode
    SetOfDataArrays<DataArrayBase> ESquarePerMode = calculateESquarePerMode(InputData, wavelengths, DataDescriptions, ESquareInterpolation);

    //Sum over all Wavelength Modes
    ChromaticFieldsSetBase cfsESquare = DataArrayManipulations.CalculateSumOfDataArraysPerWavelength(ESquarePerMode.DataArrays.ToList(),
                                                                                                         wavelengths,
                                                                                                         "Summed Squared Amplitude ",
                                                                                                         ESquareInterpolation,
                                                                                                         Oversampling‿Factor);


    // sample detector output for documents
    detectorResults.Add(VL_Detectors.CreateDetectorResult(cfsESquare, "My Detector Result"));

    return detectorResults;

    #endregion
}

#region Snippet body

/// <summary>
/// private support method calculate the radiant energy density per mode
/// </summary>
/// <param name="inputDataArrays">the input data arrays that shall be used for evaluation</param>
/// <param name="listWavelengths">the list of wavelengths that are associated with the list of input data array</param>
/// <param name="listDataArrayNames">list of names of the data arrays</param>
/// <param name="mediumOfDetector">the medium of the detector</param>
/// <returns>a set of data arrays containing the radiant energy density per mode</returns>
private SetOfDataArrays<DataArrayBase> calculateESquarePerMode(List<DataArrayBase> inputDataArrays,
                                                                            List<double> listWavelengths,
                                                                            List<string> listDataArrayNames,
                                                                            InterpolationMethod interpolationMethod) {

    //check whether data array 2D
    if (inputDataArrays[0] is DataArray2D) {
        #region handling for 2D input
        //define list for calculated radiant energy density per mode
        List<DataArray2D> listDAsRadiantEnergyDensityPerMode = new List<DataArray2D>();
        //define list of captions for fields
        List<string> listCaptionsDAs = new List<string>();

        //loop over all data array
```

To make a code much more readable it is possible to define one's own functions in the *Snippet Body*, which then can be called in *Main Body*.

This is especially helpful, if the same function is called multiple times.

Source Code – Main Body



```
List<DetectorResultObject> detectorResults = new List<DetectorResultObject>();


// extract wavelength information from data arrays
List<double> wavelengths = VL_Detectors.ReadWavelengthInformation(InputData);


//extract interpolation method
InterpolationMethod ESquareInterpolation = new InterpolationMethod();

if (InterpolationᴗType.SelectedIndex == 0) { ESquareInterpolation = InterpolationMethod.Nearest; }
if (InterpolationᴗType.SelectedIndex == 1) { ESquareInterpolation = InterpolationMethod.Linear_AmplitudeAndPhase; }
if (InterpolationᴗType.SelectedIndex == 2) { ESquareInterpolation = InterpolationMethod.Cubic4P; }
if (InterpolationᴗType.SelectedIndex == 3) { ESquareInterpolation = InterpolationMethod.Cubic6P; }
if (InterpolationᴗType.SelectedIndex == 4) { ESquareInterpolation = InterpolationMethod.Cubic8P; }
if (InterpolationᴗType.SelectedIndex == 5) { ESquareInterpolation = InterpolationMethod.TruncatedSinc; }

//Calculate Squared Amplitude per Wavelength Mode
SetOfDataArrays<DataArrayBase> ESquarePerMode = calculateESquarePerMode(InputData, wavelengths, DataDescriptions, ESquareInterpolation);

//Sum over all Wavelength Modes
ChromaticFieldsSetBase cfsESquare = DataArrayManipulations.CalculateSumOfDataArraysPerWavelength(ESquarePerMode.DataArrays.ToList(),
                                                                                                     wavelengths,
                                                                                                     "Summed Squared Amplitude ",
                                                                                                     ESquareInterpolation,
                                                                                                     OversamplingᴗFactor);


// sample detector output for documents
detectorResults.Add(VL_Detectors.CreateDetectorResult(cfsESquare, "My Detector Result"));

return detectorResults;
```

Initialize the output container.

Extract all wavelength information from the input.

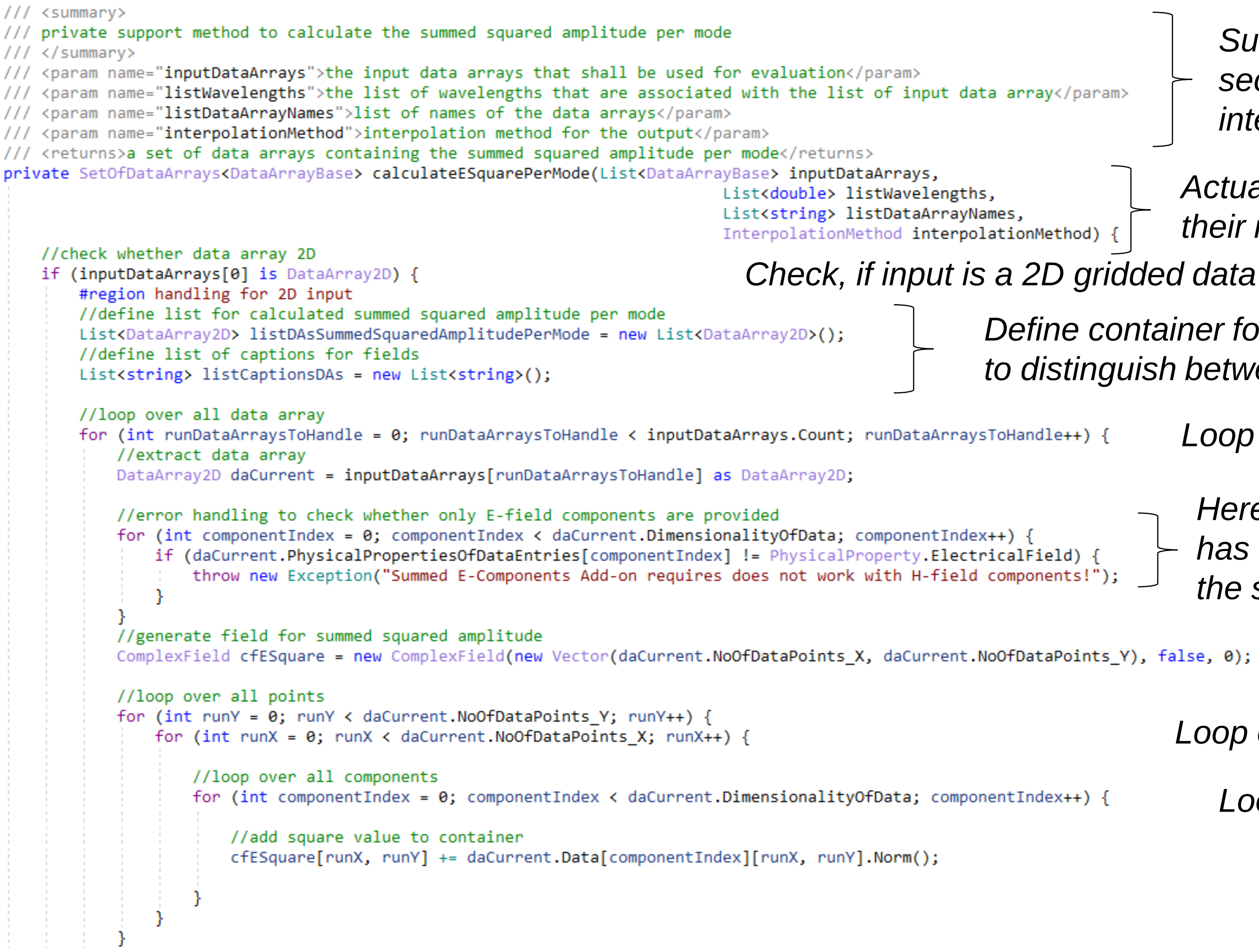
In this section we determine the interpolation type of the output. The according parameter therefore is translated to the corresponding class.

Call snippet body function to generate data array with summed squared amplitude per wavelength mode.

Integrate over all wavelengths.

Return Output.

Source Code – Snippet Body



```csharp
/// <summary>
/// private support method to calculate the summed squared amplitude per mode
/// </summary>
/// <param name="inputDataArrays">the input data arrays that shall be used for evaluation</param>
/// <param name="listWavelengths">the list of wavelengths that are associated with the list of input data array</param>
/// <param name="listDataArrayNames">list of names of the data arrays</param>
/// <param name="interpolationMethod">interpolation method for the output</param>
/// <returns>a set of data arrays containing the summed squared amplitude per mode</returns>
private SetOfDataArrays<DataArrayBase> calculateESquarePerMode(List<DataArrayBase> inputDataArrays,
                                                                            List<double> listWavelengths,
                                                                            List<string> listDataArrayNames,
                                                                            InterpolationMethod interpolationMethod) {

    //check whether data array 2D
    if (inputDataArrays[0] is DataArray2D) {
        #region handling for 2D input
        //define list for calculated summed squared amplitude per mode
        List<DataArray2D> listDAsSummedSquaredAmplitudePerMode = new List<DataArray2D>();
        //define list of captions for fields
        List<string> listCaptionsDAs = new List<string>();

        //loop over all data array
        for (int runDataArraysToHandle = 0; runDataArraysToHandle < inputDataArrays.Count; runDataArraysToHandle++) {
            //extract data array
            DataArray2D daCurrent = inputDataArrays[runDataArraysToHandle] as DataArray2D;

            //error handling to check whether only E-field components are provided
            for (int componentIndex = 0; componentIndex < daCurrent.DimensionalityOfData; componentIndex++) {
                if (daCurrent.PhysicalPropertiesOfDataEntries[componentIndex] != PhysicalProperty.ElectricalField) {
                    throw new Exception("Summed E-Components Add-on requires does not work with H-field components!");
                }
            }
            //generate field for summed squared amplitude
            ComplexField cfESquare = new ComplexField(new Vector(daCurrent.NoOfDataPoints_X, daCurrent.NoOfDataPoints_Y), false, 0);

            //loop over all points
            for (int runY = 0; runY < daCurrent.NoOfDataPoints_Y; runY++) {
                for (int runX = 0; runX < daCurrent.NoOfDataPoints_X; runX++) {

                    //loop over all components
                    for (int componentIndex = 0; componentIndex < daCurrent.DimensionalityOfData; componentIndex++) {

                        //add square value to container
                        cfESquare[runX, runY] += daCurrent.Data[componentIndex][runX, runY].Norm();

                    }
                }
            }
```

Summary of the parameters and result of the function. This section is not necessary but may be helpful to clarify the intent of the function.

Actual definition of the function. Here all parameters (and their respective classes) needs to be defined.

Check, if input is a 2D gridded data array.

Define container for output. The list of strings is necessary to distinguish between multiple wavelength modes.

Loop over all wavelengths.

Here we check if only Ex, Ey, Ez is active – as the H-field has a different unit and therefore shall not be included in the summed squared amplitude.

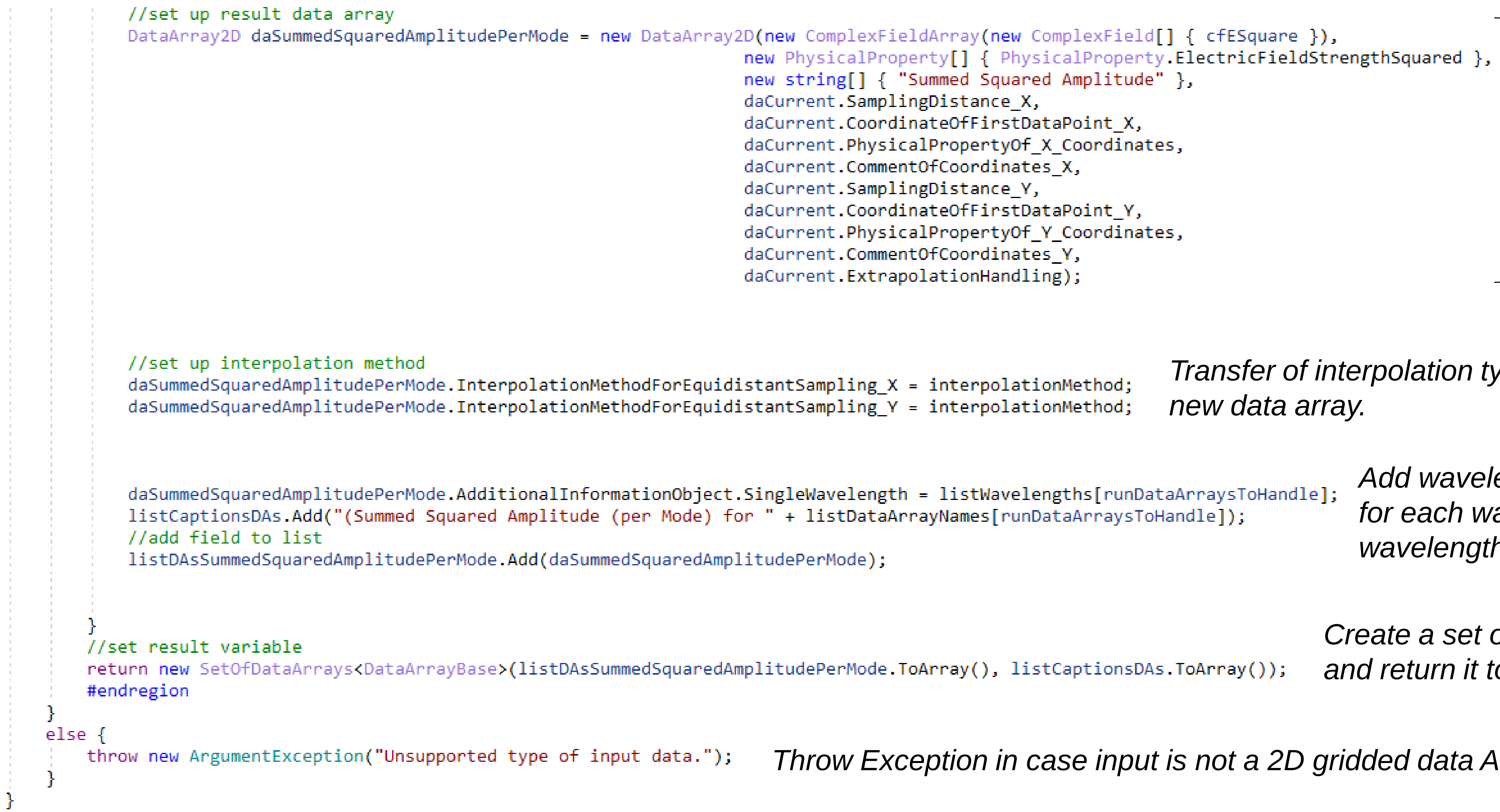
Create container for summed squared amplitude per mode.

Loop over all x,y.

Loop over components.

Calculate Squared Amplitude per point and add it to result.

Source Code – Snippet Body



```
            //set up result data array
            DataArray2D daSummedSquaredAmplitudePerMode = new DataArray2D(new ComplexFieldArray(new ComplexField[] { cfESquare }),
                                                                        new PhysicalProperty[] { PhysicalProperty.ElectricFieldStrengthSquared },
                                                                        new string[] { "Summed Squared Amplitude" },
                                                                        daCurrent.SamplingDistance_X,
                                                                        daCurrent.CoordinateOfFirstDataPoint_X,
                                                                        daCurrent.PhysicalPropertyOf_X_Coordinates,
                                                                        daCurrent.CommentOfCoordinates_X,
                                                                        daCurrent.SamplingDistance_Y,
                                                                        daCurrent.CoordinateOfFirstDataPoint_Y,
                                                                        daCurrent.PhysicalPropertyOf_Y_Coordinates,
                                                                        daCurrent.CommentOfCoordinates_Y,
                                                                        daCurrent.ExtrapolationHandling);



            //set up interpolation method
            daSummedSquaredAmplitudePerMode.InterpolationMethodForEquidistantSampling_X = interpolationMethod;
            daSummedSquaredAmplitudePerMode.InterpolationMethodForEquidistantSampling_Y = interpolationMethod;


            daSummedSquaredAmplitudePerMode.AdditionalInformationObject.SingleWavelength = listWavelengths[runDataArraysToHandle];
            listCaptionsDAs.Add("(Summed Squared Amplitude (per Mode) for " + listDataArrayNames[runDataArraysToHandle]);
            //add field to list
            listDAsSummedSquaredAmplitudePerMode.Add(daSummedSquaredAmplitudePerMode);


        }
        //set result variable
        return new SetOfDataArrays<DataArrayBase>(listDAsSummedSquaredAmplitudePerMode.ToArray(), listCaptionsDAs.ToArray());
        #endregion
    }
    else {
        throw new ArgumentException("Unsupported type of input data.");
    }
}
```

Conversion of the data container into data array that can be visualized in the software. Please find an overview over the different forms of data container on the next page.

Transfer of interpolation type information to the new data array.

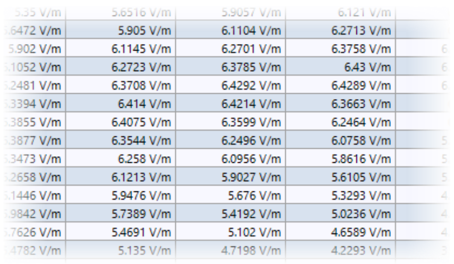
Add wavelength information and a caption for each wavelength mode to each wavelength mode and add it to a list.

Create a set of data arrays out of list and return it to the main body.

Throw Exception in case input is not a 2D gridded data Array.

An Overview of Data Container

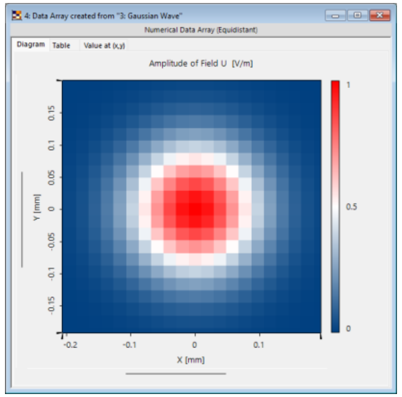
Complex Field:



5.35 V/m	5.6516 V/m	5.9057 V/m	6.121 V/m	
.6472 V/m	5.905 V/m	6.1104 V/m	6.2713 V/m	
5.902 V/m	6.1145 V/m	6.2701 V/m	6.3758 V/m	6
.1052 V/m	6.2723 V/m	6.3785 V/m	6.43 V/m	6
.2481 V/m	6.3708 V/m	6.4292 V/m	6.4289 V/m	6
.3394 V/m	6.414 V/m	6.4214 V/m	6.3663 V/m	
.3855 V/m	6.4075 V/m	6.3599 V/m	6.2464 V/m	
.3877 V/m	6.3544 V/m	6.2496 V/m	6.0758 V/m	5
.3473 V/m	6.258 V/m	6.0956 V/m	5.8616 V/m	5
.2658 V/m	6.1213 V/m	5.9027 V/m	5.6105 V/m	5
.1446 V/m	5.9476 V/m	5.676 V/m	5.3293 V/m	4
.9842 V/m	5.7389 V/m	5.4192 V/m	5.0236 V/m	4
.7626 V/m	5.4691 V/m	5.102 V/m	4.6589 V/m	4
.4782 V/m	5.135 V/m	4.7198 V/m	4.2293 V/m	3

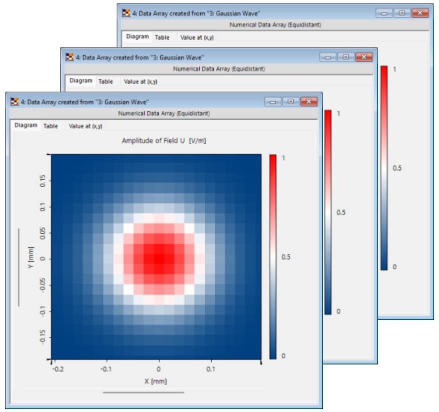
A 1D or 2D matrix of data entries. Entries can be complex or real and have a unit attached to them. But neither coordinates nor sampling information are included and thus this data container cannot be visualized as a field.

Data Array:


4: Data Array created from "3: Gaussian Wave"
Numerical Data Array (Equidistant)
Diagram
Table
Value at (x,y)
Amplitude of Field U [V/m]
Y [mm]
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.1
0
0.1
X [mm]
1
0.5
0

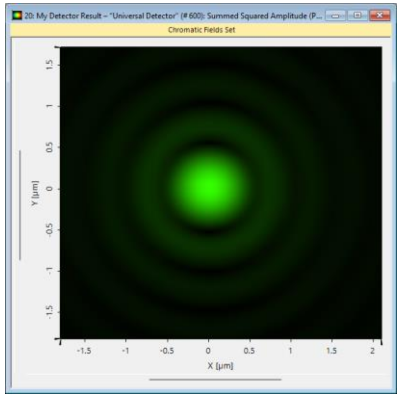
Data arrays normally include complex fields (or alternatively double arrays) as data and have additionally coordinates and sampling information specified. Hence, they can be visualized as fields. Data arrays can have multiple subsets, which all need to have the same sampling parameter (normally used for the components of a field).

Set of Data Array:


4: Data Array created from "3: Gaussian Wave"
Numerical Data Array (Equidistant)
Diagram
Table
Value at (x,y)
4: Data Array created from "3: Gaussian Wave"
Numerical Data Array (Equidistant)
Diagram
Table
Value at (x,y)
4: Data Array created from "3: Gaussian Wave"
Numerical Data Array (Equidistant)
Diagram
Table
Value at (x,y)
Amplitude of Field U [V/m]
Y [mm]
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.1
0
0.1
X [mm]
1
0.5
0
1
0.5
0
1
0.5
0

A set of data arrays contains multiple individual data arrays as subsets. Different to the case of the data array, these subsets can all have different sampling parameter. Normally these subsets represent the wavelength modes of a field, but could also be used for anything.

Chromatic Field Set:


20: My Detector Result – "Universal Detector" (# 600): Summed Squared Amplitude (P...
Chromatic Fields Set
Y [µm]
1.5
1
0.5
0
-0.5
-1
-1.5
-1.5
-1
-0.5
0
0.5
1
1.5
2
X [µm]

A *Chromatic Field Set* is a special version of *Set of Data Array* in which the subsets do need to represent the wavelength modes of the field. It offers additional functionalities and views in the main window.

Document Information

title	Programming Detector Add-ons in VirtualLab Fusion
document code	SWF.0058
document version	1.0
required packages	-
software version	2024.1 (Build 1.132)
category	Feature Use Case
further reading	- Universal Detector