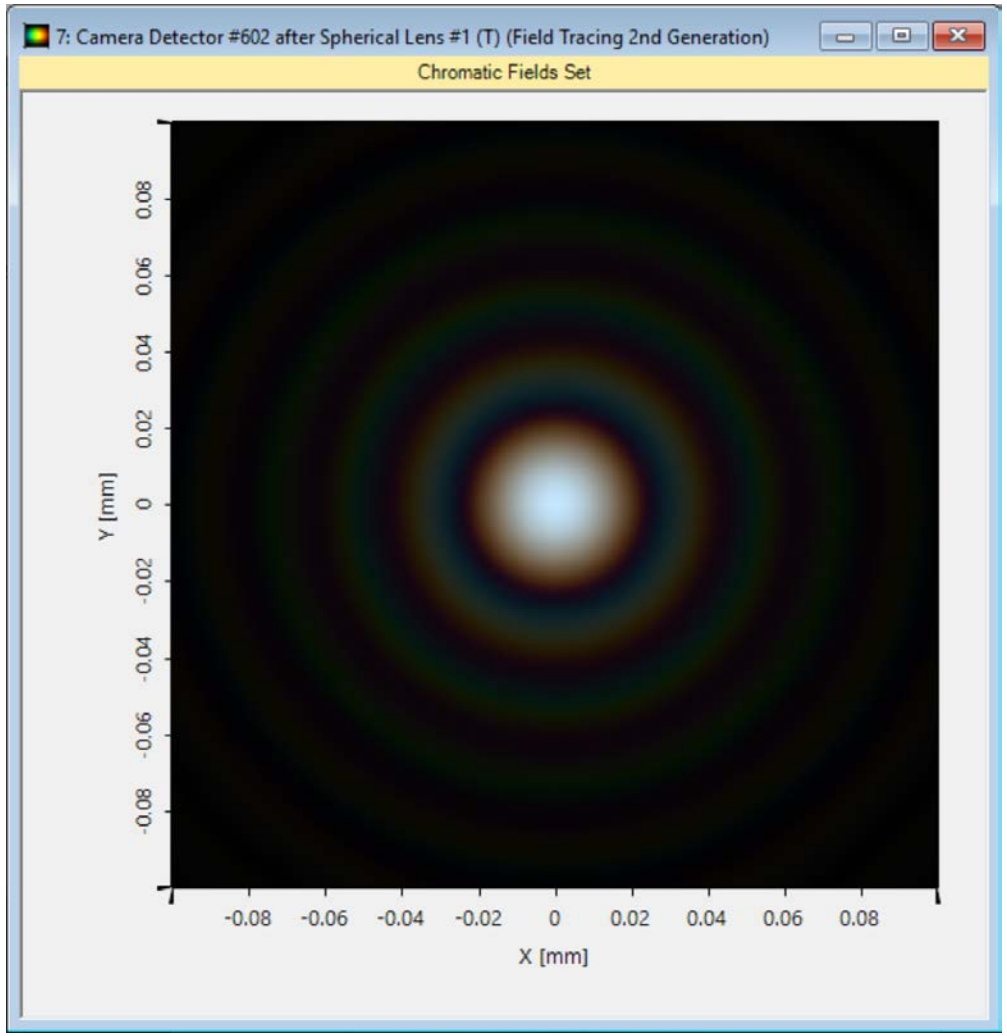
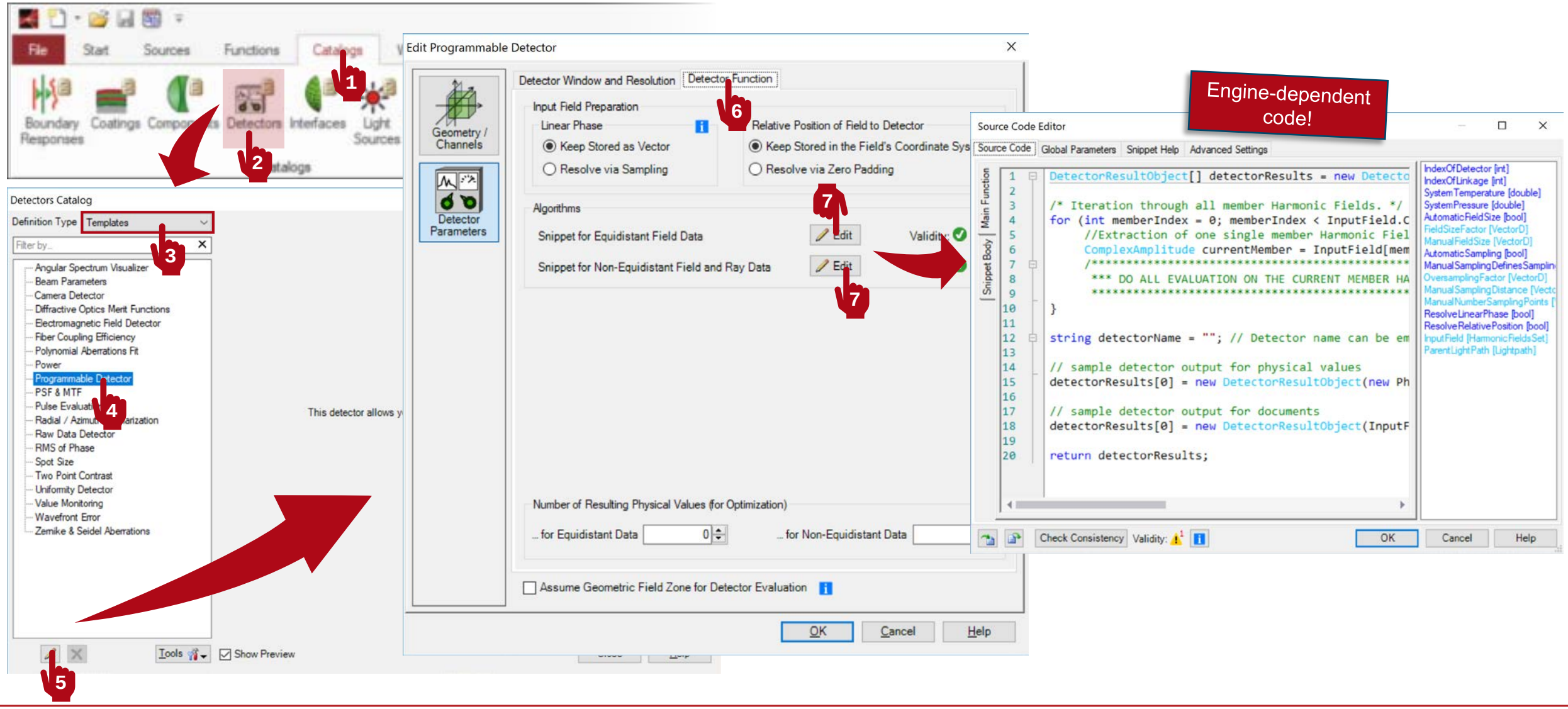
How to Work with the Programmable Detector and Example (Minimum and Maximum Wavelengths)

Abstract

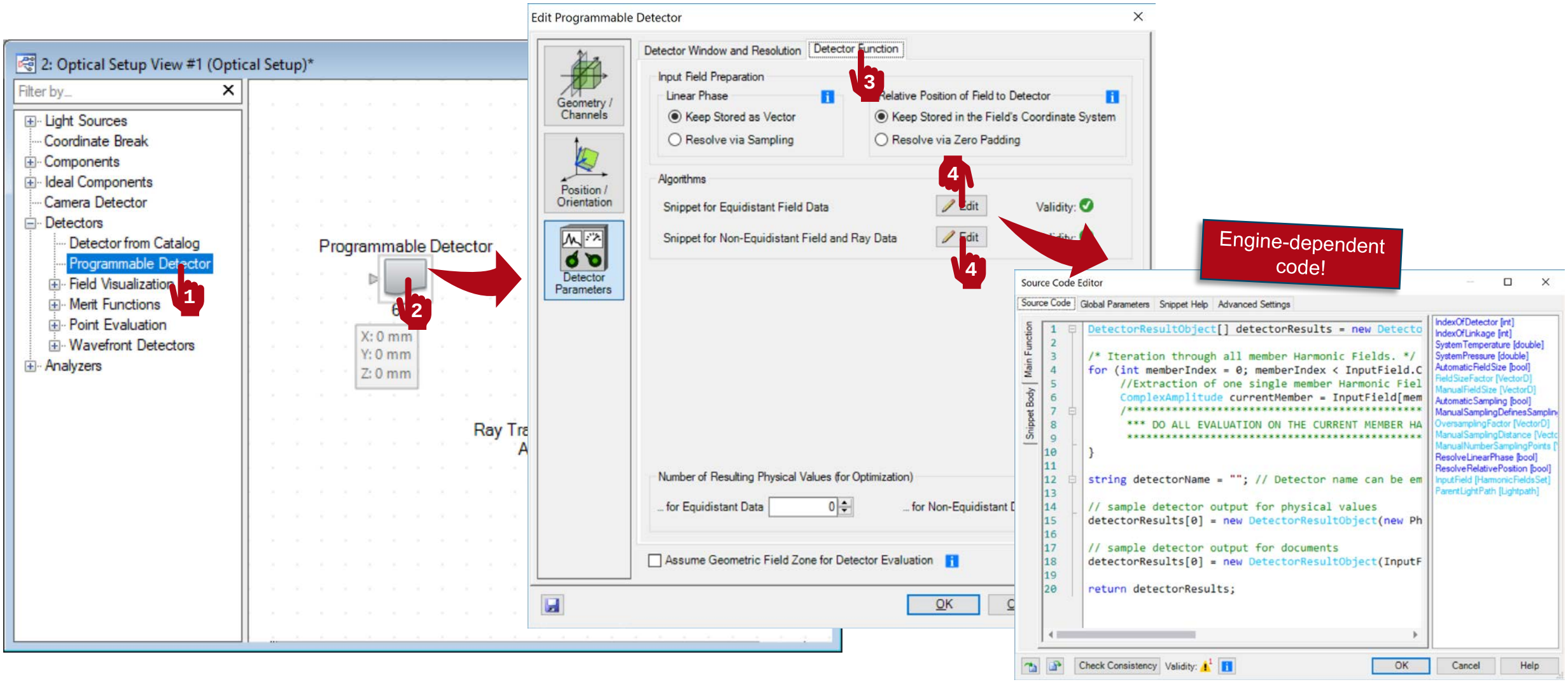


Providing maximum versatility for your optical simulations is one of our most fundamental objectives. In this tutorial we go over the Programmable Detector: any physical information contained in the vector field or ray bundles reaching the detector (depending on the simulation engine) can be accessed with maximum flexibility. We include here a simple programming example to illustrate its handling.

Where to Find the Programmable Detector: Catalog

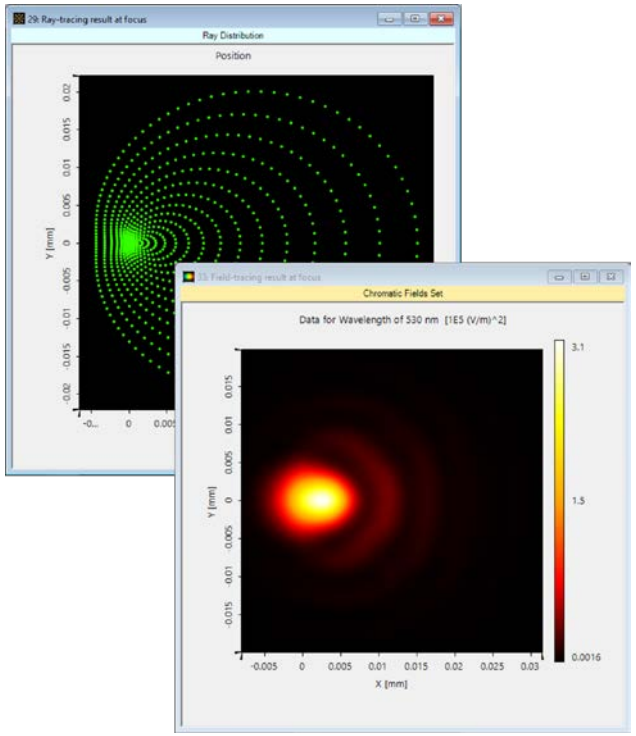


Where to Find the Programmable Detector: Optical Setup

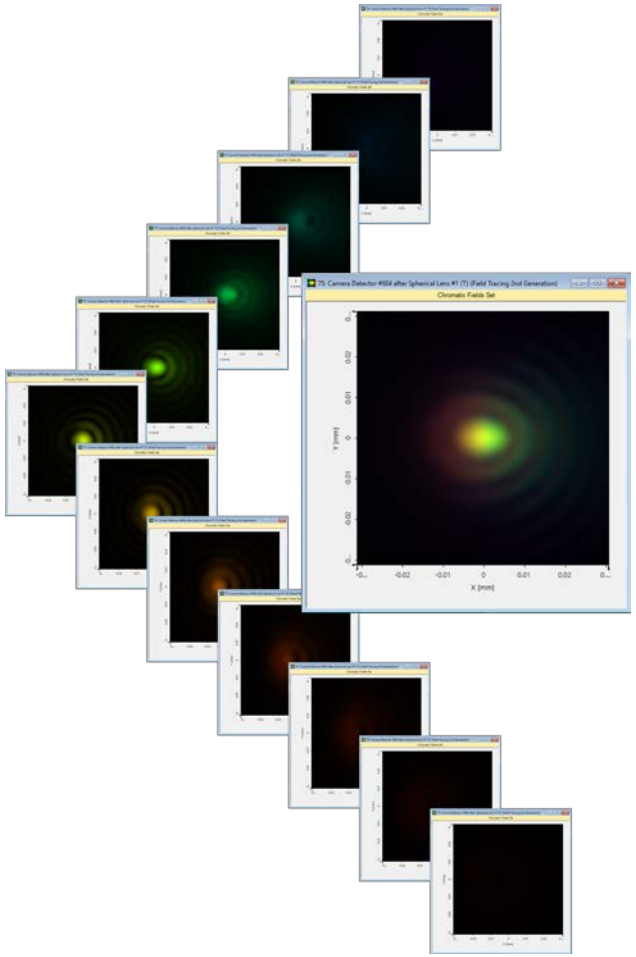


A Note on the Light Representation

- The vector electromagnetic field that represents light in physical optics is always fully accessible in VirtualLab Fusion as it is traced through the system.
- For this approach to be practical from the point of view of computational efficiency, it is paramount to have at our disposal a diverse set of mathematical techniques (efficient Fourier transform algorithms, interpolation and fitting methods, heterogeneous sampling mechanisms, among others).
- In the current version of VirtualLab Fusion, this translates into the coexistence of several simulation engines:
  - Ray tracing: pure ray tracing, yielding both 2 and 3D results
  - Classic Field Tracing: handles equidistantly sampled EM field data
  - 2nd Generation Field Tracing: is also able to handle non-equidistant EM field data
- This is relevant to the Programmable Detector: a good implementation of your detector needs to take into account how light is represented in the different engines!

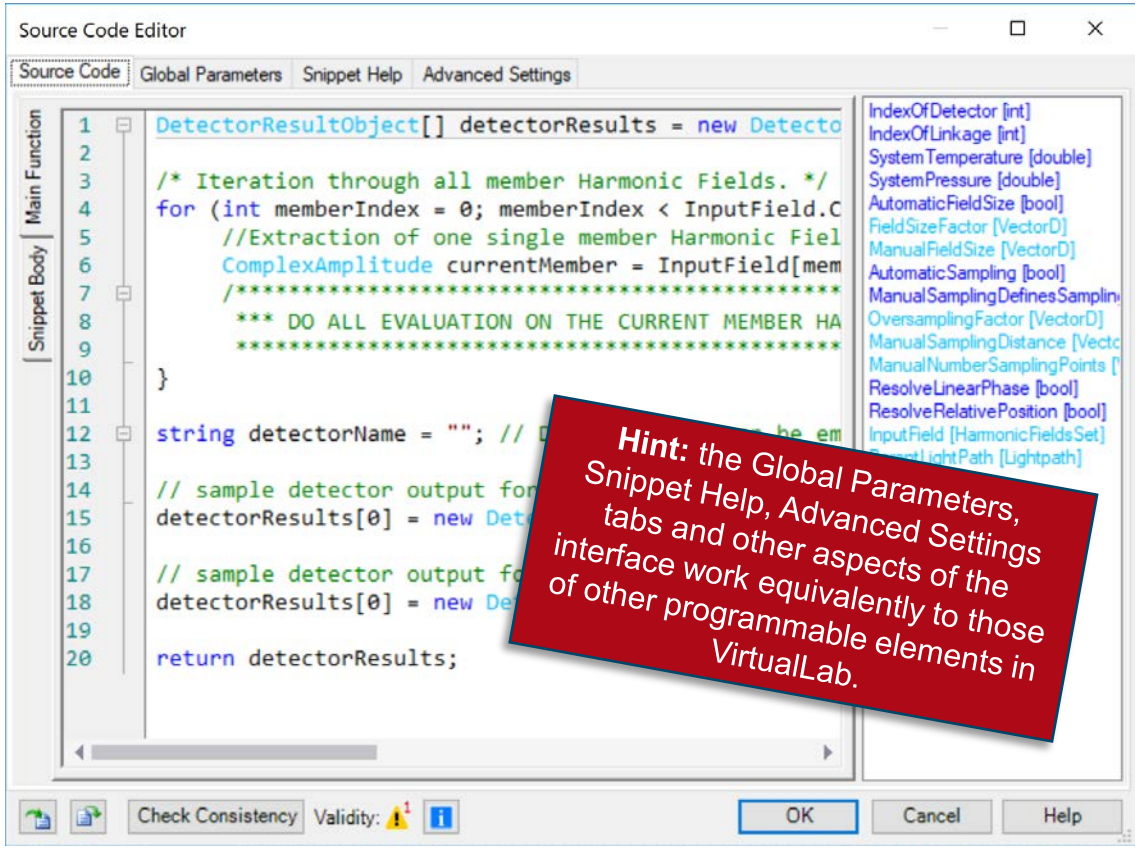


A Note on the Light Representation



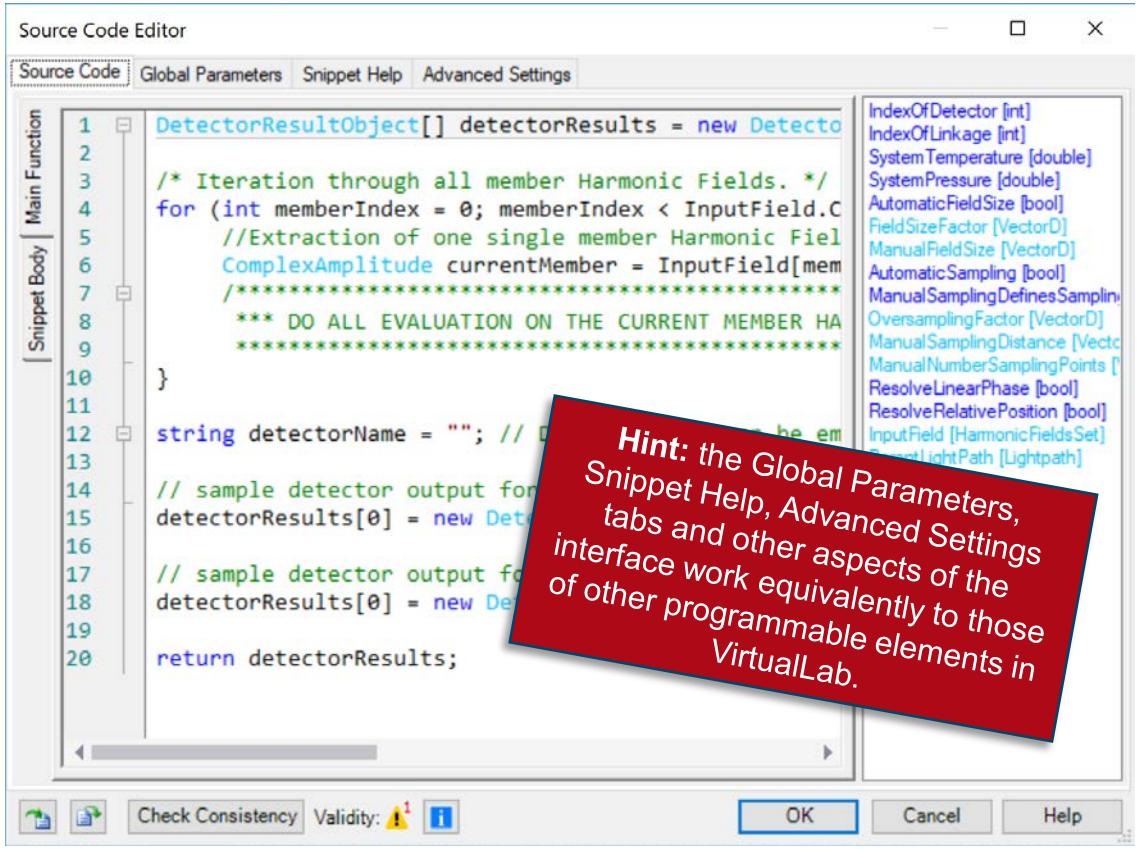
- Additionally, in order to replicate a series of important physical properties of light (partial coherence, for instance, whether temporal or spatial) VirtualLab uses a mode decomposition.
- The different modes are accessible in the Programmable Detector via a series of indices.
- Taking the different modes into account is also fundamental if a Programmable Detector is to exhibit the correct desired physical behaviour!

Writing the Code: Equidistant Field Data



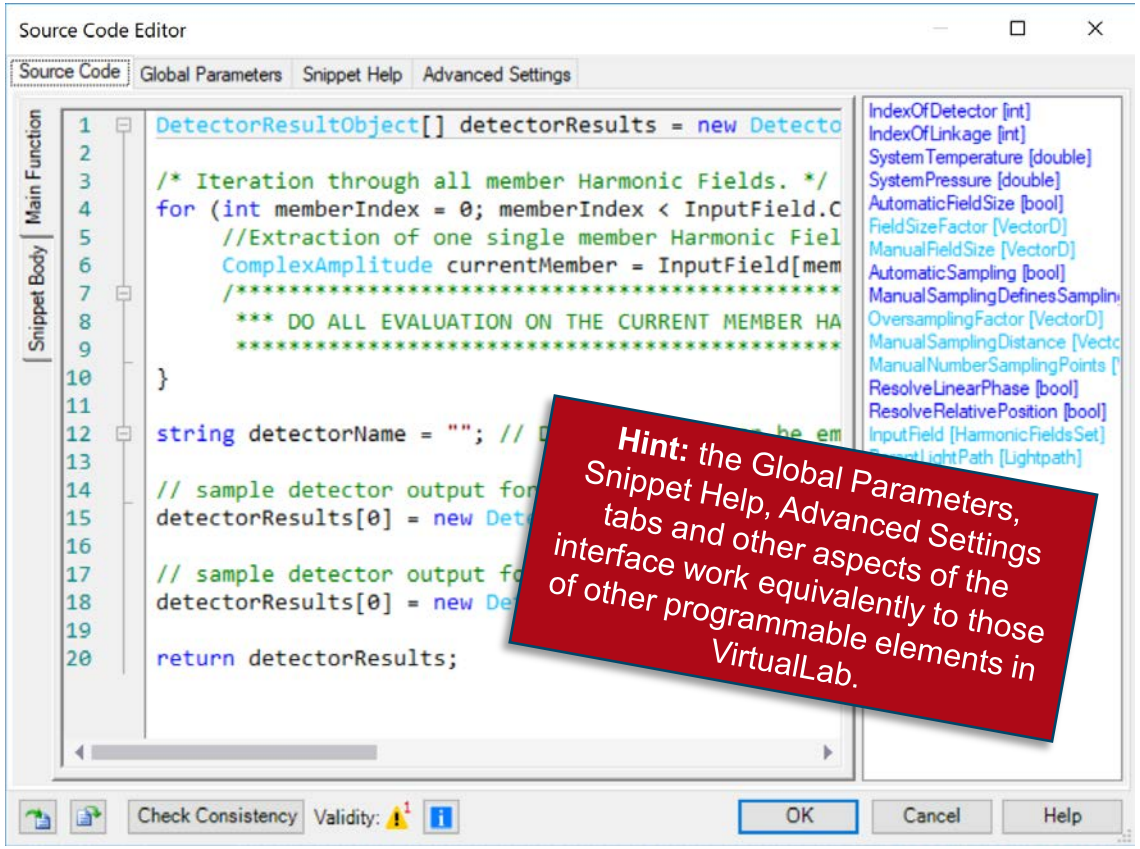
- The Programmable Detector provides two different programming dialogs. These are related to the simulation engines. The first, titled Snippet for Equidistant Field Data, handles electromagnetic field objects sampled on an equidistant, rectangular *x, y* grid.
- It is a direct result of Maxwell's equations that in homogeneous media only two of the six electromagnetic components are independent; consequently, the fields reaching the detector consist only of *Ex* and *Ey* components, all the others being thus unequivocally determined and possible to calculate on demand if so required.
- Depending on the polarization characteristics of the incoming field, *Ex* and *Ey* can be two independent functions (local polarization) or obtained from a single field function U via a constant Jones' vector (constant in *x* and *y*), so that *Ex* = *Jx* * *U* and *Ey* = *Jy* * *U*.

Writing the Code: Equidistant Field Data



- The panel on the right shows a list of available independent parameters.
- `IndexOfDetector` and `IndexOfLinkage` refer to the corresponding elements in the configuration of the Optical Setup containing the detector in question.
- `SystemTemperature` and `SystemPressure` are parameters of the whole system, whose value can be used in the code to implement temperature- and pressure-dependent responses.
- `AutomaticFieldSize`, …, `ManualNumberSamplingPoints` are all parameters which must influence the ultimate sampling of the eventual equidistantly sampled field result, and whose value can be modified in the Detector Window and Resolution tab of the detector configuration dialog.

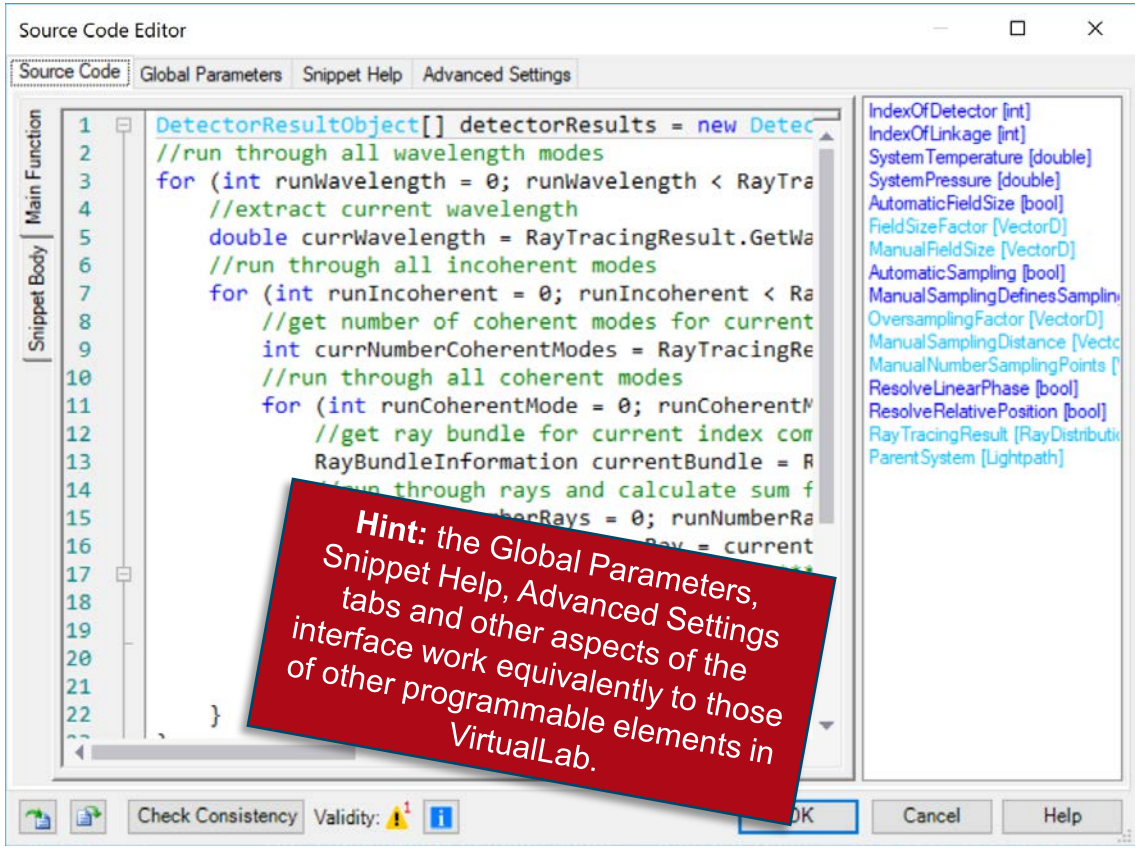
Writing the Code: Equidistant Field Data



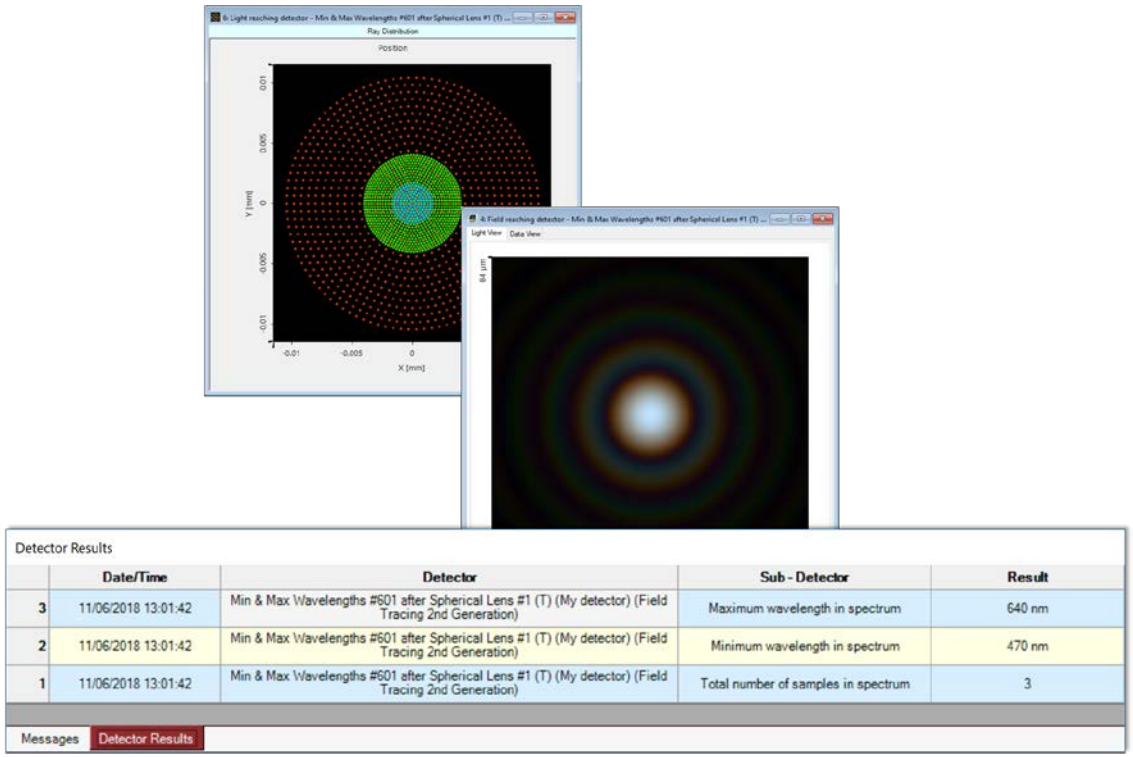
- `ResolveLinearPhase` and `ResolveRelativePosition` are flags whose value can be modified in the detector configuration dialog, in the Detector Function tab. They indicate the user's wish to keep linear phases and position shifts stored in the internal coordinate system of the field or, conversely, resolved explicitly (which results in higher sampling requirements, as per Shannon-Nyquist). It is the programmer's responsibility to implement a code which reflects these wishes correctly one way or another.
- `InputField` represents the field (equidistantly sampled) reaching the detector. Following VirtualLab's mode concept, it consists of a set of fully self-correlated electromagnetic modes, which can exhibit different coherence properties among themselves to faithfully mimic the coherence properties of the physical field.
- `ParentLightPath` refers to the Optical Setup containing the detector in question. Use the Snippet Body to group parts of the code in support functions.

Writing the Code: Non-Equidistant Field and Ray Data

- The other programming dialog in the Programmable Detector handles non-equidistantly sampled field data and rays.
- The panel on the right shows, again, a list of available independent parameters.
- The only difference with the snippet for equidistantly sampled field is in the fact that `InputField` is replaced by `RayTracingResult`.
- Do not let the name `RayTracingResult` fool you! This nomenclature is obsolete and will be phased out in future versions.
- For non-equidistant fields, the vector field samples may coincide with the ray samples. This snippet can therefore return both ray information—if the simulation is run with the Ray Tracing Engine—and physical optics results—when the chosen engine is 2nd Generation Field Tracing. It is the programmer's responsibility to account for both instances.



Output



- The Programmable Detector must return, for both snippets, a `DetectorResultObject[]` array.
- This type of object may contain both
  - Physical magnitudes: for instance, a detector that computes the power carried by
  - 2D graphic representations: think a detector that shows all six electromagnetic components in the detector plane.
- Each of the `DetectorResultObject[i]` corresponds to either one physical magnitude or one 2D graphic.
- The results of the Programmable Detector can be used in the Parameter Run or Parametric Optimization!
- The custom detector can be saved in the catalog for later use.

Programming a Detector That Retrieves the Minimum and Maximum Wavelengths in the Incoming Spectrum

Specifications Of the Desired Custom Detector

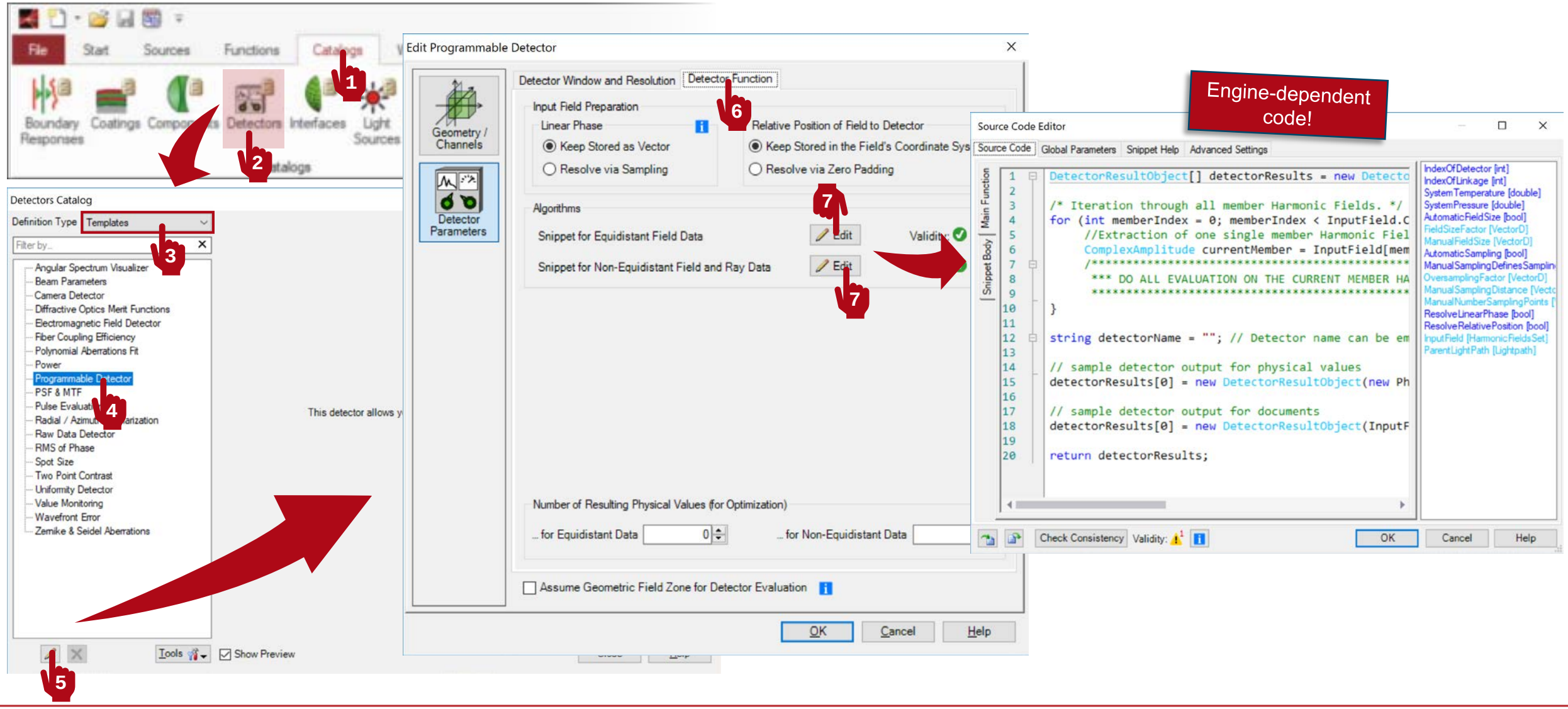
- The custom detector resulting from this exercise must work across the board, for ray and both field tracing engines.

- The Programmable Detector will yield at least three results:
  - The total number of samples in the spectrum
  - The value of the minimum wavelength present in the spectrum
  - The value of the maximum wavelength present in the spectrum

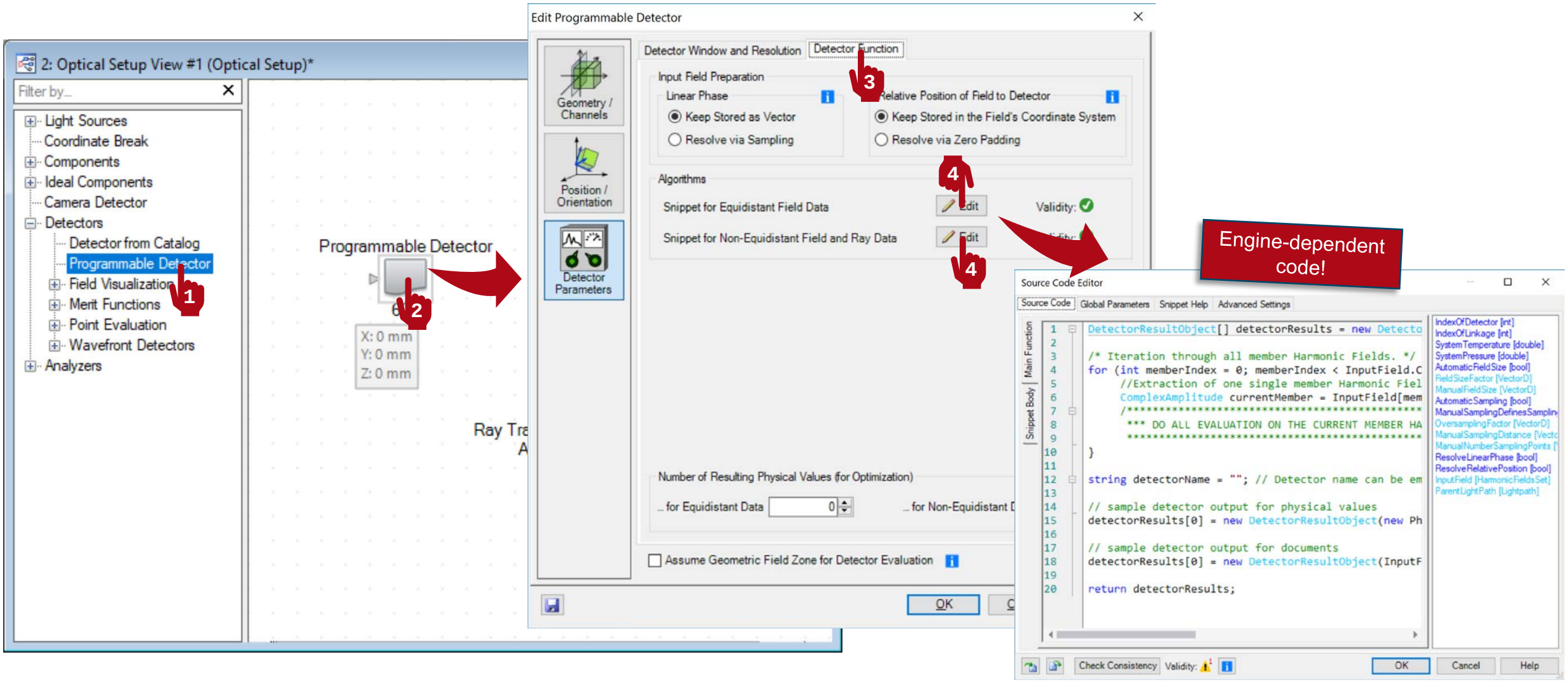
- Additionally, a user-controlled Boolean parameter will be included.

- This Boolean parameter will allow the user to select whether they want an additional result to be returned: this additional result corresponds to the light-representing object (rays or fields) reaching the detector.

Where to Find the Programmable Detector: Catalog

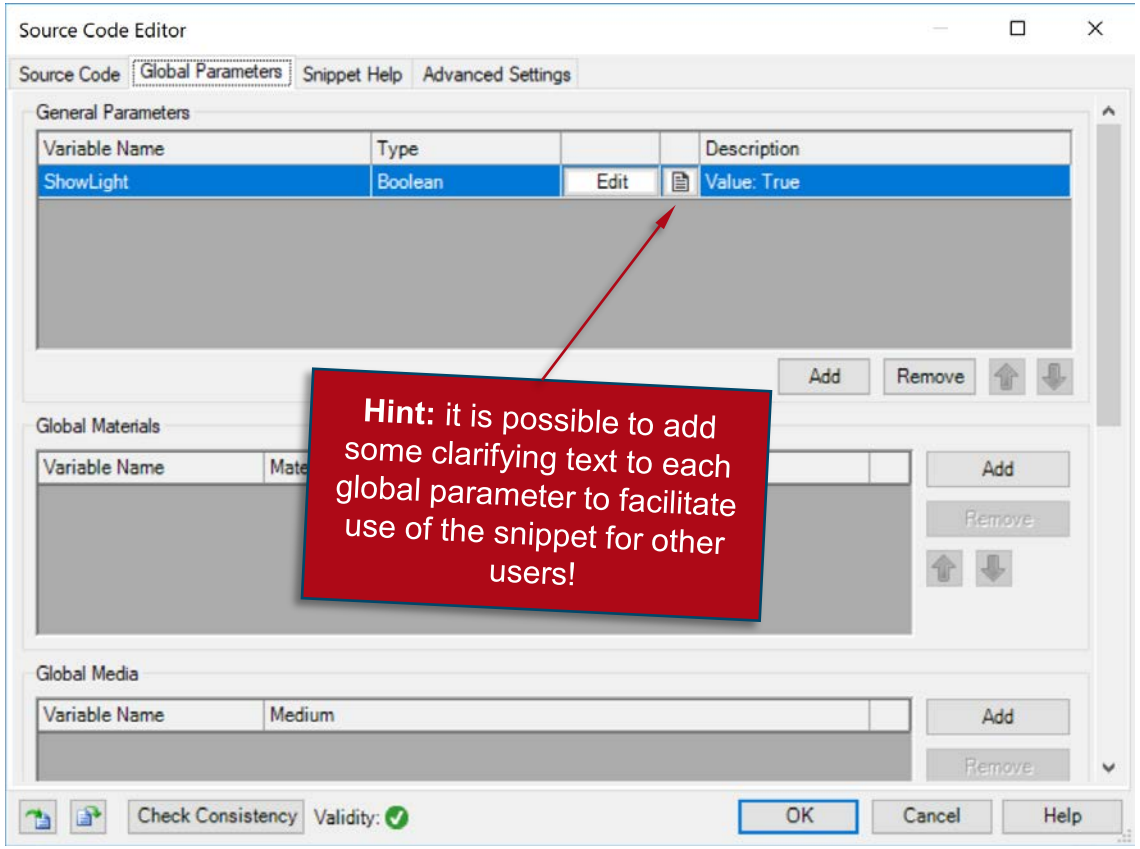


Where to Find the Programmable Detector: Optical Setup

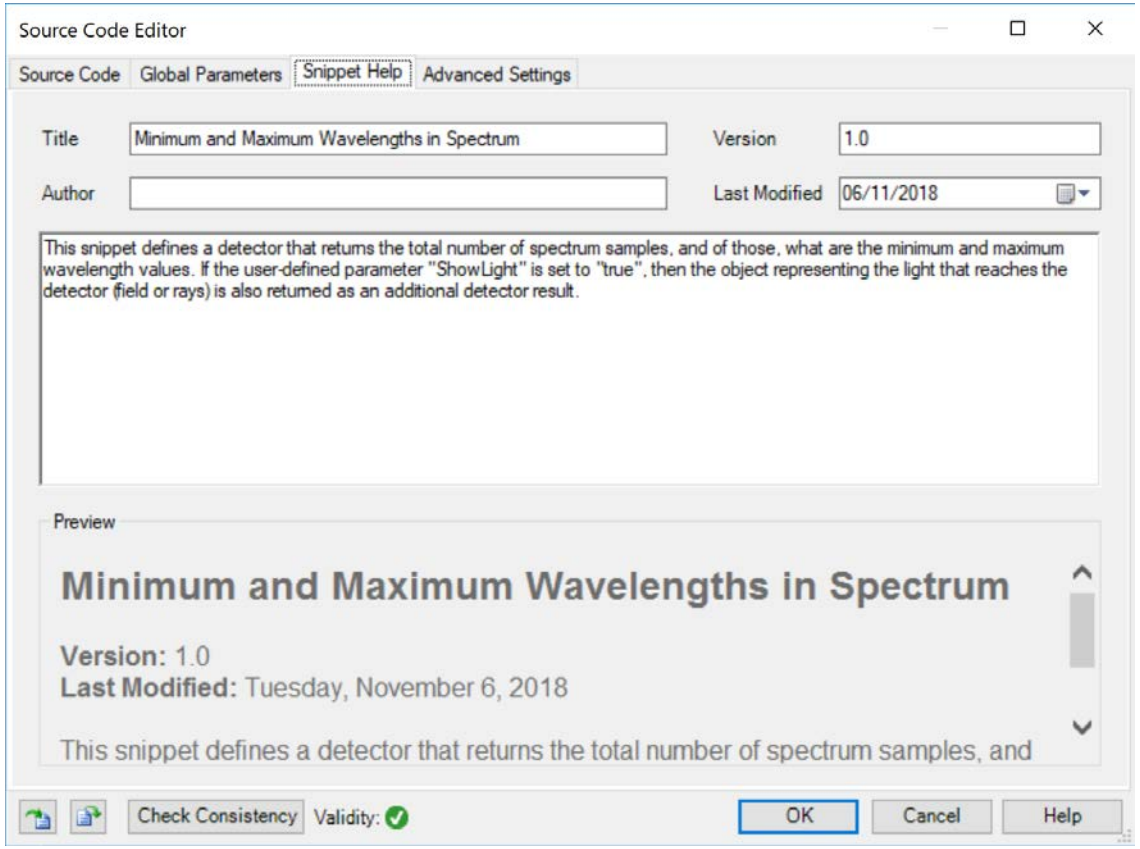


Programmable Detector: Global Parameters

- Once you have triggered open the Edit dialog, go to the Global Parameters tab.
- There, Add and Edit one global parameter:
  - `Boolean ShowLight = false (false, true)`: a user-defined parameter which serves to determine whether the object representing the light that reaches the detector (vector field or rays) will be returned as a detector result alongside the values of the minimum and maximum wavelengths present in the spectrum.
- Note that the Global Parameters, Snippet Help and Advanced Settings tabs, and the Snippet Body are all shared by and common to the two specification modes (equidistantly sampled fields, and rays and non-equidistantly sampled fields).

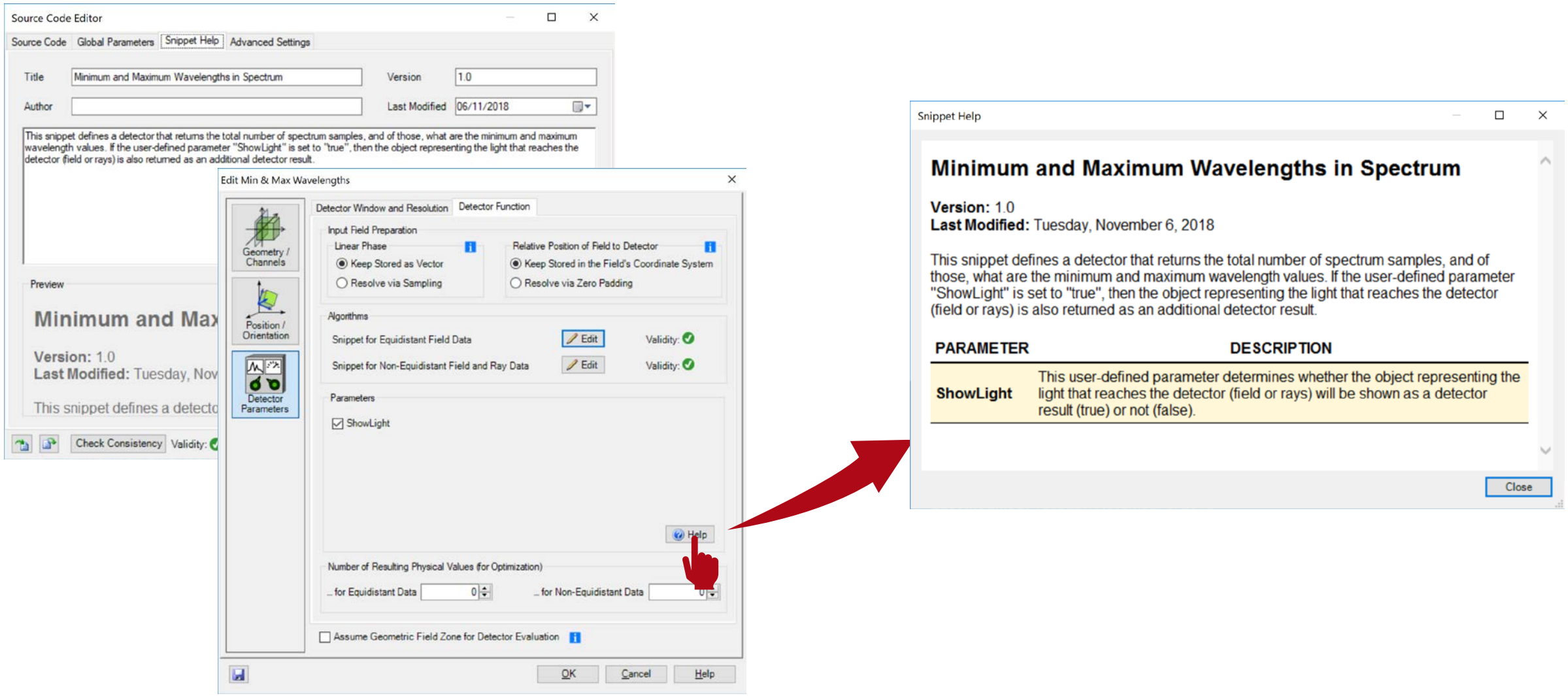


Programmable Detector: Snippet Help



- **Optional:** you can use the Snippet Help tab to write instructions, clarifications, and some metadata associated to your snippet.
- This option is very helpful to keep track of your progress with a programmable element.
- It is especially useful when the programmable element is later disseminated to be handled by other users!

Programmable Detector: Snippet Help



Programmable Detector: Writing the Code (1)

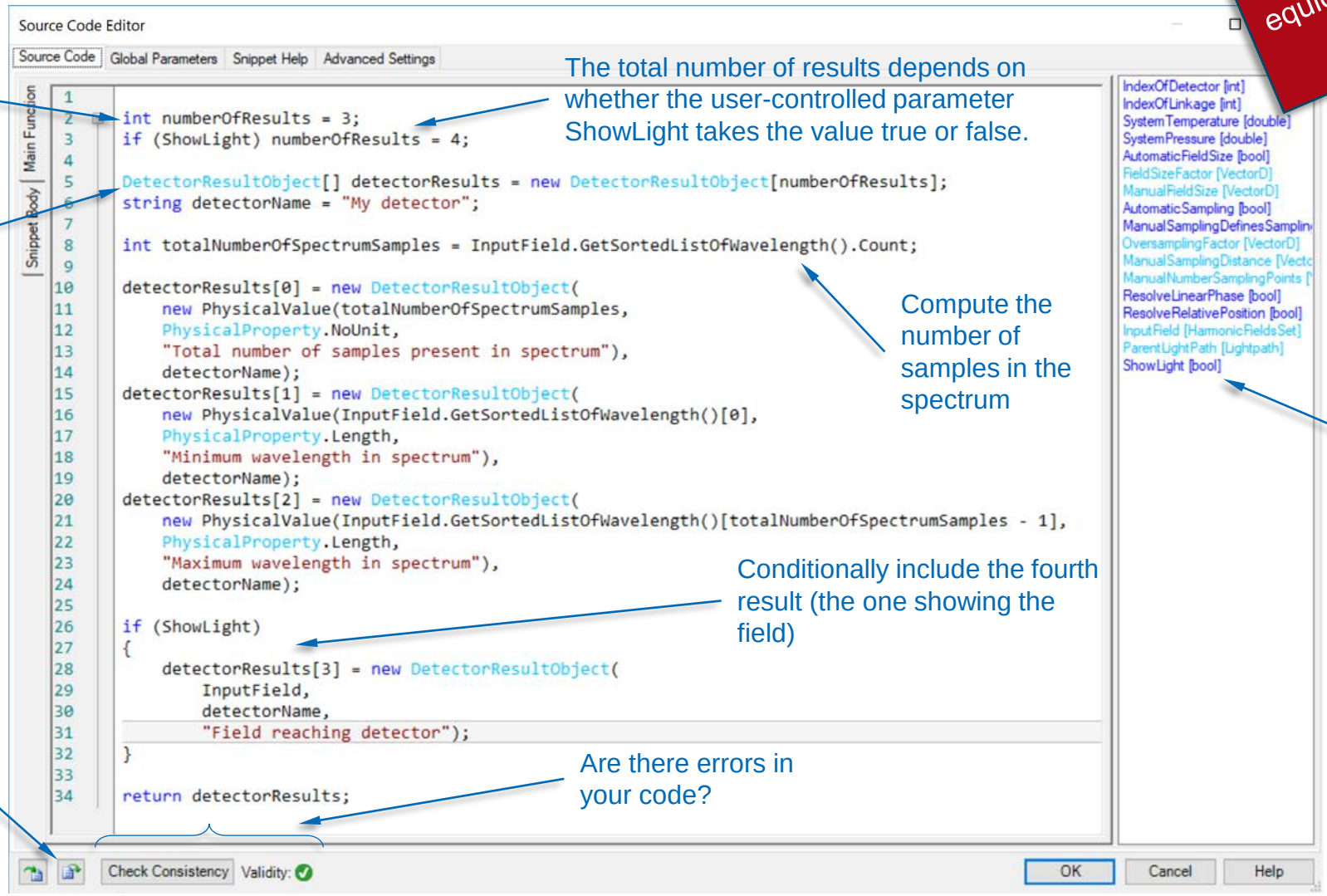
Snippet for equidistantly sampled field results!

Declare and assign a parameter for the dimension of the DetectorResultObject[] array

Declare the object to be returned by the code

Include the three default results in the DetectorResultsObject[] to be returned

Export Snippet to save your work!



The total number of results depends on whether the user-controlled parameter ShowLight takes the value true or false.

Compute the number of samples in the spectrum

Conditionally include the fourth result (the one showing the field)

Are there errors in your code?

Default global parameters/variables

Global parameter defined by user in Global Parameters tab

Programmable Detector: Writing the Code (2)

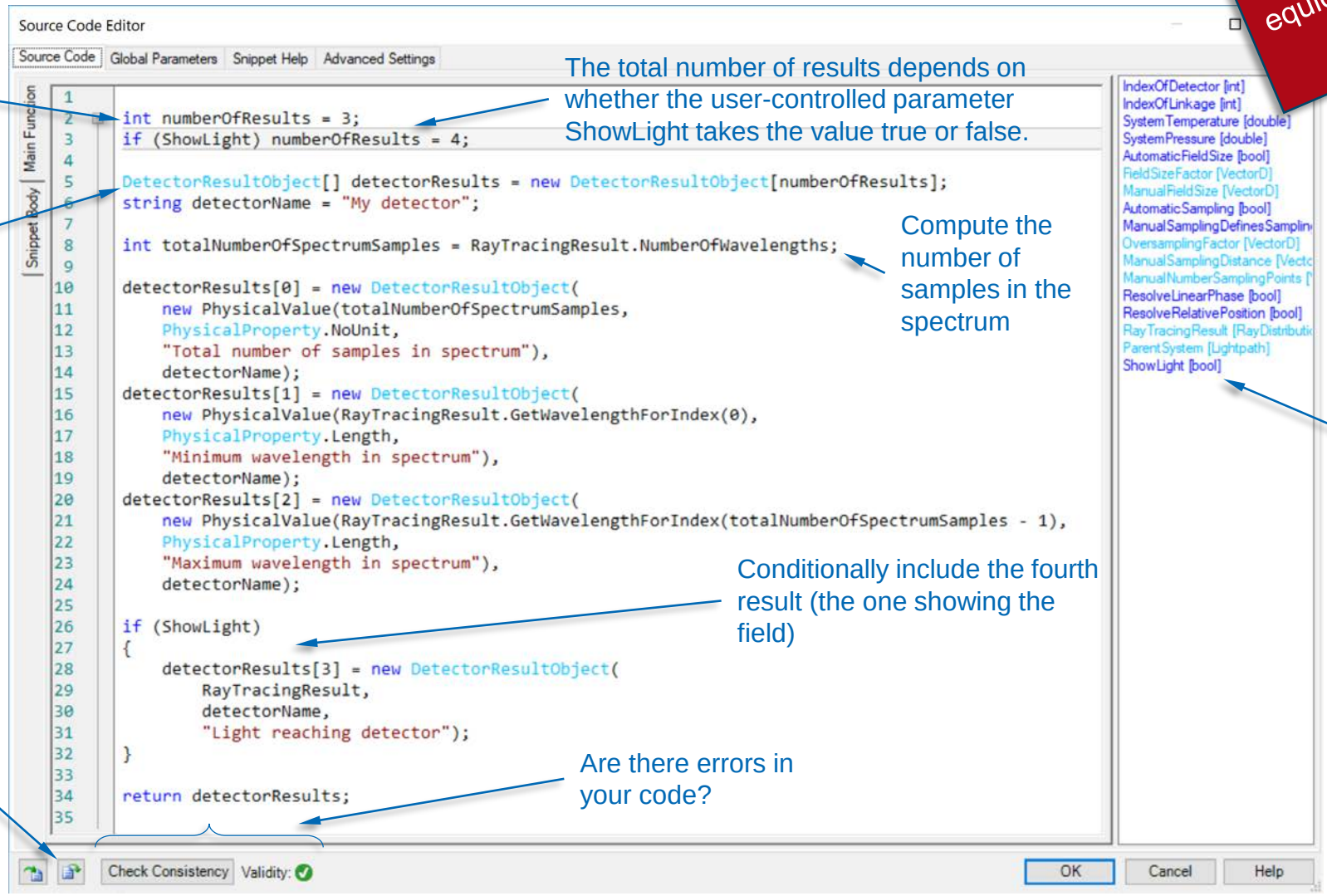
Snippet for ray & non-equidistantly sampled field results!

Declare and assign a parameter for the dimension of the DetectorResultObject[] array

Declare the object to be returned by the code

Include the three default results in the DetectorResultsObject[] to be returned

Export Snippet to save your work!



The total number of results depends on whether the user-controlled parameter ShowLight takes the value true or false.

Compute the number of samples in the spectrum

Conditionally include the fourth result (the one showing the field)

Are there errors in your code?

Default global parameters/variables

Global parameter defined by user in Global Parameters tab

Programmable Detector: Comparing the Snippets

Snippet for equidistantly sampled field results!

Snippet Help | Advanced Settings

```
int numberOfResults = 3;
if (ShowLight) numberOfResults = 4;

DetectorResultObject[] detectorResults = new DetectorResultObject
string detectorName = "My detector";

int totalNumberOfSpectrumSamples = InputField.GetSortedList

detectorResults[0] = new DetectorResultObject(
    new PhysicalValue(totalNumberOfSpectrumSamples,
    PhysicalProperty.NoUnit,
    "Total number of samples present in spectrum"),
    detectorName);
detectorResults[1] = new DetectorResultObject(
    new PhysicalValue(InputField.GetSortedListOfWavelength()
    PhysicalProperty.Length,
    "Minimum wavelength in spectrum"),
    detectorName);
detectorResults[2] = new DetectorResultObject(
    new PhysicalValue(InputField.GetSortedListOfWavelengt
    PhysicalProperty.Length,
    "Maximum wavelength in spectrum"),
    detectorName);

if (ShowLight)
{
    detectorResults[3] = new DetectorResultObject(
        InputField,
        detectorName,
        "Field reaching detector");
}

return detectorResults;
```

Check Consistency | Validity:

Snippet for ray & non-equidistantly sampled field results!

Snippet Help | Advanced Settings

```
int numberOfResults = 3;
if (ShowLight) numberOfResults = 4;

DetectorResultObject[] detectorResults = new DetectorResultObject
string detectorName = "My detector";

int totalNumberOfSpectrumSamples = RayTracingResult.NumberOfWa

detectorResults[0] = new DetectorResultObject(
    new PhysicalValue(totalNumberOfSpectrumSamples,
    PhysicalProperty.NoUnit,
    "Total number of samples in spectrum"),
    detectorName);
detectorResults[1] = new DetectorResultObject(
    new PhysicalValue(RayTracingResult.GetWavelengthForIndex(0
    PhysicalProperty.Length,
    "Minimum wavelength in spectrum"),
    detectorName);
detectorResults[2] = new DetectorResultObject(
    new PhysicalValue(RayTracingResult.GetWavelengthForIndex
    PhysicalProperty.Length,
    "Maximum wavelength in spectrum"),
    detectorName);

if (ShowLight)
{
    detectorResults[3] = new DetectorResultObject(
        RayTracingResult,
        detectorName,
        "Light reaching detector");
}

return detectorResults;
```

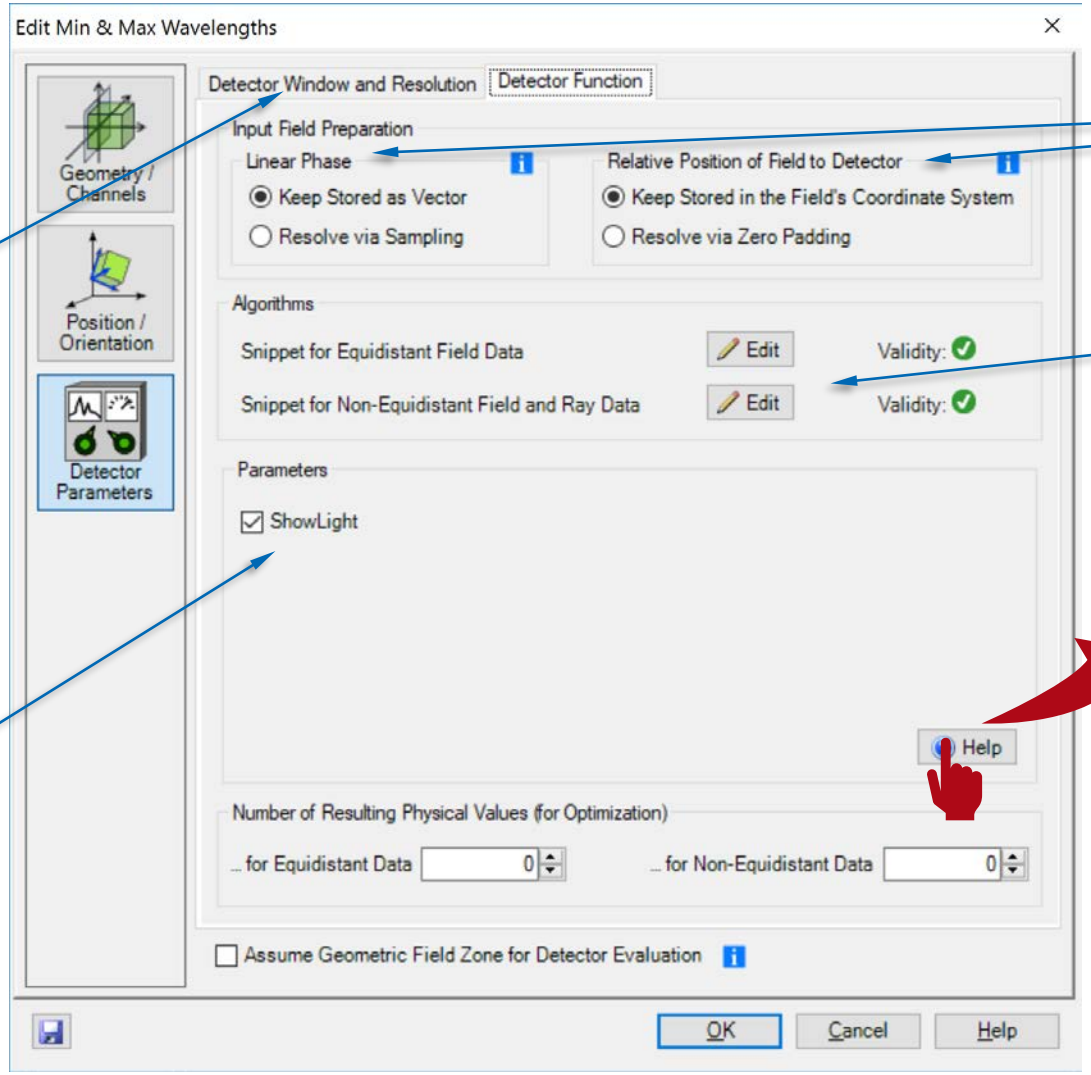
Check Consistency | Validity:

- Variables need to be declared separately and independently in both snippets.
- It would even be possible to use different nomenclature!
- It is the programmer's responsibility to ensure that the code functions in an equivalent manner in both snippets.
- Of all the global parameters (including those defined by the user) only one is snippet-dependent: the one corresponding to light representation (InputField ←→ RayTracing Result)

Programmable Detector: Using Your Snippet

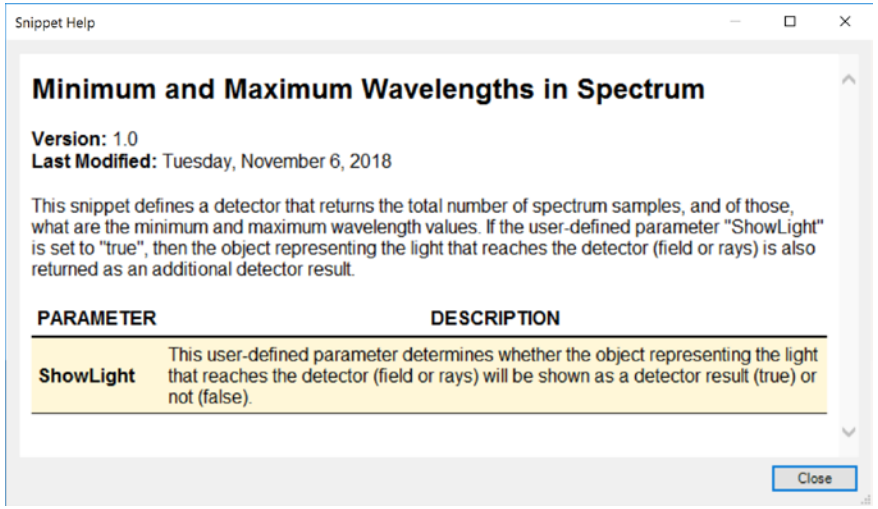
In the Detector Window and Resolution tab you can modify the values of the parameters AutomaticFieldSize, …, ManualNumberSampling Points.

You can modify the value of the global parameters you defined here

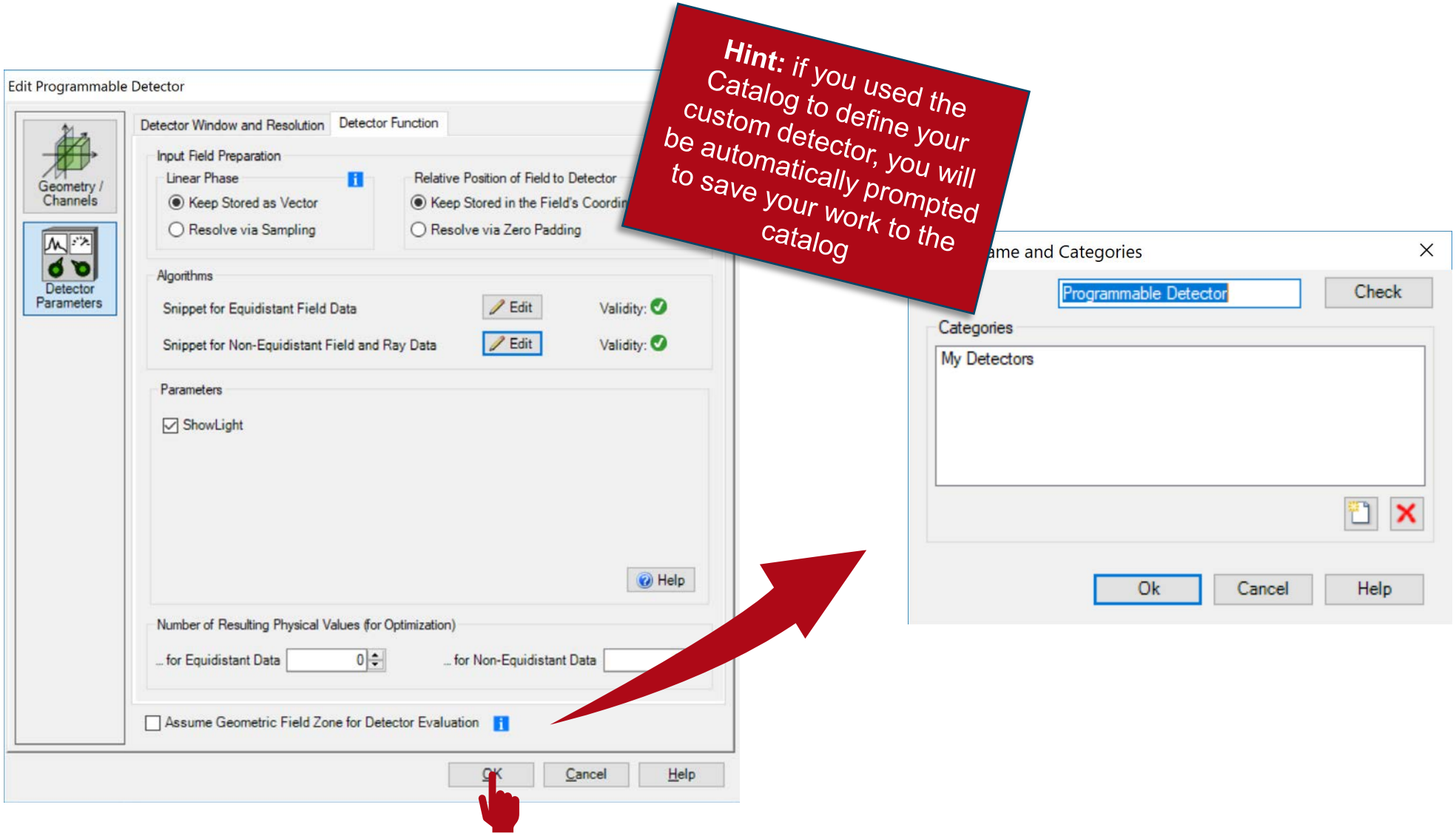


These correspond to the ResolveLinearPhase and ResolveRelativePosition Boolean variables in the snippets. Their value is modified here.

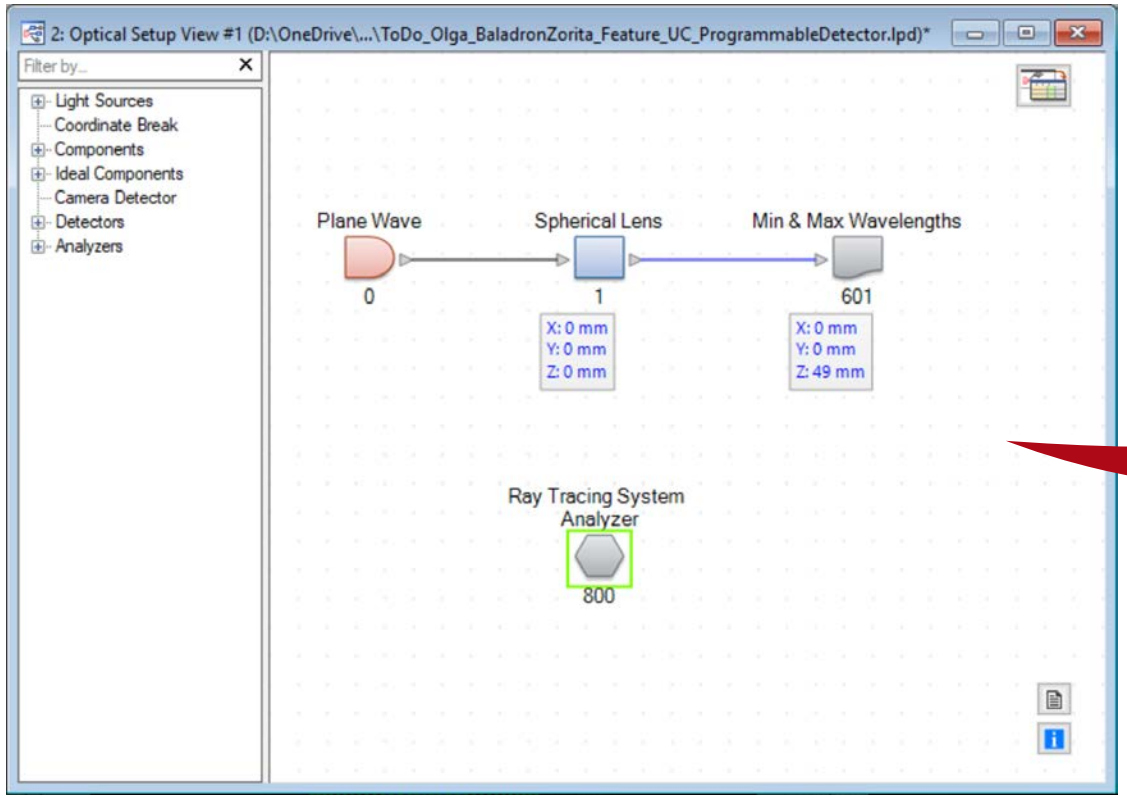
Modify your snippets by clicking on Edit

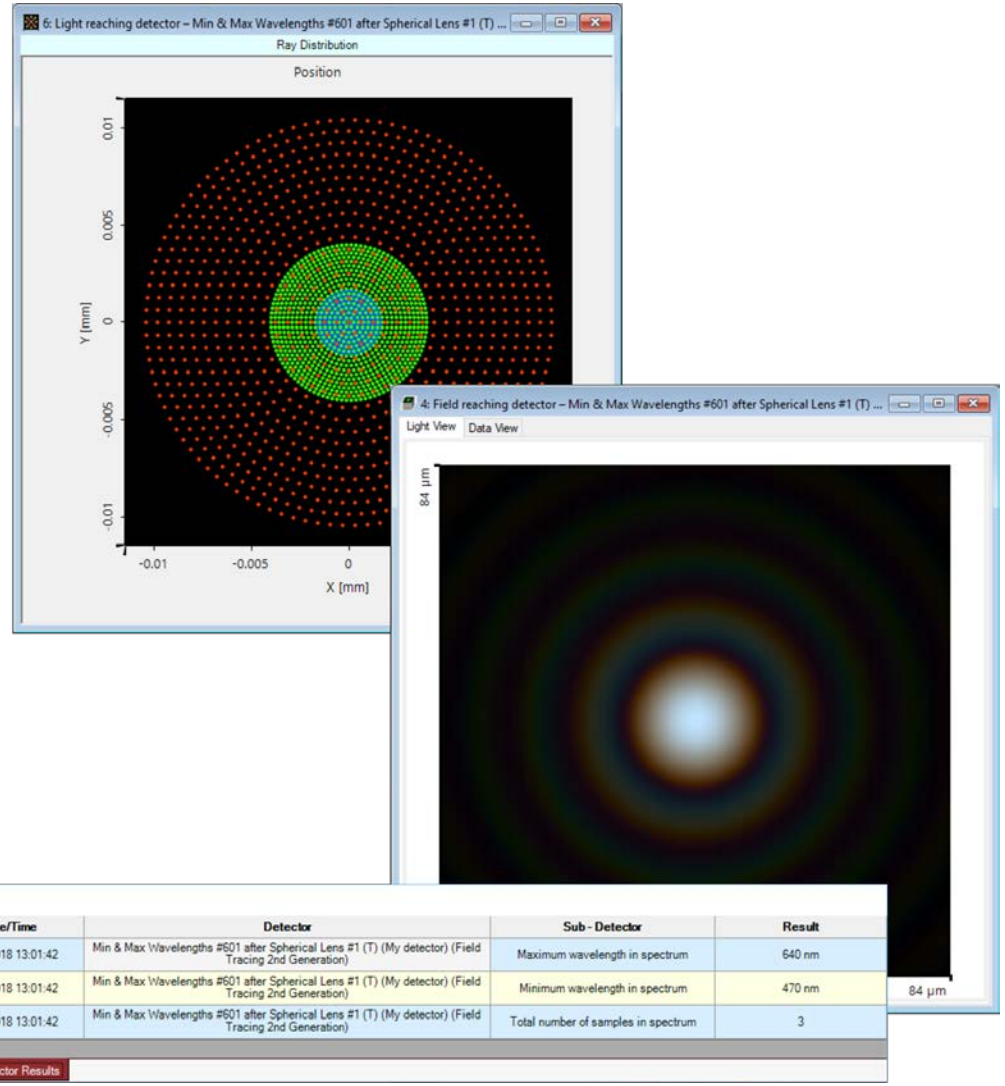


Save the Custom Detector to the Catalog



Output of the Programmable Detector





Test the Code!

Main Function (Equidistant)

```
// Declare and assign an integer parameter that will define the total number of
// results produced by the detector when a simulation is run:
int numberOfResults = 3;

// We modify the total number of results produced by the detector according to
// the value (true or false) of the user-defined Boolean variable that controls
// whether an additional result with the light-representing object (rays or
// fields) will be returned.
if (ShowLight) numberOfResults = 4;

// Declare the object to be returned by the code: an array of DetectorResultObject.
// The size of the array is determined by the variable defined above.
DetectorResultObject[] detectorResults = new DetectorResultObject[numberOfResults];

// A string that gives a name to the Programmable Detector. This name will appear
// alongside the results when the simulation is run.
string detectorName = "My detector";
```

Test the Code!

Main Function (Equidistant)

```
// Obtain one of the magnitudes to be provided  by the detector: the total number
// of samples contained in the spectrum that reaches the detector.
int totalNumberOfSpectrumSamples = InputField.GetSortedListOfWavelength().Count;

// Construct the results:
detectorResults[0] = new DetectorResultObject(
    new PhysicalValue(totalNumberOfSpectrumSamples,
    PhysicalProperty.NoUnit,
    "Total number of samples present in spectrum"),
    detectorName);
detectorResults[1] = new DetectorResultObject(
    new PhysicalValue(InputField.GetSortedListOfWavelength()[0],
    PhysicalProperty.Length,
    "Minimum wavelength in spectrum"),
    detectorName);
detectorResults[2] = new DetectorResultObject(
    new PhysicalValue(InputField.GetSortedListOfWavelength()[totalNumberOfSpectrumSamples - 1],
    PhysicalProperty.Length,
    "Maximum wavelength in spectrum"),
    detectorName);
```

Test the Code!

Main Function (Equidistant)

```
// Conditionally include the last result:
if (ShowLight)
{
    detectorResults[3] = new DetectorResultObject(
        InputField,
        detectorName,
        "Field reaching detector");
}

return detectorResults;
```

Test the Code!

Main Function (Rays and Non-Equidistant)

```
// Declare and assign an integer parameter that will define the total number of
// results produced by the detector when a simulation is run:
int numberOfResults = 3;

// We modify the total number of results produced by the detector according to
// the value (true or false) of the user-defined Boolean variable that controls
// whether an additional result with the light-representing object (rays or
// fields) will be returned.
if (ShowLight) numberOfResults = 4;

// Declare the object to be returned by the code: an array of DetectorResultObject.
// The size of the array is determined by the variable defined above.
DetectorResultObject[] detectorResults = new DetectorResultObject[numberOfResults];

// A string that gives a name to the Programmable Detector. This name will appear
// alongside the results when the simulation is run.
string detectorName = "My detector";
```

Test the Code!

Main Function (Rays and Non-Equidistant)

```
// Obtain one of the magnitudes to be provided  by the detector: the total number
// of samples contained in the spectrum that reaches the detector.
int totalNumberOfSpectrumSamples = RayTracingResult.NumberOfWavelengths;

// Construct the results:
detectorResults[0] = new DetectorResultObject(
    new PhysicalValue(totalNumberOfSpectrumSamples,
    PhysicalProperty.NoUnit,
    "Total number of samples in spectrum"),
    detectorName);
detectorResults[1] = new DetectorResultObject(
    new PhysicalValue(RayTracingResult.GetWavelengthForIndex(0),
    PhysicalProperty.Length,
    "Minimum wavelength in spectrum"),
    detectorName);
detectorResults[2] = new DetectorResultObject(
    new PhysicalValue(RayTracingResult.GetWavelengthForIndex(totalNumberOfSpectrumSamples - 1),
    PhysicalProperty.Length,
    "Maximum wavelength in spectrum"),
    detectorName);
```

Test the Code!

Main Function (Rays and Non-Equidistant)

```
// Conditionally include the last result:
if (ShowLight)
{
    detectorResults[3] = new DetectorResultObject(
        RayTracingResult,
        detectorName,
        "Light reaching detector");
}

return detectorResults;
```

Document Information

title	How to Work with the Programmable Detector and Example (Minimum and Maximum Wavelengths)
document code	CZT.0098
version	1.0
toolbox(es)	Starter Toolbox
VL version used for simulations	7.4.0.49
category	Feature Use Case
further reading	- Programming a Degree of Coherence Detector - Programming a Detector for Diffractive Optics Merit Functions Calculation