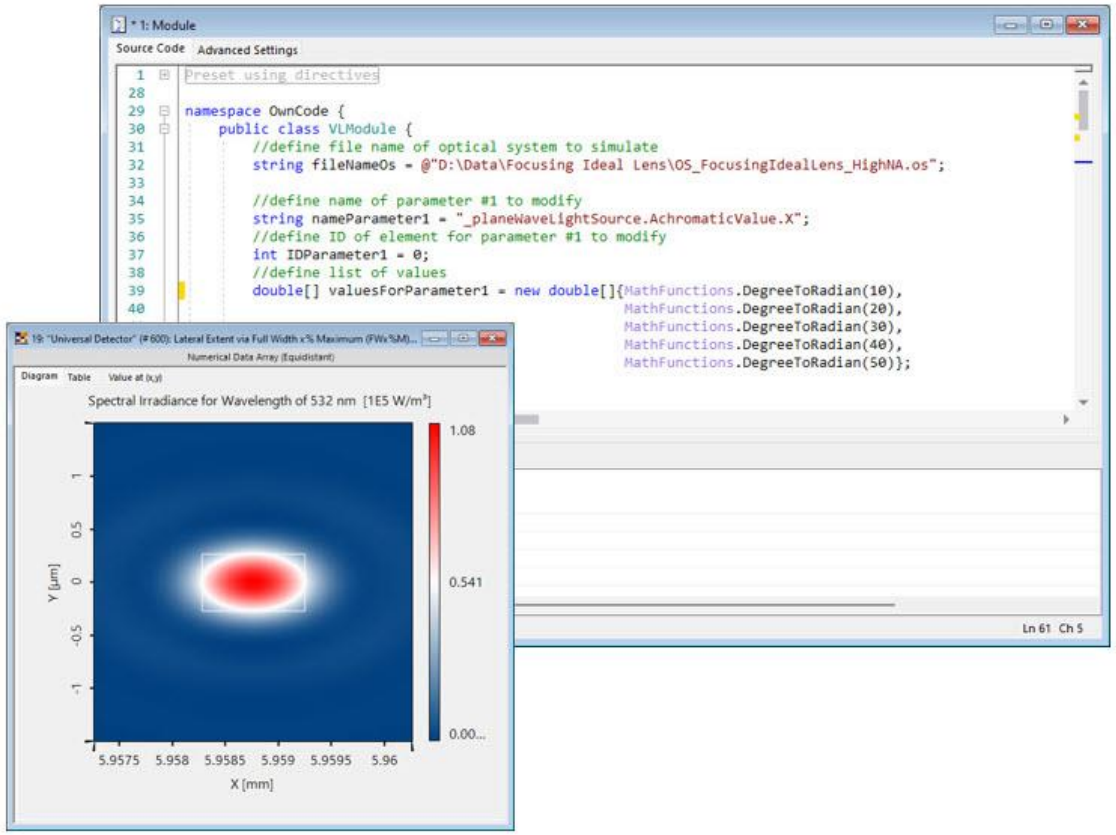
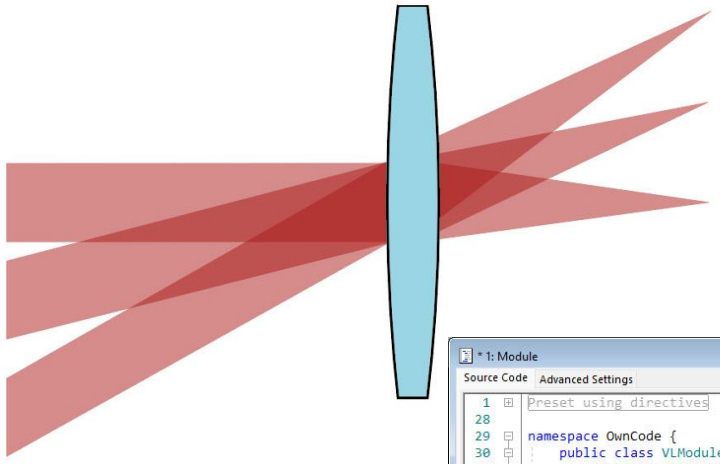
Simulation of a High NA Focusing System via Module

Abstract

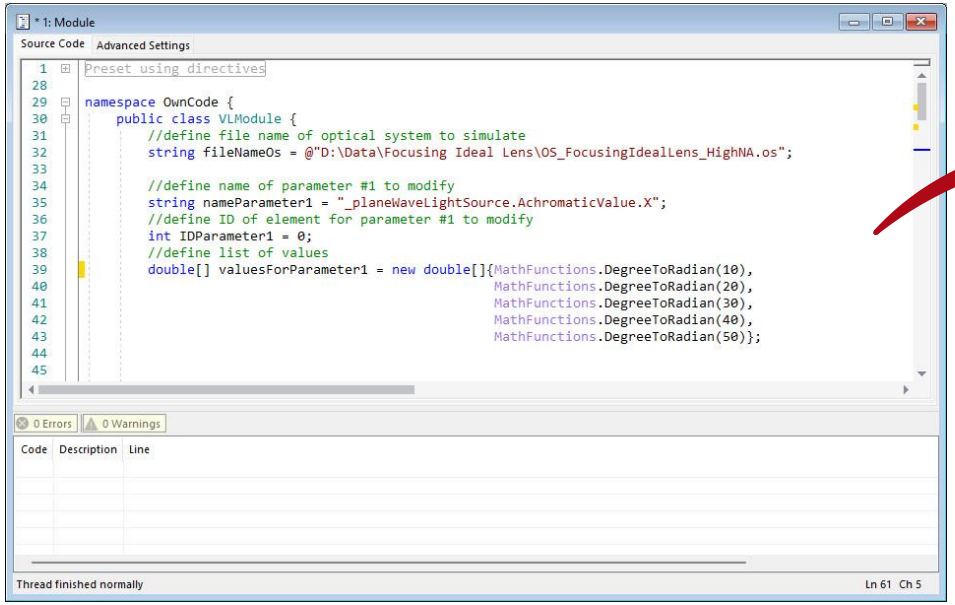


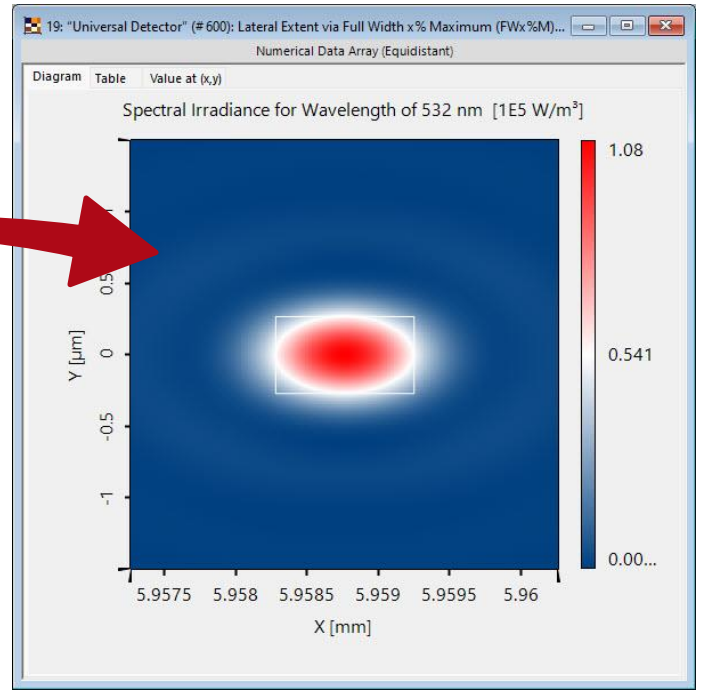
VirtualLab Fusion offers the user the ability to call its solvers from other environments (e.g. python, …) and using XML-files to specify all necessary simulation parameters. In this use case, we demonstrate how to perform a simulation of a high-NA focusing system via a module and adapt parameters of the preset optical system.

This Use Case Shows ….



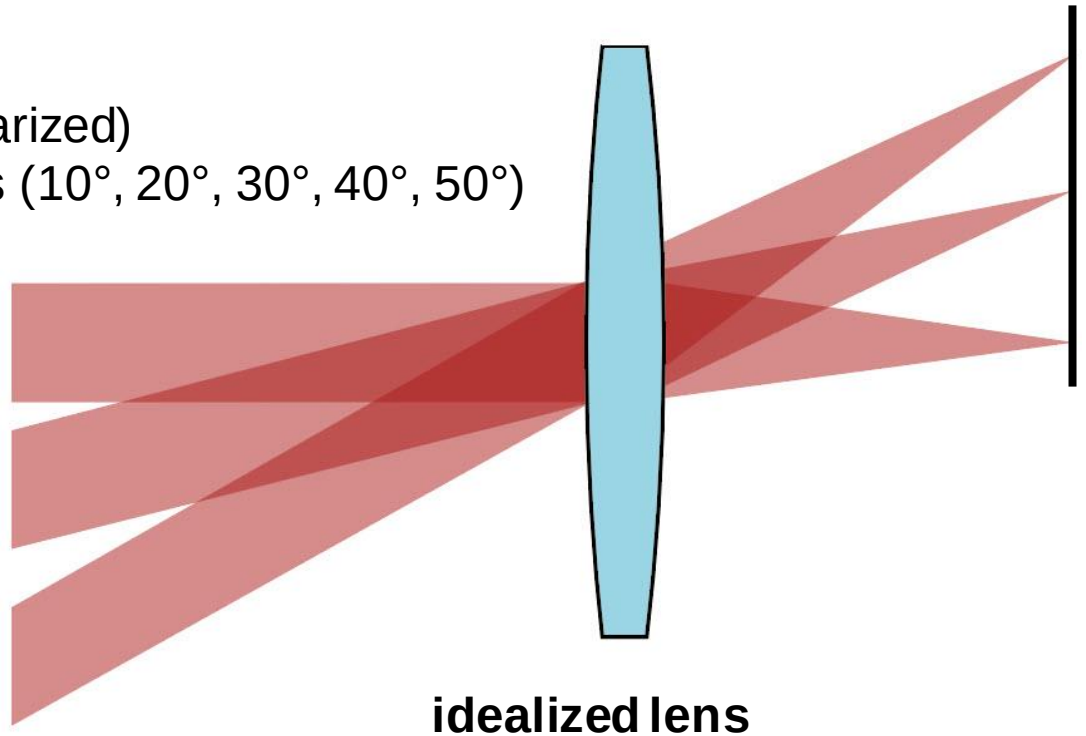
How to use a programmable module to change simulation parameter and perform an optical simulation.





Optical System

source (Plane Wave)
- wavelength: 532 nm
- polarization: linear (x-polarized)
- 5 different incident angles (10°, 20°, 30°, 40°, 50°)



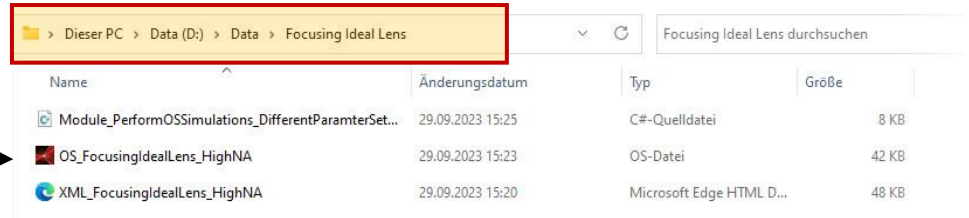
detectors
- irradiance
- FWHM

idealized lens
- focal length: 5 mm

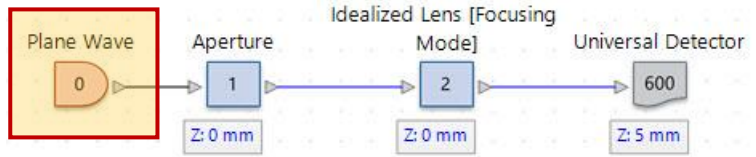
The Module – Preparations I

```
namespace OwnCode {
    public class VLModule {
        //define file name of optical system to simulate
        string fileNameOs = @"D:\Data\Focusing Ideal Lens\OS_FocusingIdealLens_HighNA.os";

        //define name of parameter #1 to modify
        string nameParameter1 = "_planeWaveLightSource.AchromaticValue.X";
        //define ID of element for parameter #1 to modify
        int IDParameter1 = 0;
        //define list of values
        double[] valuesForParameter1 = new double[]{MathFunctions.DegreeToRadian(10),
                                                    MathFunctions.DegreeToRadian(20),
                                                    MathFunctions.DegreeToRadian(30),
                                                    MathFunctions.DegreeToRadian(40),
                                                    MathFunctions.DegreeToRadian(50)};


        //define name of detector to use (can be only part of the full detector name)
        string nameDetectorToUse = "600";
        //define name of sub-detector to use (can be only part of the full detector name)
        //string nameSubDetectorToUse = "Size X";
        string nameSubDetectorToUse = "Original Data";


        //run method of the module
        public void Run() {
            //load optical setup
            Lightpath osToUseForSimulation = Lightpath.Load(fileNameOs);
            //error handling
            if (osToUseForSimulation == null) {
                Globals.DataDisplay.LogError("Optical Setup could not be loaded");
                return;
            }

            //define list for numerical results
            List<PhysicalValue> listNumericalValuesFromSimulations = new List<PhysicalValue>();

            //loop over all parameters to modify
            for (int runAllParameters = 0; runAllParameters < valuesForParameter1.Length; runAllParameters++) {
                //set current parameter to load optical setup
                LightPathExportImportSupport.ChangeParameterFromExtern(ref osToUseForSimulation,
                                                                       this.IDParameter1,
                                                                       nameParameter1,
                                                                       valuesForParameter1[runAllParameters]);


                //generate simulation wrapper
                LightPathDiagramWrapper osWrapper = new LightPathDiagramWrapper(osToUseForSimulation, false);
                //perform simuation
                osWrapper.Perform();

                //get results from result matrix
                List<DetectorResultObject> simulationResults = osWrapper.SimulationResults;

                //define list of results that match to the detector name
                List<DetectorResultObject> simulationResult_MatchDetector = new List<DetectorResultObject>();
                //loop run through all simulation results
```

An initial *Optical Setup* file is used as an initial starting point. Here, the path of the OS-file needs to be defined.



The ID Parameter is used to identify the components in the module and to be able to apply changes. In this case, the parameter of the source (ID 0) are configured.



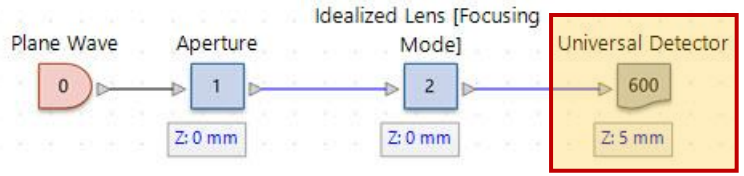
In this section, a list of (double) values is defined, which is going to be used later. Here, the list contains five different angles, which will be used to define different angles of incidence.

The Module – Preparations II

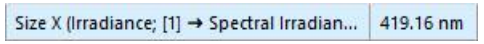
```
namespace OwnCode {
    public class VLModule {
        //define file name of optical system to simulate
        string fileNameOs = @"D:\Data\Focusing Ideal Lens\OS_FocusingIdealLens_HighNA.os";

        //define name of parameter #1 to modify
        string nameParameter1 = "_planeWaveLightSource.AchromaticValue.X";
        //define ID of element for parameter #1 to modify
        int IDParameter1 = 0;
        //define list of values
        double[] valuesForParameter1 = new double[]{MathFunctions.DegreeToRadian(10),
                                                    MathFunctions.DegreeToRadian(20),
                                                    MathFunctions.DegreeToRadian(30),
                                                    MathFunctions.DegreeToRadian(40),
                                                    MathFunctions.DegreeToRadian(50)};


        //define name of detector to use (can be only part of the full detector name)
        string nameDetectorToUse = "600";
        //define name of sub-detector to use (can be only part of the full detector name)
        //string nameSubDetectorToUse = "Size X";
        string nameSubDetectorToUse = "Original Data";


        //run method of the module
        public void Run() {
            //load optical setup
            Lightpath osToUseForSimulation = Lightpath.Load(fileNameOs);
            //error handling
            if (osToUseForSimulation == null) {
                Globals.DataDisplay.LogError("Optical Setup could not be loaded");
                return;
            }

            //define list for numerical results
            List<PhysicalValue> listNumericalValuesFromSimulations = new List<PhysicalValue>();

            //loop over all parameters to modify
            for (int runAllParameters = 0; runAllParameters < valuesForParameter1.Length; runAllParameters++) {
                //set current parameter to load optical setup
                LightPathExportImportSupport.ChangeParameterFromExtern(ref osToUseForSimulation,
                                                                       this.IDParameter1,
                                                                       nameParameter1,
                                                                       valuesForParameter1[runAllParameters]);


                //generate simulation wrapper
                LightPathDiagramWrapper osWrapper = new LightPathDiagramWrapper(osToUseForSimulation, false);
                //perform simuation
                osWrapper.Perform();

                //get results from result matrix
                List<DetectorResultObject> simulationResults = osWrapper.SimulationResults;

                //define list of results that match to the detector name
                List<DetectorResultObject> simulationResult_MatchDetector = new List<DetectorResultObject>();
                //loop run through all simulation results
```

In this section the detector and the detected quantity are defined. In general, VirtualLab Fusion distinguishes between a *Detector* (such as *Universal Detector*, *Camera Detector*, etc.) and a *Subdetector*, which usually represent a specific quantity (*Position*, *Size*, *Beam Waist Diameter*, etc.).

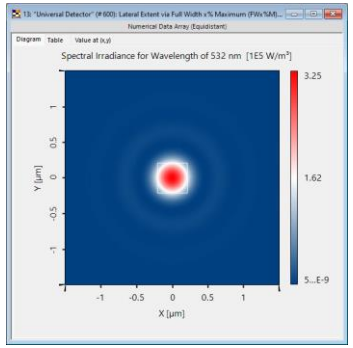
The string variable *nameDetectorToUse* will define the ID of the used detector, in this case the *Universal Detector* with ID 600.



Further, the parameter *nameSubDetectorToUse* specifies the detected quantity. Here, two options are available: either the size of the field or the field (as 2D data array).

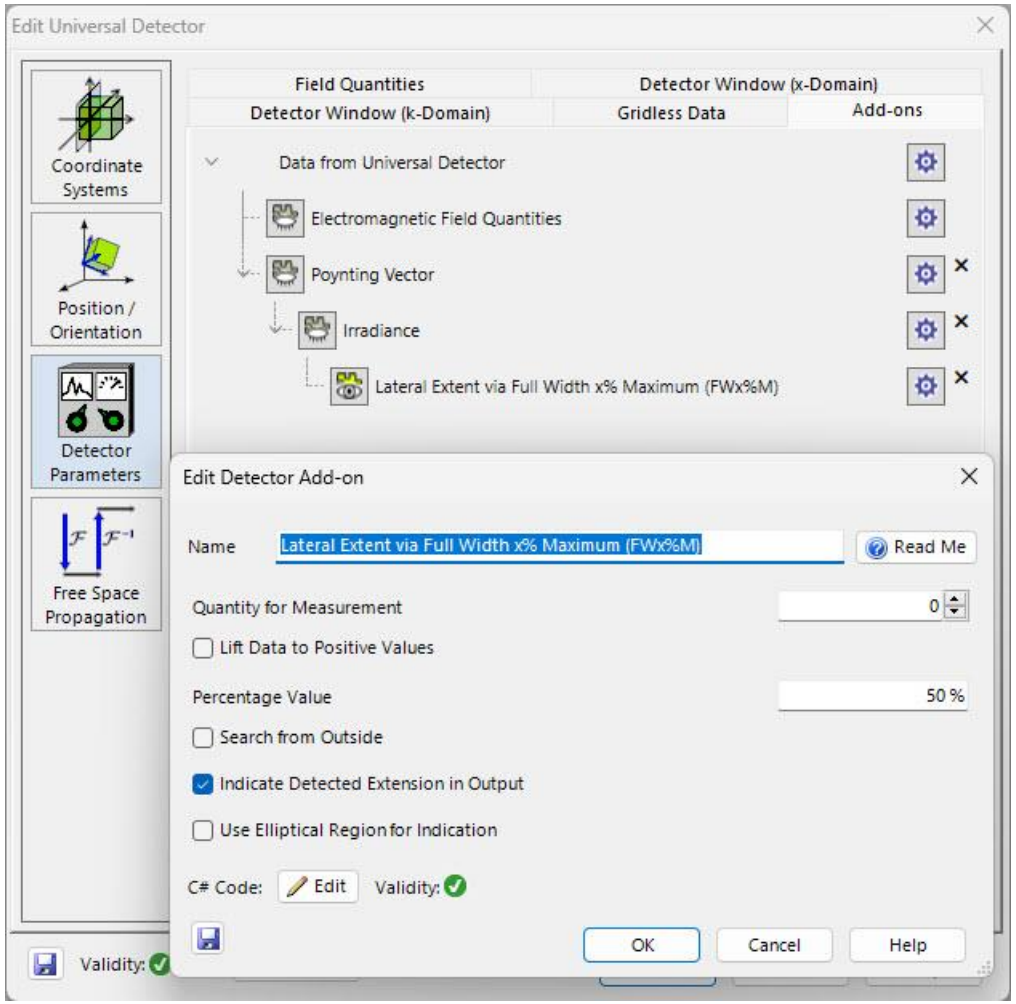


Size X



Original Data

Search Strings for Detector and Subdetector



"Universal Detector" (# 600): Lateral Extent via Full Width x% Maximum (FWx%M) (Profile: General)	Maximum Position X (Irradiance; [1] → Spectral Irradiance for Wavelength of 532 nm)	0 mm
	Maximum Position Y (Irradiance; [1] → Spectral Irradiance for Wavelength of 532 nm)	0 mm
	Center X (Irradiance; [1] → Spectral Irradiance for Wavelength of 532 nm)	-1.0588e-07 fm
	Center Y (Irradiance; [1] → Spectral Irradiance for Wavelength of 532 nm)	-1.0588e-07 fm
	Size X (Irradiance; [1] → Spectral Irradiance for Wavelength of 532 nm)	419.16 nm
	Size Y (Irradiance; [1] → Spectral Irradiance for Wavelength of 532 nm)	419.16 nm

A simulation will reveal the proper detector and subdetector names in the *Detector Results* tab.

The entire string of either the subdetector or the detector is not required, as VirtualLab Fusion uses a search algorithm which identify detectors and subdetectors just by defining parts of the actual name. In the example above, the subdetector named *Size X (Irradiance;[1] -> Spectral Irradiance for Wavelength of 532 nm)* is still found, if the user specifies "Size X" as a string variable.

Be careful, however, because the search in VirtualLab Fusion always returns the first (sub)detector found. In case the user would use only "X" as the string variable, the result would be 0 mm, as the *Maximum Position X* subdetector would be found instead.

The Module – Running the Simulation

```
namespace OwnCode {
    public class VLModule {
        //define file name of optical system to simulate
        string fileNameOs = @"D:\Data\Focusing Ideal Lens\OS_FocusingIdealLens_HighNA.os";

        //define name of parameter #1 to modify
        string nameParameter1 = "_planeWaveLightSource.AchromaticValue.X";
        //define ID of element for parameter #1 to modify
        int IDParameter1 = 0;
        //define list of values
        double[] valuesForParameter1 = new double[]{MathFunctions.DegreeToRadian(10),
                                                    MathFunctions.DegreeToRadian(20),
                                                    MathFunctions.DegreeToRadian(30),
                                                    MathFunctions.DegreeToRadian(40),
                                                    MathFunctions.DegreeToRadian(50)};


        //define name of detector to use (can be only part of the full detector name)
        string nameDetectorToUse = "600";
        //define name of sub-detector to use (can be only part of the full detector name)
        //string nameSubDetectorToUse = "Size X";
        string nameSubDetectorToUse = "Original Data";


        //run method of the module
        public void Run() {
            //load optical setup
            Lightpath osToUseForSimulation = Lightpath.Load(fileNameOs);
            //error handling
            if (osToUseForSimulation == null) {
                Globals.DataDisplay.LogError("Optical Setup could not be loaded");
                return;
            }

            //define list for numerical results
            List<PhysicalValue> listNumericalValuesFromSimulations = new List<PhysicalValue>();

            //loop over all parameters to modify
            for (int runAllParameters = 0; runAllParameters < valuesForParameter1.Length; runAllParameters++) {
                //set current parameter to load optical setup
                LightPathExportImportSupport.ChangeParameterFromExtern(ref osToUseForSimulation,
                                                                       this.IDParameter1,
                                                                       nameParameter1,
                                                                       valuesForParameter1[runAllParameters]);


                //generate simulation wrapper
                LightPathDiagramWrapper osWrapper = new LightPathDiagramWrapper(osToUseForSimulation, false);
                //perform simuation
                osWrapper.Perform();

                //get results from result matrix
                List<DetectorResultObject> simulationResults = osWrapper.SimulationResults;

                //define list of results that match to the detector name
                List<DetectorResultObject> simulationResult_MatchDetector = new List<DetectorResultObject>();
                //loop run through all simulation results
```

Here, the initial optical setup is loaded from file. If no valid OS-file can be found an error message will be thrown.

A container is created, which is filled with the resulting outputs later.

This command adjusts the system the according to the rules we have defined earlier. A loop is used to take care of the number of configurations, which were defined.

Here, the actual simulation is performed (for each set of parameters).

This container now includes all simulation results from the simulation of a single system, meaning from all available detectors and subdetectors. It must be searched to find the desired values.

The Module – Filtering Simulation Results

```
    //define list of results that match to the detector name
    List<DetectorResultObject> simulationResult_MatchDetector = new List<DetectorResultObject>();
    //loop run through all simulation results
    for (int runSimulationResults = 0; runSimulationResults < simulationResults.Count; runSimulationResults++) {
        //check whether detector name matches (is part of) the name of current result
        if (simulationResults[runSimulationResults].Description.Contains(nameDetectorToUse)) {
            //add detector result to list
            simulationResult_MatchDetector.Add(simulationResults[runSimulationResults]);
        }
    }

    //error handling (if no result match with detector name)
    if (simulationResult_MatchDetector.Count == 0) {
        Globals.DataDisplay.LogError("No detector result that match with the detector string found.");
        return;
    }

    //define boolean whether sub detector was found
    bool subdetetorFound = false;
    //loop over all results
    for (int runSimulationResults_SubDetector = 0; runSimulationResults_SubDetector < simulationResult_MatchDetector.Count;
    runSimulationResults_SubDetector++) {
        //get current simulation result
        DetectorResultObject currDetResult = simulationResult_MatchDetector[runSimulationResults_SubDetector];
        //check type of data
        if (currDetResult.Data is IDocument) {
            //in case of IDocument we show the result directly in VLF main window
            Globals.DataDisplay.ShowDocument((currDetResult.Data as IDocument), currDetResult.Description +
            " ==> Module Simulation #" + (runAllParameters + 1).ToString());
            subdetetorFound = true;
        }
        else if (currDetResult.Data is List<PhysicalValue>) {
            //get list of physical value
            List<PhysicalValue> listDetResult = (currDetResult.Data as List<PhysicalValue>);
            //loop over all detector results
            for (int runSubDetectorResults = 0; runSubDetectorResults < listDetResult.Count; runSubDetectorResults++) {
                if (listDetResult[runSubDetectorResults].Comment.Contains(this.nameSubDetectorToUse)) {
                    listNumericalValuesFromSimulations.Add(listDetResult[runSubDetectorResults]);
                    subdetetorFound = true;
                    break;
                }
            }
        }

        if (subdetetorFound) {
            break;
        }
    }
    //plot message that simulation of current parameter set is finished
    Globals.DataDisplay.LogMessage("Simulation of parameter value #" + (runAllParameters + 1).ToString() + " finished.");
}

//check whether numerical values are present
if (listNumericalValuesFromSimulations.Count > 0) {
    //check whether more than one result is present
    if (listNumericalValuesFromSimulations.Count == 1) {
        //log detector result to detector window in VLF main window
```

In this loop, the entire simulation results in the container are searched for quantities that were output by the detector, which was specified in the preparation step. This can be done for all detectors in a specific setup.

This loop is used to search for the specified subdetector.

Subdetector results can be of different type and number. In this section one 2D information is yielded (such as a field data array). It will be outputted by the command *ShowDocument.*

In case of *PhysicalValues* (such as the result for “Size X”), the results are added to the list, that was created at the beginning. In this example, 5 values are added to the list (one for each angle of incidence).

The *LogMessage* command will add a message in the tab to let the user know, which simulation steps are already completed.

The Module – Output for PhysicalValues

```
//check whether numerical values are present
if (listNumericalValuesFromSimulations.Count > 0) {
    //check whether more than one result is present
    if (listNumericalValuesFromSimulations.Count == 1) {
        //log detector result to detector window in VLF main window
        Globals.DataDisplay.LogDetectorResult("Results from Module Execution", listNumericalValuesFromSimulations);
    }
    else {
        //if more than one numerical value given => generate data array

        //generate field for values and coordinates
        CFieldDerivative1DReal cfMeasuredValues = new CFieldDerivative1DReal(this.valuesForParameter1.Length);
        CFieldDerivative1DReal cfCoordinates = new CFieldDerivative1DReal(this.valuesForParameter1.Length);
        //fill data
        for (int runData = 0; runData < this.valuesForParameter1.Length; runData++) {
            //store coordinate
            cfCoordinates[runData] = this.valuesForParameter1[runData];
            //store measured value
            cfMeasuredValues[runData] = listNumericalValuesFromSimulations[runData].Value;
        }

        //generate data array
        DataArray1D daNumericalValuesPerSimulation = new DataArray1D(new ComplexField1DArray(new ComplexField[] { cfMeasuredValues }),
                                                                     new PhysicalProperty[] { listNumericalValuesFromSimulations[0].PhysicalProperty },
                                                                     new string[] { listNumericalValuesFromSimulations[0].Comment },
                                                                     cfCoordinates,
                                                                     PhysicalProperty.AngleDeg,
                                                                     "Incident Angle",
                                                                     this.valuesForParameter1[this.valuesForParameter1.Length - 1] + 1e-5);

        Globals.DataDisplay.ShowDocument(daNumericalValuesPerSimulation, "Numerical Result Generated by Simulation Module");
    }
}
```

As mentioned previously, *PhysicalValue* results are collected in a list. If the list at the end of the module contains only one entry, it will be output on the Detector Results tab with a single entry.

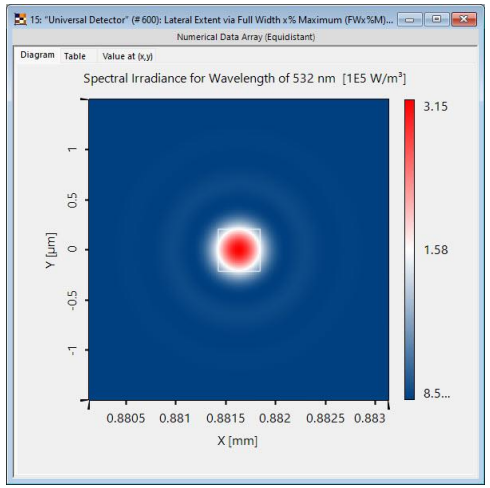
If multiple entries are found, the list is converted to another data format that forms the basis for a 1D Data Array.

In these lines, the a new 1D Data Array is constructed using certain sampling parameters and units. It is displayed by using the *ShowDocument* command.

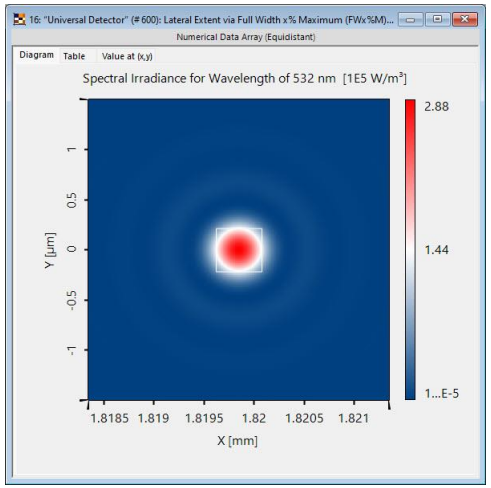
Results – Output Original Field

```
//define name of detector to use (can be only part of the full detector name)
string nameDetectorToUse = "600";
//define name of sub-detector to use (can be only part of the full detector name)
//string nameSubDetectorToUse = "Size X";
string nameSubDetectorToUse = "Original Data";
```

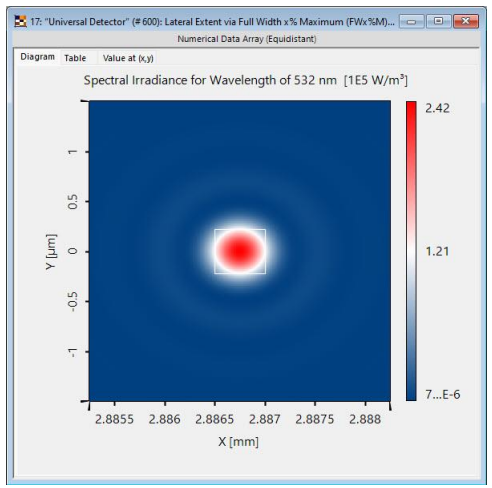
If the parameter *nameSubDetectorToUse* is set to *"Original Data"*, a series of *DataArrays* will be created, depicting the irradiance at the detector plane.



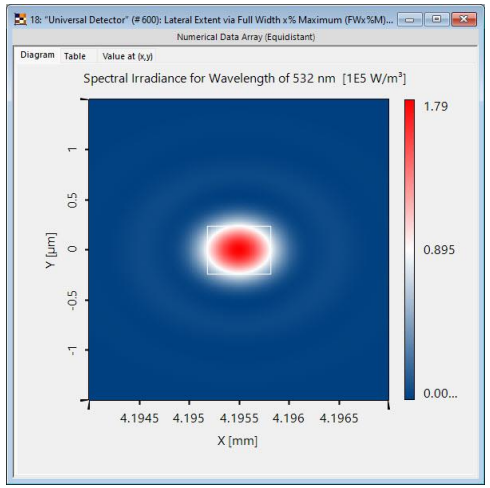
Angle: 10°



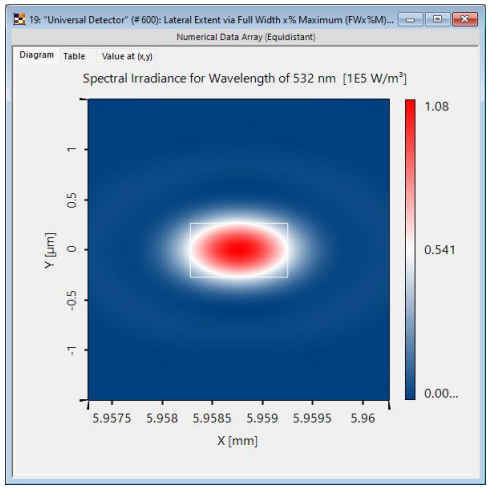
Angle: 20°



Angle: 30°



Angle: 40°

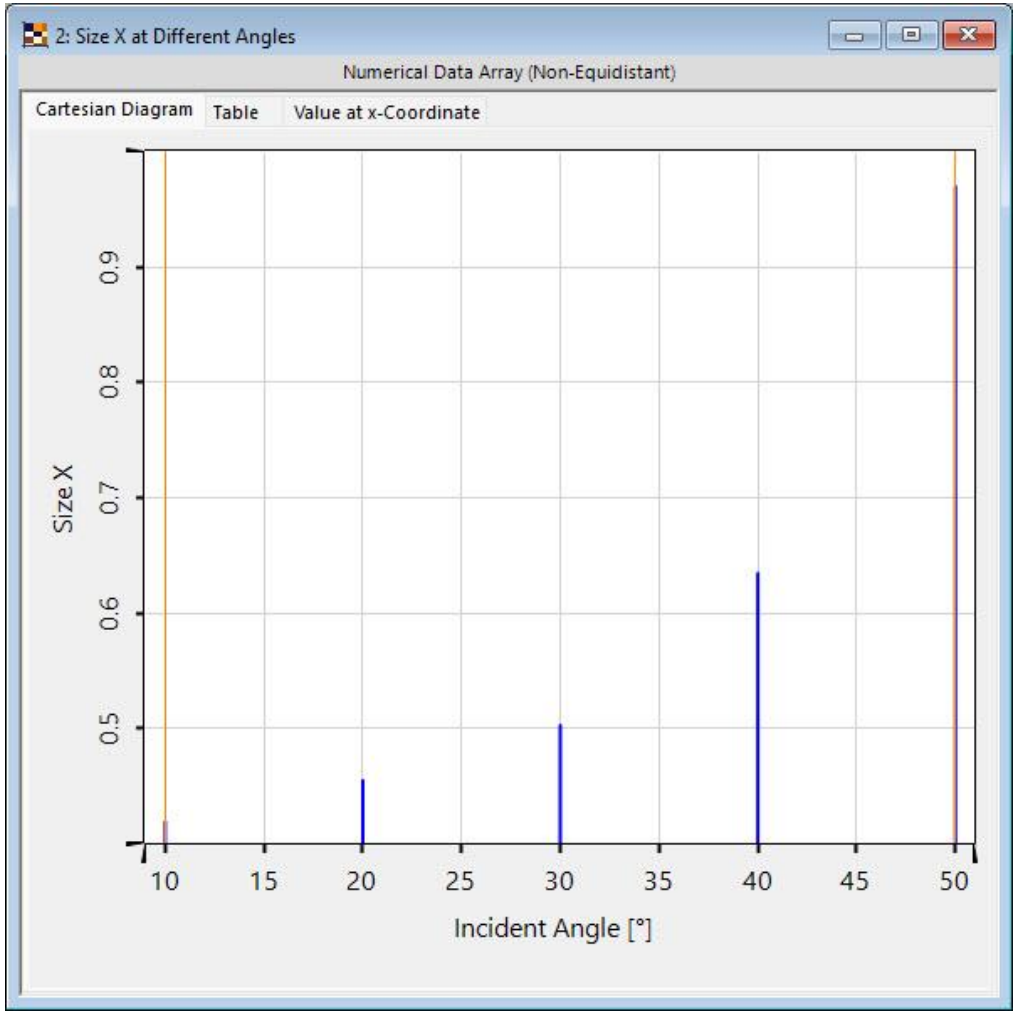


Angle: 50°

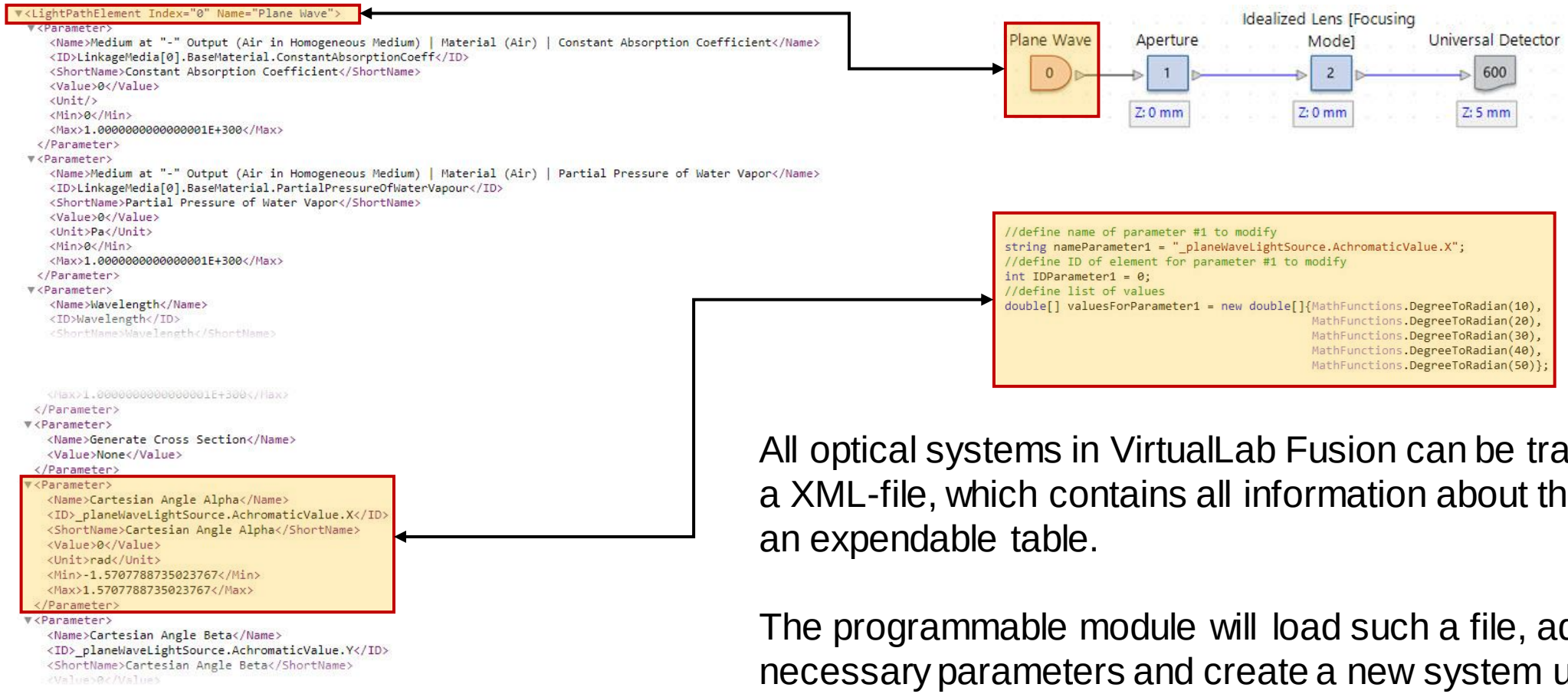
Results – Output Original Field

```
//define name of detector to use (can be only part of the full detector name)
string nameDetectorToUse = "600";
//define name of sub-detector to use (can be only part of the full detector name)
string nameSubDetectorToUse = "Size X";
//string nameSubDetectorToUse = "Original Data";
```

If the parameter *nameSubDetectorToUse* is set to *"Size X"* instead, the results of the simulations (each providing a single value) are collected and visualized in a *1DDataArray*.



Appendix – Working Principle/XML files



All optical systems in VirtualLab Fusion can be translated into a XML-file, which contains all information about the system in an expendable table.

The programmable module will load such a file, adjust the necessary parameters and create a new system using the parameters from the XML-file.

Document Information

title	High NA Lens Focusing per Module
document code	SWF.0047
document version	1.0
software edition	• VirtualLab Fusion Basic
software version	2023.2 (Build 1.242)
category	Feature Use Case
further reading	- Usage of the Parameter Run Document - Absorption in a CIGS Solar Cell