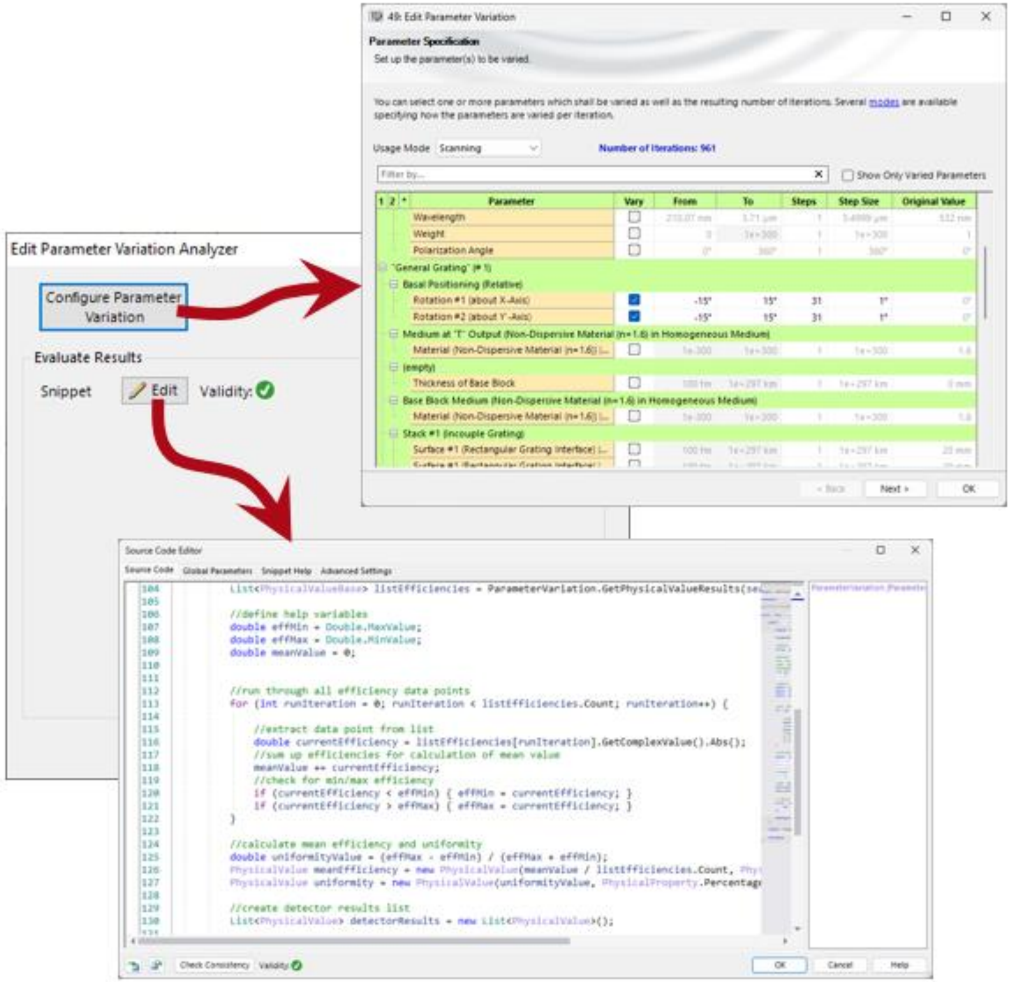
Parameter Variation Analyzer

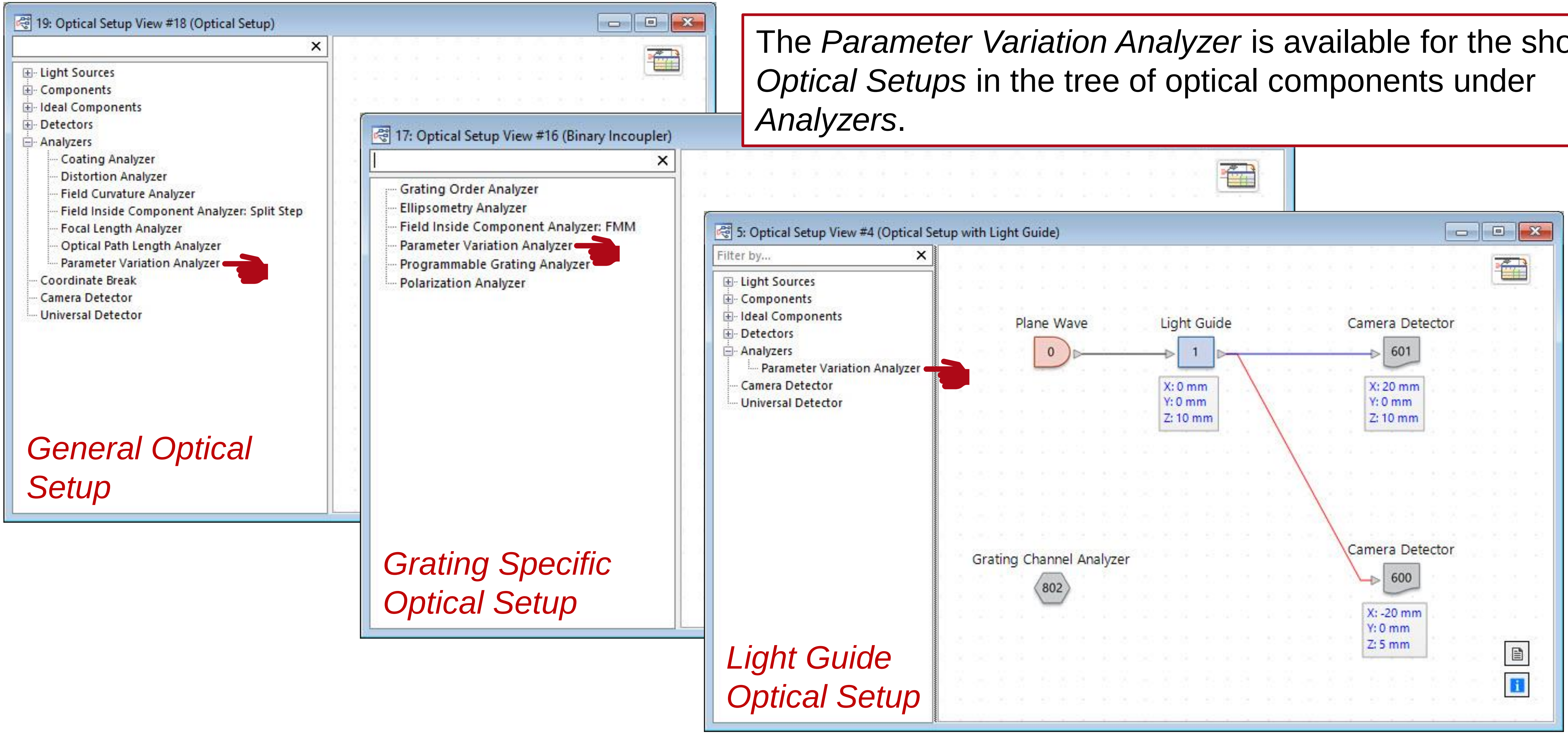
Abstract



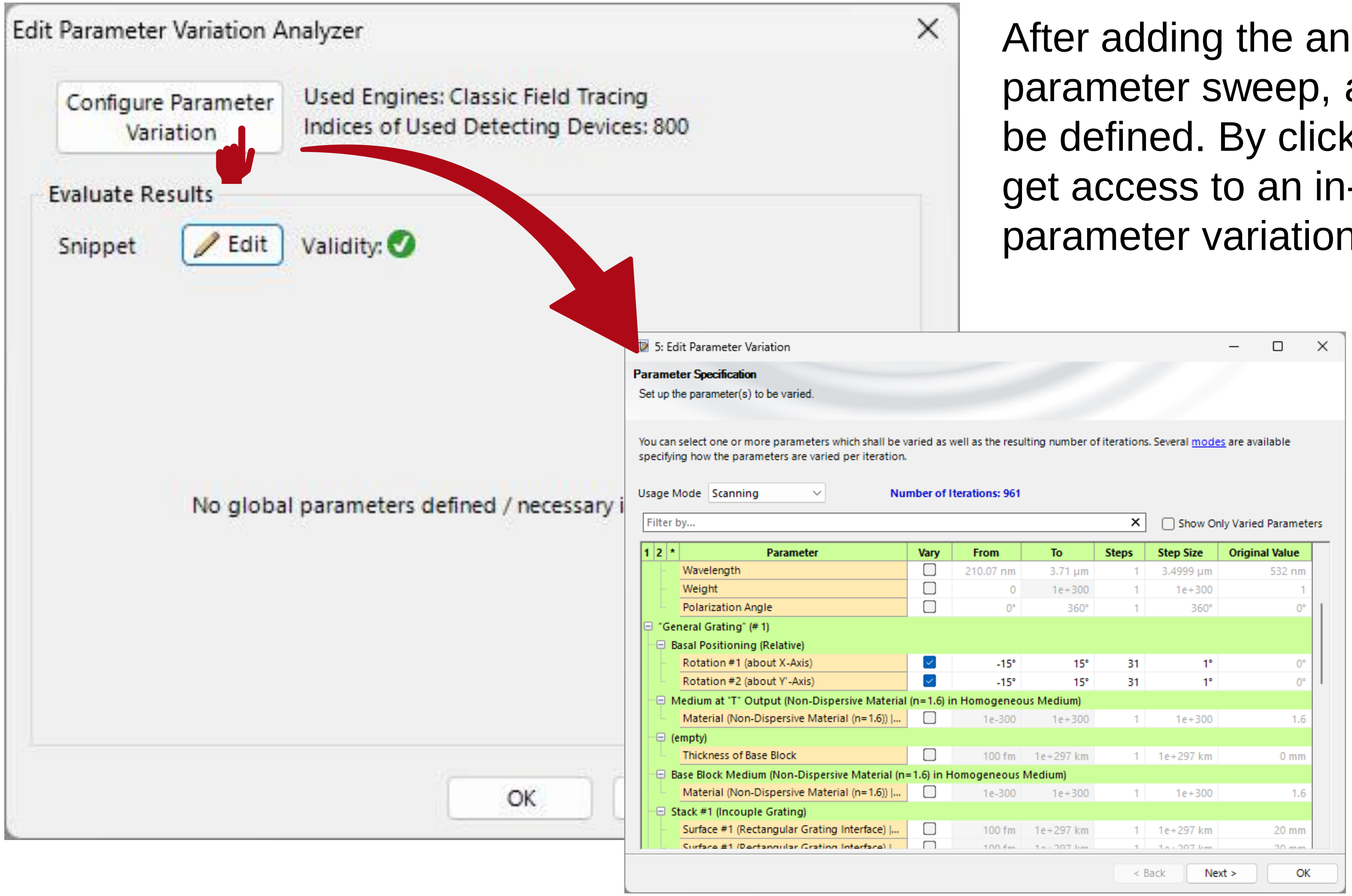
In the process of designing, optimizing and tolerancing complex optical systems it is often desirable to analyze characteristics for a set of different system parameters, not just a single configuration. Parameter Runs are the designated tool to sweep system parameters in a desired parameters space. But it does not allow to define and evaluate merit functions from the individual results that can be further processed. The new Parameter Variation Analyzer is the right tool to close this gap. With this analyzer you can basically analyze the entire system and further process the data obtained. This is very useful, among other things, when a large amount of data is generated, but the evaluation requires well-defined quality functions, which are then used in a next step of the analysis or optimization.

Where to Find the Parameter Variation Analyzer?

The *Parameter Variation Analyzer* is available for the shown *Optical Setups* in the tree of optical components under *Analyzers*.



Defining the Parameter Variation



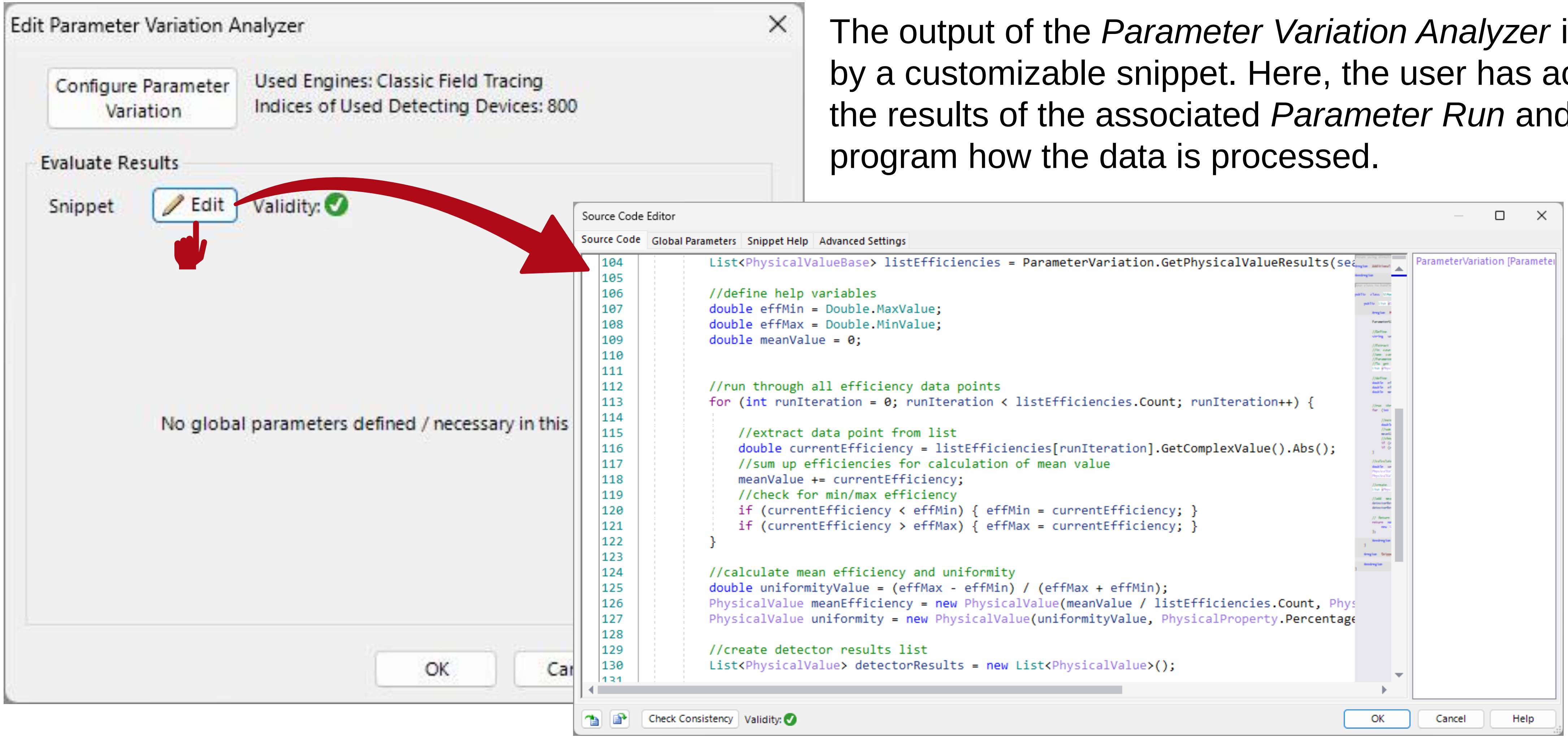
After adding the analyzer to the optical system, the parameter sweep, and the evaluation of the results must be defined. By clicking *Configure Parameter Variation you* get access to an in-built *Parameter Run* document, where parameter variation can be configured.

For a detailed introduction on how to operate the *Parameter Run* document, please see:

[Usage of the Parameter Run Document]

Evaluation of the Results

The output of the *Parameter Variation Analyzer* is defined by a customizable snippet. Here, the user has access to the results of the associated *Parameter Run* and needs to program how the data is processed.



Step #1: Extract Results

Parameter Visualization

Preview of used parameter sets per iteration

The table below shows the parameters which will be used in each iteration of the Parameter Run.

	Iteration Step	1	2	3	4	5	6	7	8
1	Rotation #1 (about X-Axis) ("General Grating" (# 1) | Basal Positioning (Relative))	-15°	-15°	-15°	-15°	-15°	-15°	-15°	-15
2	Rotation #2 (about Y'-Axis) ("General Grating" (# 1) | Basal Positioning (Relative))	-15°	-14°	-13°	-12°	-11°	-10°	-9°	-8

The table of the internal parameter run window gives an overview of the defined iterations.

Note that in the result list in the snippet, the numbering of the iterations starts with a zero (i.e. first step).

By default, the snippet to access the data of the associated internal *Parameter Run*, is already preconfigured:

```
string searchString_detectorName = "";        // enter detector name here (either full or parts of the name possible)
string searchString_subDetectorName = "";     // enter sub-detector name here (either full or parts of the name possible)

// Get the first results for the specified detector/sub-detector that fits to the search strings
List<PhysicalValueBase> physicalValues = ParameterVariation.GetPhysicalValueResults(searchString_detectorName, searchString_subDetectorName);
```

Here, the variables are used to search* for matching detectors and subdetectors in the given optical system. While the "DetectorName" refers to detectors or analyzer (e.g. "Universal Detector", "Grating Order Analyzer"), a subdetector (and the regarding "SubDetectorName") often represents a certain output of the detector (e.g. "mean efficiency", "uniformity contrast"). A simulation of the optical system will reveal the proper names in the *Detector Results* panel. For example:

Detector	Sub - Detector	Result
"Beam Parameters" (# 600) (Profile: General)	Diameter X	199.99 µm
	Diameter Y	199.99 µm

**Note: Due to the definition of search strings, in most cases it is not necessary to use the exact name of the (sub)detector.*

Step #2: Output Results

```
string searchString_detectorName = "";       // enter detector name here (either full or parts of the name possible)
string searchString_subDetectorName = "";    // enter sub-detector name here (either full or parts of the name possible)
```

```
// Get the first results for the specified detector/sub-detector that fits to the search strings
List<PhysicalValueBase> physicalValues = ParameterVariation.GetPhysicalValueResults(searchString_detectorName, searchString_subDetectorName);
```

The function generates a list of the defined values provided by the detector and its subdetector for each iteration of the parameter run.

With the values in the list, any further processing can be applied (please see examples in this document).

Finally, the results can be output again, and used for optimization merits or other purposes. The part for the output is also predefined:

```
// Return the list with the new results
return new List<DetectorResultObject>() {
    new DetectorResultObject(physicalValues, "Result")
};
```

This will output the results to the *Detector Results* panel. It is also possible to generate 1D or 2D graphs to visualize the data.

Technical Insight – Visualization of the Results

To generate a visualization of the results, 1D or 2D *Data Arrays* can be generated. For this purpose, it is necessary to extract information about the number of iterations or results and the ranges of parameter. This can be done by using the following code (in this example sampling distance and start value):

```
//Get Sampling Parameters from Parameter Run
List<VaryParameterData> variedParameters = ParameterRunSupportFunctions.ExtractVariedParameters(ParameterVariation.ParameterData);

double SamplingDistance = variedParameters[0].StepSize.Value;
double FirstDataPoint = variedParameters[0].MinValue;
```

Please note, that in order to use this function the following directive must be added to the “Additional using directives” section of the snippet:

```
#region Additional using directives

using VirtualLabAPI.Core.ParameterRuns;

#endregion
```

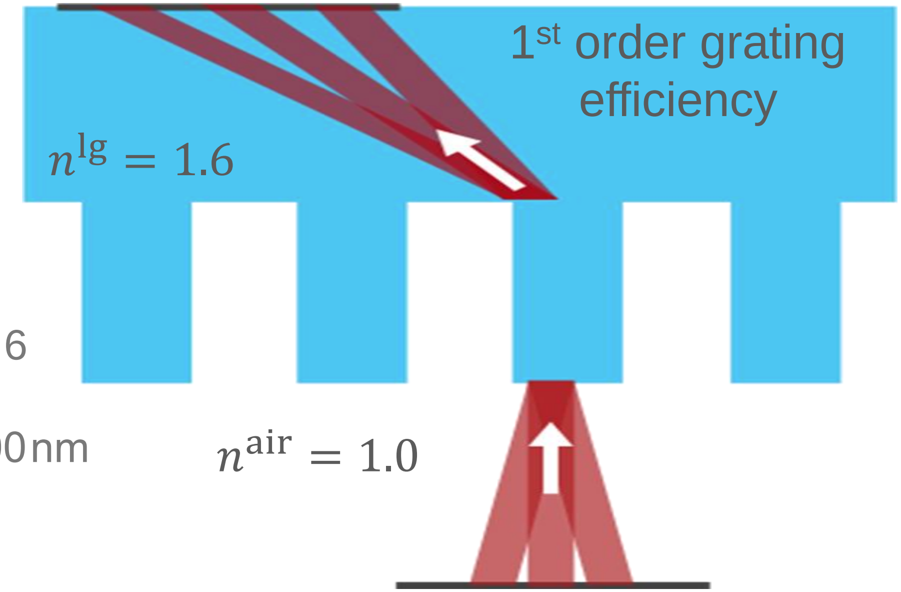
For a full example, please see the corresponding examples of this use case.

Example 1: Calculation of Mean Value and Contrast

Example – Mean Efficiency of a Binary Incoupler Grating

incouple grating

- type: binary transmission grating
- period: 410 nm
- operation order: +1
- medium in front: air
- medium behind: n=1.6
- fill factor: 50%
- modulation depth: 400 nm



field of view:

- set of plane waves: -15°..15° along x-axis & y-axis (*)
- wavelength: 532 nm
- polarization: linear along x-axis

(*) *Internally the different FOVs are modeled by tilting the grating accordingly.*

detector evaluations

$$\text{mean efficiency} = \frac{\Sigma \text{ efficiency per FOV}}{\text{number of FOVs}}$$

$$\text{uniformity contrast} = \frac{\text{max. efficiency} - \text{min. efficiency}}{\text{max. efficiency} + \text{min. efficiency}}$$

Parameter Variation Analyzer result

Detector	Sub - Detector	Result
"Parameter Variation Analyzer" (# 801) (Result)	mean efficiency	11.107 %
	uniformity contrast	90.872 %

Code in the Parameter Variation Analyzer (Incoupler Grating Example)

```
ParameterVariation.StartParameterRun();
```

Command to start the internal Parameter Run (present by default).

```
//Define which Detectors shall be investigated
string searchString_detectorName1 = "Grating Order Analyzer";

//Extract information from the various detectors
//in case a detector outputs multiple results and only one of them is needed here,
//one can identify the desired one by specifying a second searchSting in the command below
//ParameterVariation.GetPhysicalValueResults("stringDetectorName", "stringSubDetectorName");
//To get the subdetector name, please simulate the OS once and check the desired name in the detector panel.
List<PhysicalValueBase> listEfficiencies = ParameterVariation.GetPhysicalValueResults(searchString_detectorName1, "");
```

Here, search strings for the detector name and value (subdetector name) are defined. In this example, it is the efficiency of the 1st order in transmission provided by the Grating Order Analyzer.

```
//define help variables
double effMin = Double.MaxValue;
double effMax = Double.MinValue;
double meanValue = 0;


//run through all efficiency data points
for (int runIteration = 0; runIteration < listEfficiencies.Count; runIteration++) {

    //extract data point from list
    double currentEfficiency = listEfficiencies[runIteration].GetComplexValue().Abs();
    //sum up efficiencies for calculation of mean value
    meanValue += currentEfficiency;
    //check for min/max efficiency
    if (currentEfficiency < effMin) { effMin = currentEfficiency; }
    if (currentEfficiency > effMax) { effMax = currentEfficiency; }
}
```

This loop sums up the individual efficiencies and determines the minimum and maximum values.

```
//calculate mean efficiency and uniformity
double uniformityValue = (effMax - effMin) / (effMax + effMin);
PhysicalValue meanEfficiency = new PhysicalValue(meanValue / listEfficiencies.Count, PhysicalProperty.Percentage, "mean efficiency");
PhysicalValue uniformity = new PhysicalValue(uniformityValue, PhysicalProperty.Percentage, "uniformity contrast");
```

The mean efficiency and uniformity contrast are calculated according to the formulas from the previous page.

```
//create detector results list
List<PhysicalValue> detectorResults = new List<PhysicalValue>();

//add mean efficiency and uniformity to detector results
detectorResults.Add(meanEfficiency);
detectorResults.Add(uniformity);

// Return the list with the new results
return new List<DetectorResultObject>() {
    new DetectorResultObject(detectorResults, "Result")
};
```

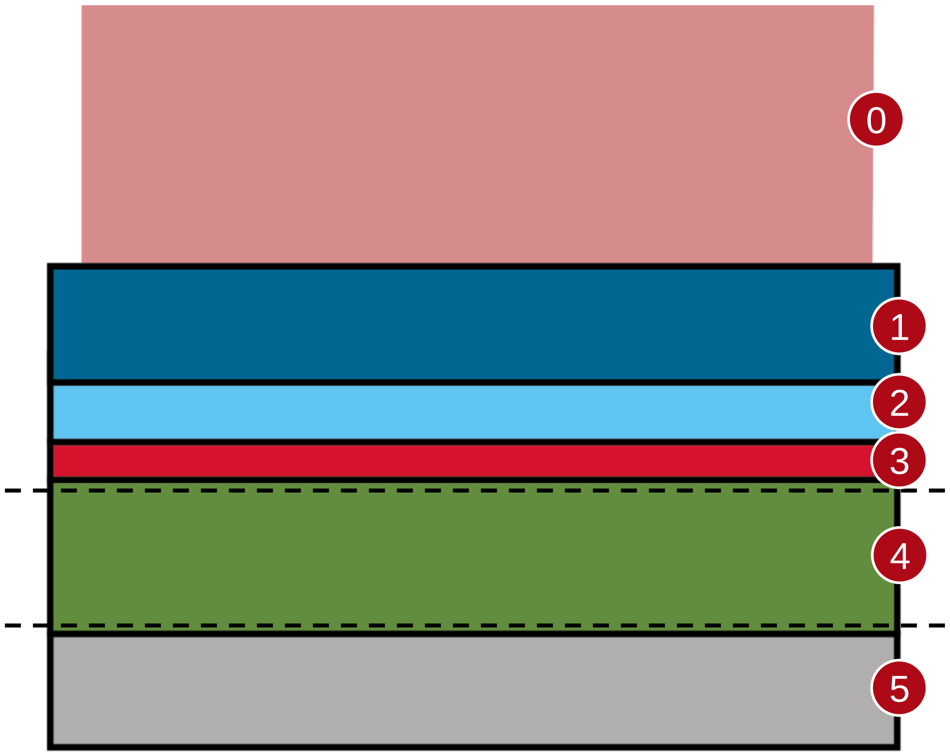
We recommend using a single list of physical quantities as output of a snippet. Hence, the two parameters are combined into a single detector output.

Example 2: Calculation of Detailed Results

Example – Absorption in a CIGS Solar Cell

plane wave

homogeneous spectrum from 300 nm to 1100 nm



detectors

Radiant Flux (absorbed energy is calculated as the difference between the fluxes at the boundaries of layer 4).

For detailed information please see:
Absorption in a CIGS Solar Cell

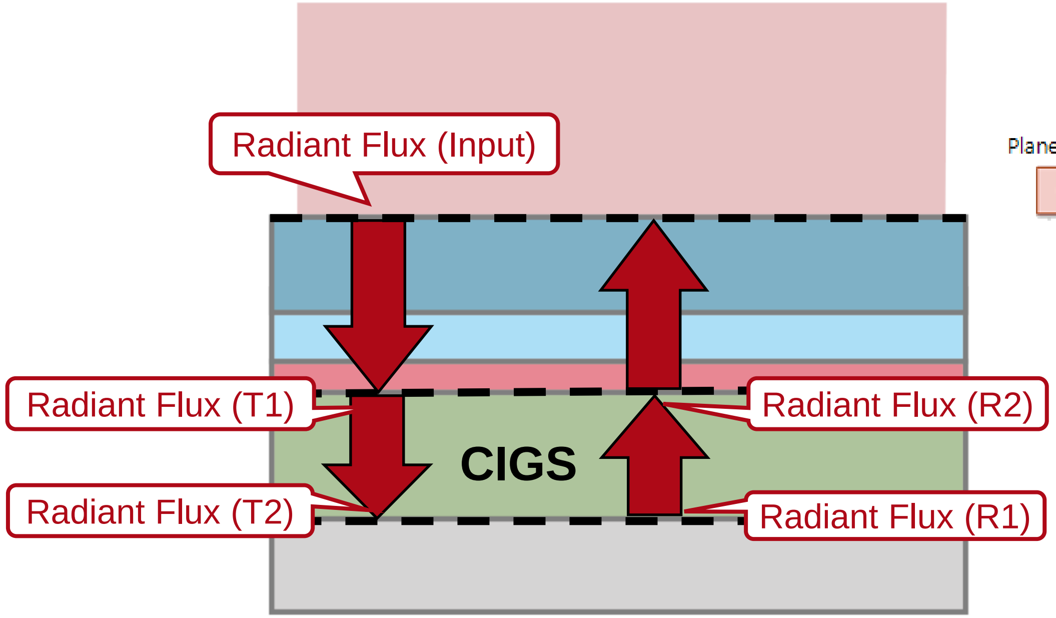
solar cell

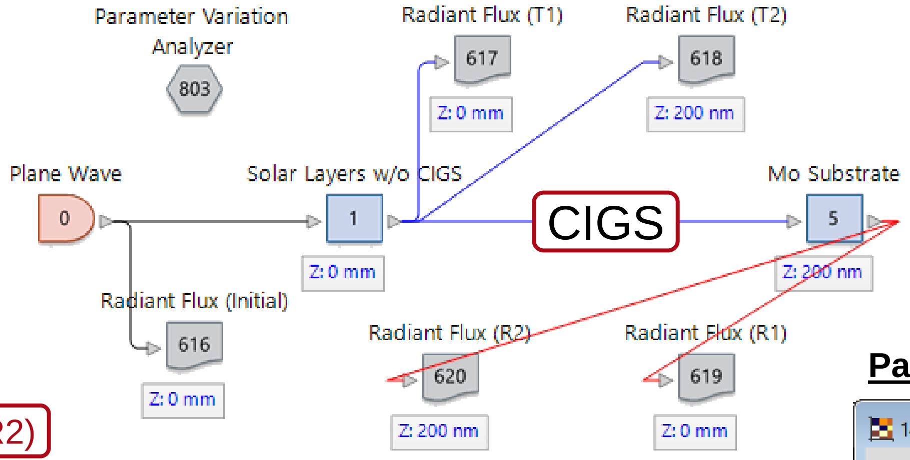
no.	Material	thickness
0	fused silica*	-
1	ZnO:Al	100 nm
2	i-ZnO	70 nm
3	ZnS	50 nm
4	CIGS	100/150/200 nm
5	molybdenum	substrate

*We assume that the solar cell is protected by a layer of fused silica with anti-reflection coating.

System from: *J. Goffard et al., "Light Trapping in Ultrathin CIGS Solar Cells with Nanostructured Back Mirrors," in IEEE Journal of Photovoltaics, vol. 7, no. 5, pp. 1433-1441, Sept. 2017, doi: 10.1109/JPHOTOV.2017.2726566.*

Absorption in a CIGS Solar Cell – Detection Principle

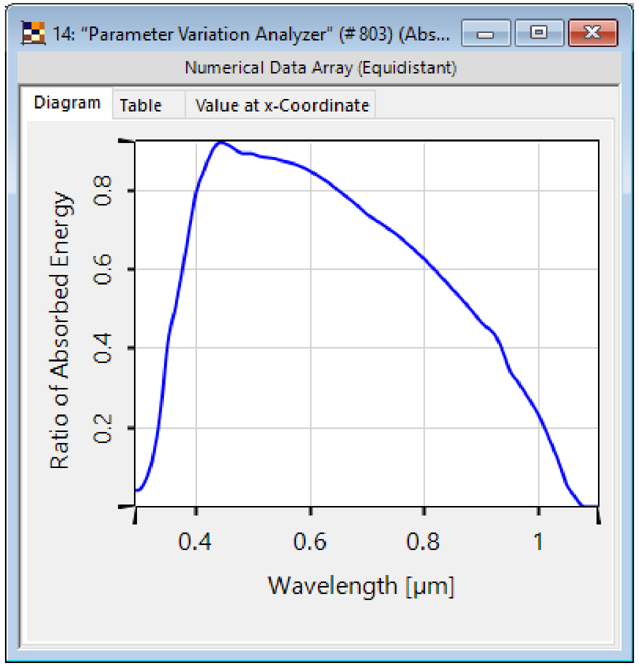




applied formula

$$\begin{matrix}\text{absorbed} \\ \text{energy ratio}\end{matrix} = \frac{T1 - T2 + R1 - R2}{\text{Input}}$$

Parameter Variation Analyzer Result



The absorbed energy inside the CIGS layer per wavelength is determined by adding/subtracting the values of the radiant flux from 4 different detectors:

- at the begin of CIGS layer: **T**ransmitted part (T1) and **R**eflected part (R2)
- at the end of CIGS layer: **T**ransmitted part (T2) and **R**eflected part (R1)

With the *Parameter Variation Analyzer,* the subtraction can be done automatically, outputting the resulting absorption curve by a single simulation.

Code in the Parameter Variation Analyzer (CIGS Absorption Example)

```
ParameterVariation.StartParameterRun();
```

Command to start the internal Parameter Run (present by default).

```
//Define which Detectors shall be investigated
string searchString_detectorNameT1 = "Radiant Flux (T1)";
string searchString_detectorNameT2 = "Radiant Flux (T2)";
string searchString_detectorNameR1 = "Radiant Flux (R1)";
string searchString_detectorNameR2 = "Radiant Flux (R2)";
string searchString_detectorNameIn = "Radiant Flux (Input)";


//Extract information from the various detectors
//in case a detector outputs multiple results and only one of them is needed here,
//one can identify the desired one by specifying a second searchSting in the command below
//ParameterVariation.GetPhysicalValueResults("stringDetectorName", "stringSubDetectorName");
//To get the subdetector name, please simulate the OS once and check the desired name in the detector panel.
List<PhysicalValueBase> lsRadiantFluxResults_T1 = ParameterVariation.GetPhysicalValueResults(searchString_detectorNameT1, "");
List<PhysicalValueBase> lsRadiantFluxResults_T2 = ParameterVariation.GetPhysicalValueResults(searchString_detectorNameT2, "");
List<PhysicalValueBase> lsRadiantFluxResults_R1 = ParameterVariation.GetPhysicalValueResults(searchString_detectorNameR1, "");
List<PhysicalValueBase> lsRadiantFluxResults_R2 = ParameterVariation.GetPhysicalValueResults(searchString_detectorNameR2, "");
List<PhysicalValueBase> lsRadiantFluxResults_In = ParameterVariation.GetPhysicalValueResults(searchString_detectorNameIn, "");
```

Here, search strings for the detector name and value (subdetector name) are defined. In this example, these are the outputs (radiant flux) of the 5 detectors in the system.

```
//Get Sampling Parameters from Parameter Run
List<VaryParameterData> variedParameters = ParameterRunSupportFunctions.ExtractVariedParameters(ParameterVariation.ParameterData);

double samplingDistance = variedParameters[0].StepSize.Value;
double firstDataCoordinate = variedParameters[0].MinValue;
```

This section is used to extract the sampling parameters from the internal Parameter Run, which are needed to generate a 1D Data Array. Please note, the following directive must be added to the “Additional using directives” section:

```
#region Additional using directives

using VirtualLabAPI.Core.ParameterRuns;

#endregion
```

```
//Perform calculations
double[] arrAbsorption = new double[lsRadiantFluxResults_T1.Count];

double fluxT1, fluxT2, fluxR1, fluxR2, fluxIn, absorption;
for (int runIndex = 0; runIndex < lsRadiantFluxResults_T1.Count; runIndex++) {
    fluxT1 = lsRadiantFluxResults_T1[runIndex].GetComplexValue().Abs();
    fluxT2 = lsRadiantFluxResults_T2[runIndex].GetComplexValue().Abs();
    fluxR1 = lsRadiantFluxResults_R1[runIndex].GetComplexValue().Abs();
    fluxR2 = lsRadiantFluxResults_R2[runIndex].GetComplexValue().Abs();
    fluxIn = lsRadiantFluxResults_In[runIndex].GetComplexValue().Abs();

    absorption = (fluxT1 - fluxT2 + fluxR1 - fluxR2) / fluxIn;
    arrAbsorption[runIndex] = absorption;
}
```

The absorbed energy in the CIGS layer is calculated per wavelength by subtracting the transmitted and reflected fluxes from the initial value.

```
//Generate 1D Data Array
DataArray1D da1d_absorptionCurve = new DataArray1D(data: arrAbsorption,
                                                   physicalPropertyOfData: PhysicalProperty.NoUnit,
                                                   commentOfData: "Ratio of Absorbed Energy",
                                                   samplingDistance: samplingDistance,
                                                   coordinateOfFirstDataPoint: firstDataCoordinate,
                                                   physicalPropertyOfCoordinates: PhysicalProperty.Length,
                                                   commentOfCoordinates: "Wavelength");

// Return the list with the new results
return new List<DetectorResultObject>() {
    new DetectorResultObject(da1d_absorptionCurve, "Absorption Curve")
};
```

To illustrate the result a 1D Data Array is generated, which shows the amount of absorbed energy over wavelengths, which is then output.

Document Information

title	Parameter Variation Analyzer
document code	SWF.0046
document version	1.1
required packages	- (for the Analyzer); Grating Package (for incoupling grating example)
software version	2024.1 (Build 1.132)
category	Feature Use Case
further reading	- Usage of the Parameter Run Document - Absorption in a CIGS Solar Cell