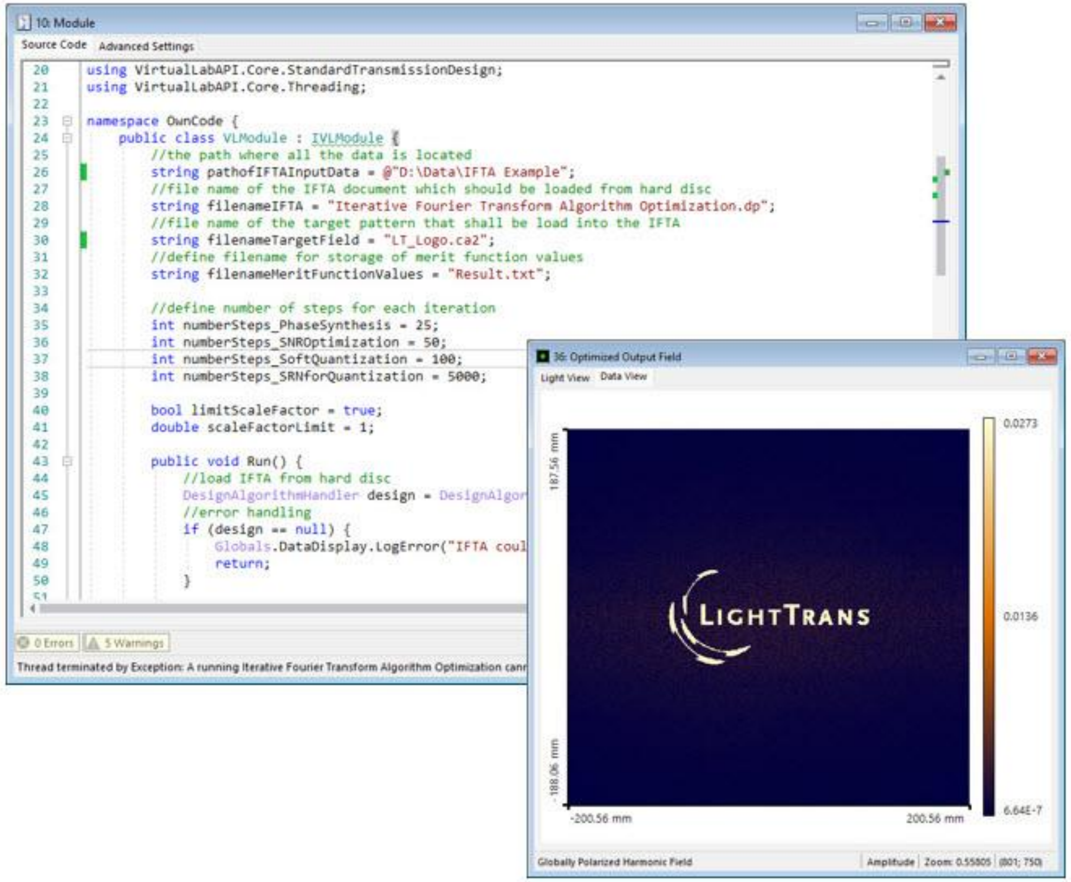
Iterative Fourier Transform Algorithm (IFTA) Design via Module

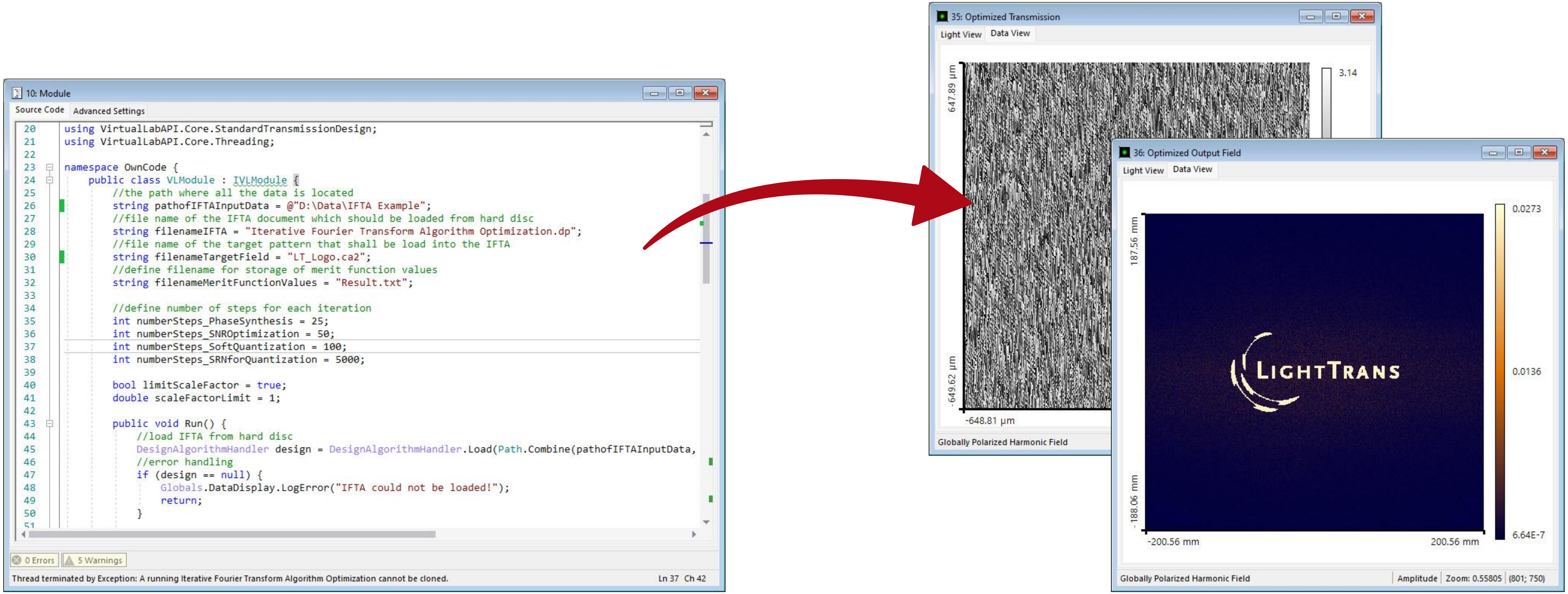
Abstract



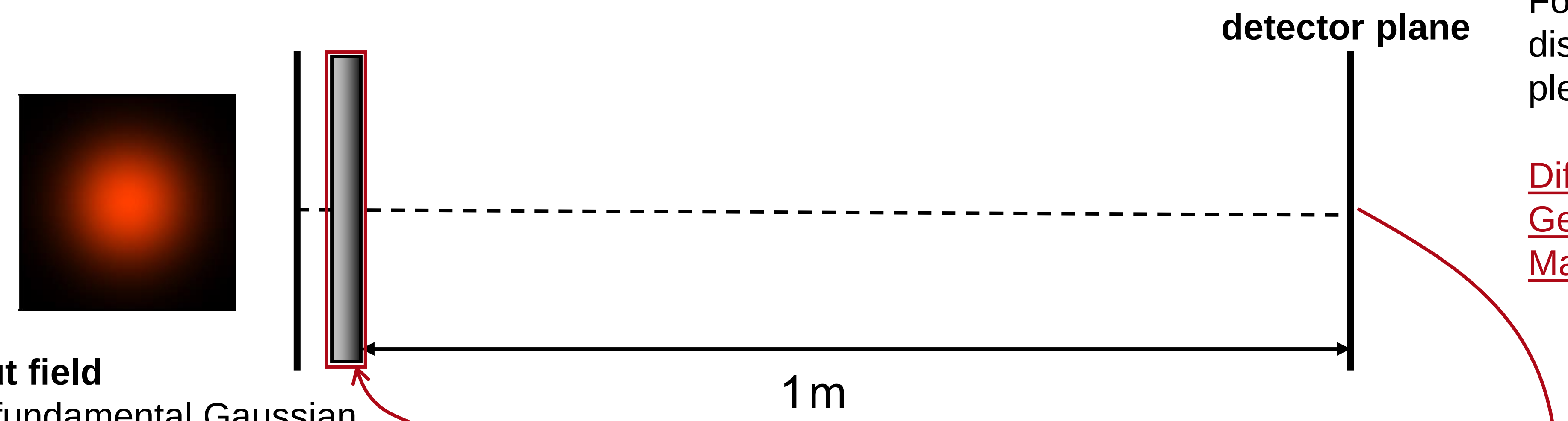
The Iterative Fourier Transform Algorithm (IFTA) is a powerful tool to perform structure design for diffusors, beam splitter and beam shapers. In this Use Case we will demonstrate how to control the design tool by a programmed module, providing a flexible workflow for the user to operate the IFTA even when using VirtualLab Fusion in batch mode or in a python environment.

This Use Case Shows ….

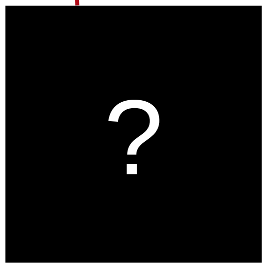
How to use a programmable module to perform an IFTA - design.



Design Task



input field
- fundamental Gaussian
- wavelength: 650 nm
- diameter (waist): 1 mm



diffractive diffuser
4-level phase

target pattern

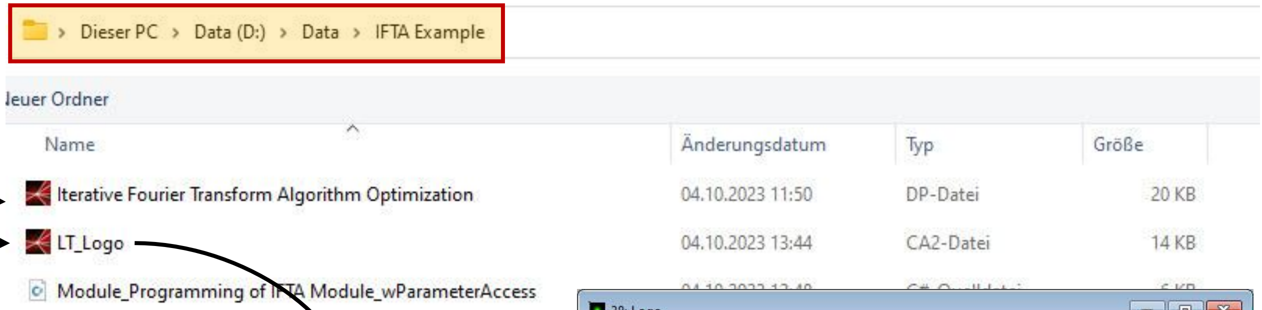
For a detailed discussion of the task, please see:

Diffractive Diffusor to Generate a LightTrans Mark

The Module – Preparations I

```
namespace OwnCode {
    public class VLModule : IVLModule {
        //the path where all the data is located
        string pathofIFTAInputData = @"D:\Data\IFTA Example";
        //file name of the IFTA document which should be loaded from hard disc
        string filenameIFTA = "Iterative Fourier Transform Algorithm Optimization.dp";
        //file name of the target pattern that shall be load into the IFTA
        string filenameTargetField = "LT_Logo.ca2";
        //define filename for storage of merit function values
        string filenameMeritFunctionValues = "Result.txt";

        //define number of steps for each iteration
        int numberSteps_PhaseSynthesis = 25;
        int numberSteps_SNROptimization = 50;
        int numberSteps_SoftQuantization = 100;
        int numberSteps_SRNforQuantization = 5000;

        bool limitScaleFactor = true;
        double scaleFactorLimit = 1;

        public void Run() {
            //load IFTA from hard disc
            DesignAlgorithmHandler design = DesignAlgorithmHandler.Load(Path.Combine(pathofIFTAInputData, filenameIFTA));
            //error handling
            if (design == null) {
                Globals.DataDisplay.LogError("IFTA could not be loaded!");
                return;
            }

            //load signal field
            ComplexAmplitude caSignalField = ComplexAmplitude.Load(Path.Combine(pathofIFTAInputData, filenameTargetField));

            //error handling
            if (caSignalField == null) {
                Globals.DataDisplay.LogError("Signal could not be loaded!");
                return;
            }

            //set signal field to IFTA document
            design.ConstraintSpecification.SetSignalFieldAndSignalRegion(caSignalField.Field, null);

            //define number of steps
            design.NumItSPO = numberSteps_PhaseSynthesis;
            design.NumItSNRPhase = numberSteps_SNROptimization;
            design.NumItSoftQuant = numberSteps_SoftQuantization;
            design.NumItSNRQuant = numberSteps_SRNforQuantization;

            //set parameter for limit scale factor
            design.ConstraintSpecification.LimitScaleFactor = limitScaleFactor;
            design.ConstraintSpecification.ScaleFactorLimitGoalEff = scaleFactorLimit;


            //read sampling parameters from design document
            SamplingParameters sPara = new SamplingParameters();
            sPara = new SamplingParameters(design.ConstraintSpecification.SamplingPoints,
                                           design.ConstraintSpecification.SamplingDistance);
```

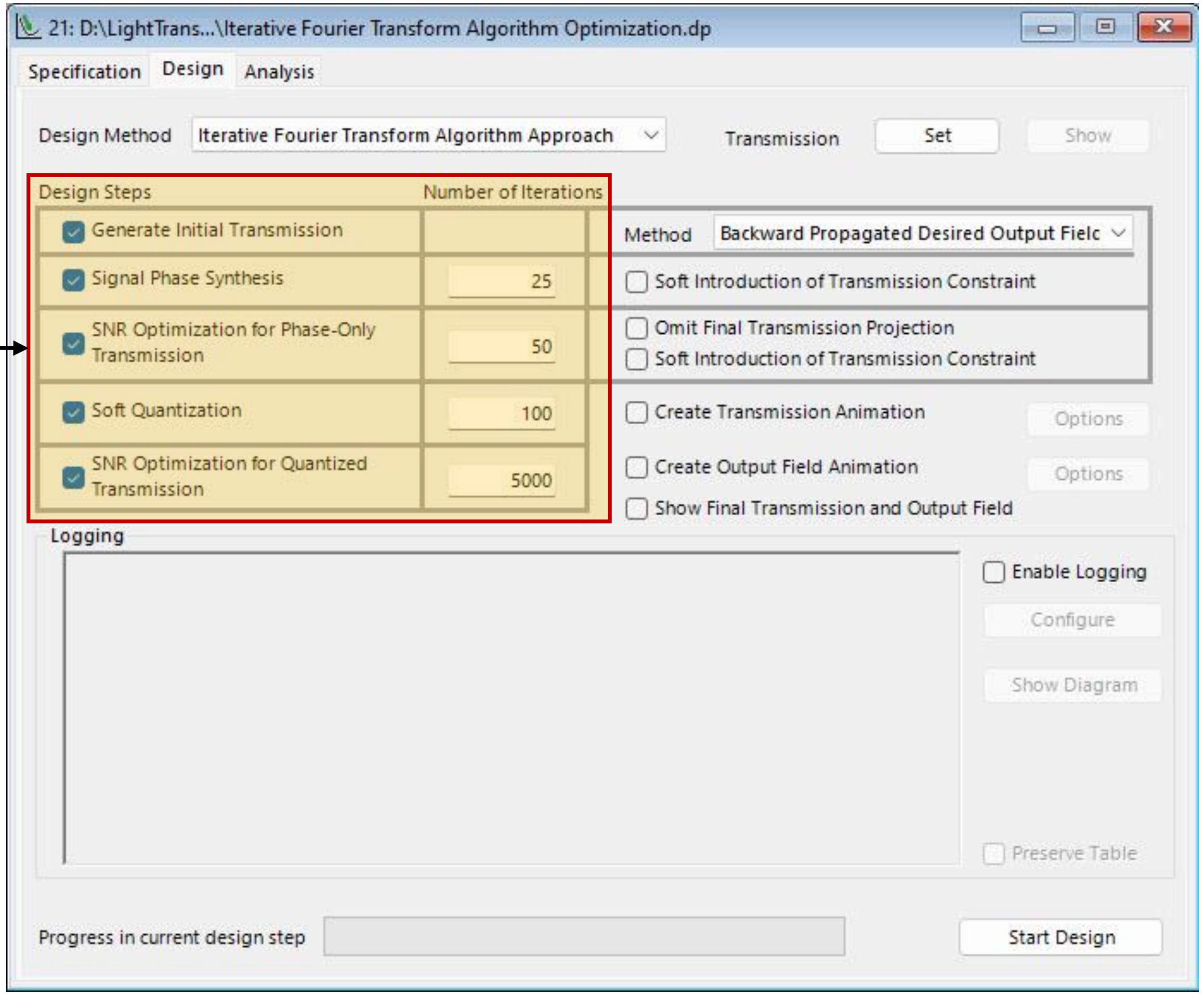
Here, the path of the folder, where the OS- and DP-files are saved needs to be defined.



Please note, that the sampling parameters of the desired output field must match the sampling specified in the IFTA (DP)-file, otherwise an error will occur.

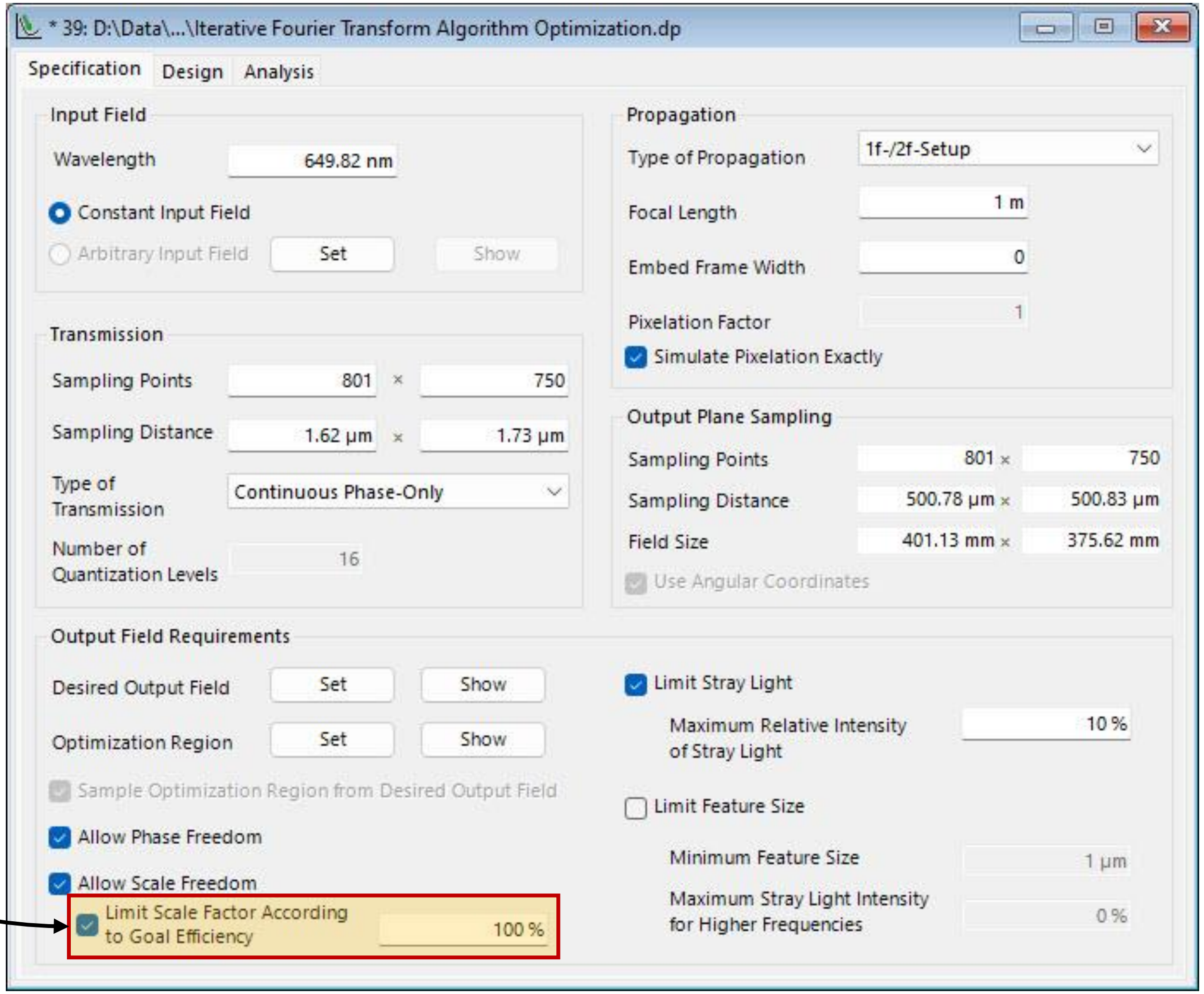
The Module – Preparations II

```
namespace OwnCode {
    public class VLModule : IVLModule {
        //the path where all the data is located
        string pathofIFTAInputData = @"D:\Data\IFTA Example";
        //file name of the IFTA document which should be loaded from hard disc
        string filenameIFTA = "Iterative Fourier Transform Algorithm Optimization.dp";
        //file name of the target pattern that shall be load into the IFTA
        string filenameTargetField = "Desired Output Field 3x4.ca2";
        //define filename for storage of merit function values
        string filenameMeritFunctionValues = "Result.txt";

        //define number of steps for each iteration
        int numberSteps_PhaseSynthesis = 25;
        int numberSteps_SNROptimization = 50;
        int numberSteps_SoftQuantization = 100;
        int numberSteps_SRNforQuantization = 5000;

        bool limitScaleFactor = true;
        double scaleFactorLimit = 1;

        public void Run() {
            //load IFTA from hard disc
            DesignAlgorithmHandler design = DesignAlgorithmHandler.Load(Path.Combine(pathofIFTAInputData, filenameIFTA));
            //error handling
            if (design == null) {
                Globals.DataDisplay.LogError("IFTA could not be loaded!");
                return;
            }

            //load signal field
            ComplexAmplitude caSignalField = ComplexAmplitude.Load(Path.Combine(pathofIFTAInputData, filenameTargetField));

            //error handling
            if (caSignalField == null) {
                Globals.DataDisplay.LogError("Signal could not be loaded!");
                return;
            }

            //set signal field to IFTA document
            design.ConstraintSpecification.SetSignalFieldAndSignalRegion(caSignalField.Field, null);

            //define number of steps
            design.NumItSPO = numberSteps_PhaseSynthesis;
            design.NumItSNRPhase = numberSteps_SNROptimization;
            design.NumItSoftQuant = numberSteps_SoftQuantization;
            design.NumItSNRQuant = numberSteps_SRNforQuantization;

            //set parameter for limit scale factor
            design.ConstraintSpecification.LimitScaleFactor = limitScaleFactor;
            design.ConstraintSpecification.ScaleFactorLimitGoalEff = scaleFactorLimit;



            //read sampling parameters from design document
            SamplingParameters sPara = new SamplingParameters();
            sPara = new SamplingParameters(design.ConstraintSpecification.SamplingPoints,
                                           design.ConstraintSpecification.SamplingDistance);
```



The Module – Preparations III

```
namespace OwnCode {
    public class VLModule : IVLModule {
        //the path where all the data is located
        string pathofIFTAInputData = @"D:\Data\IFTA Example";
        //file name of the IFTA document which should be loaded from hard disc
        string filenameIFTA = "Iterative Fourier Transform Algorithm Optimization.dp";
        //file name of the target pattern that shall be load into the IFTA
        string filenameTargetField = "Desired Output Field 3x4.ca2";
        //define filename for storage of merit function values
        string filenameMeritFunctionValues = "Result.txt";

        //define number of steps for each iteration
        int numberSteps_PhaseSynthesis = 25;
        int numberSteps_SNROptimization = 50;
        int numberSteps_SoftQuantization = 100;
        int numberSteps_SRNforQuantization = 5000;

        bool limitScaleFactor = true;
        double scaleFactorLimit = 1;

        public void Run() {
            //load IFTA from hard disc
            DesignAlgorithmHandler design = DesignAlgorithmHandler.Load(Path.Combine(pathofIFTAInputData, filenameIFTA));
            //error handling
            if (design == null) {
                Globals.DataDisplay.LogError("IFTA could not be loaded!");
                return;
            }

            //load signal field
            ComplexAmplitude caSignalField = ComplexAmplitude.Load(Path.Combine(pathofIFTAInputData, filenameTargetField));

            //error handling
            if (caSignalField == null) {
                Globals.DataDisplay.LogError("Signal could not be loaded!");
                return;
            }

            //set signal field to IFTA document
            design.ConstraintSpecification.SetSignalFieldAndSignalRegion(caSignalField.Field, null);

            //define number of steps
            design.NumItSPO = numberSteps_PhaseSynthesis;
            design.NumItSNRPhase = numberSteps_SNROptimization;
            design.NumItSoftQuant = numberSteps_SoftQuantization;
            design.NumItSNRQuant = numberSteps_SRNforQuantization;

            //set parameter for limit scale factor
            design.ConstraintSpecification.LimitScaleFactor = limitScaleFactor;
            design.ConstraintSpecification.ScaleFactorLimitGoalEff = scaleFactorLimit;


            //read sampling parameters from design document
            SamplingParameters sPara = new SamplingParameters();
            sPara = new SamplingParameters(design.ConstraintSpecification.SamplingPoints,
                                           design.ConstraintSpecification.SamplingDistance);
```



The Module – Loading the Target Field and IFTA Parameters

```
namespace OwnCode {
    public class VLModule : IVLModule {
        //the path where all the data is located
        string pathofIFTAInputData = @"D:\Data\IFTA Example";
        //file name of the IFTA document which should be loaded from hard disc
        string filenameIFTA = "Iterative Fourier Transform Algorithm Optimization.dp";
        //file name of the target pattern that shall be load into the IFTA
        string filenameTargetField = "Desired Output Field 3x4.ca2";
        //define filename for storage of merit function values
        string filenameMeritFunctionValues = "Result.txt";

        //define number of steps for each iteration
        int numberSteps_PhaseSynthesis = 25;
        int numberSteps_SNROptimization = 50;
        int numberSteps_SoftQuantization = 100;
        int numberSteps_SRNforQuantization = 5000;

        bool limitScaleFactor = true;
        double scaleFactorLimit = 1;

        public void Run() {
            //load IFTA from hard disc
            DesignAlgorithmHandler design = DesignAlgorithmHandler.Load(Path.Combine(pathofIFTAInputData, filenameIFTA));
            //error handling
            if (design == null) {
                Globals.DataDisplay.LogError("IFTA could not be loaded!");
                return;
            }

            //load signal field
            ComplexAmplitude caSignalField = ComplexAmplitude.Load(Path.Combine(pathofIFTAInputData, filenameTargetField));

            //error handling
            if (caSignalField == null) {
                Globals.DataDisplay.LogError("Signal could not be loaded!");
                return;
            }

            //set signal field to IFTA document
            design.ConstraintSpecification.SetSignalFieldAndSignalRegion(caSignalField.Field, null);

            //define number of steps
            design.NumItSPO = numberSteps_PhaseSynthesis;
            design.NumItSNRPhase = numberSteps_SNROptimization;
            design.NumItSoftQuant = numberSteps_SoftQuantization;
            design.NumItSNRQuant = numberSteps_SRNforQuantization;

            //set parameter for limit scale factor
            design.ConstraintSpecification.LimitScaleFactor = limitScaleFactor;
            design.ConstraintSpecification.ScaleFactorLimitGoalEff = scaleFactorLimit;


            //read sampling parameters from design document
            SamplingParameters sPara = new SamplingParameters();
            sPara = new SamplingParameters(design.ConstraintSpecification.SamplingPoints,
                                           design.ConstraintSpecification.SamplingDistance);
```

In the first step, the previously specified IFTA DP-file will be loaded. All information about the optical setup will be extracted from this file. To perform the IFTA per module, a new *DesignAlgorithmHandler* is created.

In the next section, the desired output field is loaded from file, which was defined above. Please note, that the sampling parameters of this file must match the sampling specified in the DP-file, otherwise an error will occur.

Next, the parameters, which were defined above are loaded into the *DesignAlgorithmHandler.* In this example, only the *Limit Scale Factor According to Goal Efficiency factor* is specified, but all other parameter of the IFTA document can be adapted in the same way.

The Module – Performing the Design

```
//read sampling parameters from design document
SamplingParameters sPara = new SamplingParameters();
sPara = new SamplingParameters(design.ConstraintSpecification.SamplingPoints,
                               design.ConstraintSpecification.SamplingDistance);
```

To visualize the results appropriately, the sampling parameters defined in the *DesignAlgorithmHandler* are extracted.

```
if (design.Transmission != null) {
    //show initial transmission
    ComplexAmplitude initialTransmission = new ComplexAmplitude(design.Transmission);
    initialTransmission.SamplingDistance = sPara.SamplingDistance;
    Globals.DataDisplay.ShowDocument(initialTransmission, "Initial Transmission");
}
```

This section outputs the initial transmission, if one has been defined.

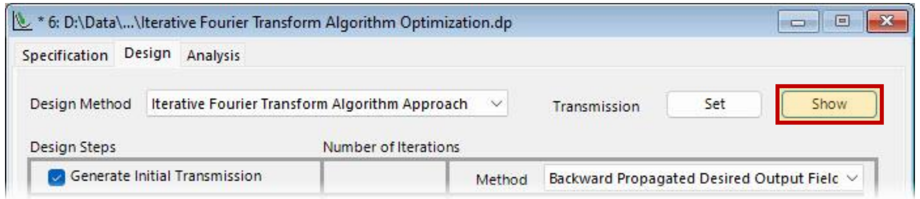
```
//perform design
IterationDataOutput iterationDataOutput = new IterationDataOutput();
WorkerThread worker = new WorkerThread("Design", null, null, null);
design.DoAllSelectedSteps(iterationDataOutput, worker);
```

This command performs the IFTA design, finally.

```
//show optimized transmission
ComplexAmplitude optimizedTransmission = new ComplexAmplitude(design.Transmission);
optimizedTransmission.SamplingDistance = sPara.SamplingDistance;
Globals.DataDisplay.ShowDocument(optimizedTransmission, "Optimized Transmission");
```

Here, the transmission function designed by the IFTA is extracted and output. The output is the same as if clicking the *Show* button on the *Design* tab.

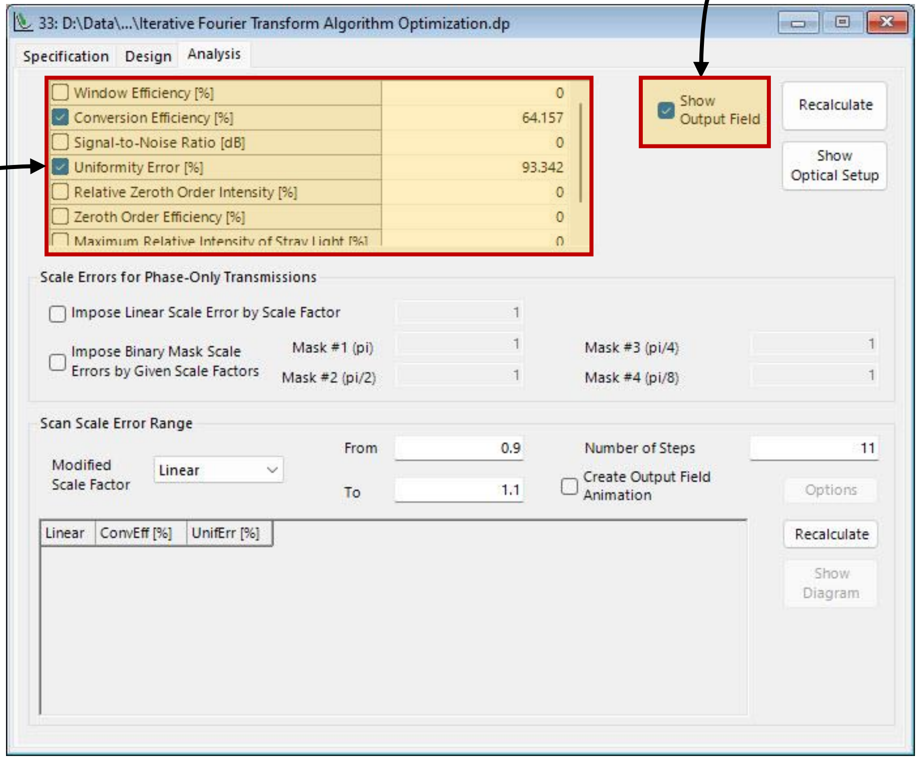
```
SamplingParameters sParaOutPutField = new SamplingParameters();
sParaOutPutField = new SamplingParameters(design.ConstraintSpecification.SamplingPoints,
                                          design.ConstraintSpecification.SamplingDistanceOut);
//design.ConstraintSpecification.
ComplexField cfOutputField = design.DoForwardProp();
//show optimized transmission
ComplexAmplitude resultField = new ComplexAmplitude(cfOutputField);
resultField.SamplingDistance = sParaOutPutField.SamplingDistance;
Globals.DataDisplay.ShowDocument(resultField, "Optimized Output Field");

double conversionEff = design.GetConversionEfficiency(cfOutputField, sParaOutPutField);
double uniformityError = design.GetUniformityError(cfOutputField, sParaOutPutField);
//log conversion efficiency
Globals.DataDisplay.LogMessage("Conversion Efficiency = " + PhysicalUnits.FormatPhysicalUnit(conversionEff,
PhysicalProperty.Percentage));
Globals.DataDisplay.LogMessage("Uniformity Error = " + PhysicalUnits.FormatPhysicalUnit(uniformityError,
PhysicalProperty.Percentage));


Globals.DataDisplay.ShowDocument(design, "Modified Design (performed!)");

//define path for storing merit functions
string filenameMeritFunction = Path.Combine(pathofIFTAInputData, filenameMeritFunctionValues);
//check whether file already exist
if (File.Exists(filenameMeritFunction)) {
    File.Delete(filenameMeritFunction);
}

VL_Files.WriteLineToLogFile(filenameMeritFunction, "Conversion Efficiency = " +
PhysicalUnits.FormatPhysicalUnit(conversionEff, PhysicalProperty.Percentage));
VL_Files.WriteLineToLogFile(filenameMeritFunction, "Uniformity Error = " +
PhysicalUnits.FormatPhysicalUnit(uniformityError, PhysicalProperty.Percentage));
```



The Module – Analysis of the Result

```
//read sampling parameters from design document
SamplingParameters sPara = new SamplingParameters();
sPara = new SamplingParameters(design.ConstraintSpecification.SamplingPoints,
                               design.ConstraintSpecification.SamplingDistance);

if (design.Transmission != null) {
    //show initial transmission
    ComplexAmplitude initialTransmission = new ComplexAmplitude(design.Transmission);
    initialTransmission.SamplingDistance = sPara.SamplingDistance;
    Globals.DataDisplay.ShowDocument(initialTransmission, "Initial Transmission");
}

//perform design
IterationDataOutput iterationDataOutput = new IterationDataOutput();
WorkerThread worker = new WorkerThread("Design", null, null, null);
design.DoAllSelectedSteps(iterationDataOutput, worker);


//show optimized transmission
ComplexAmplitude optimizedTransmission = new ComplexAmplitude(design.Transmission);
optimizedTransmission.SamplingDistance = sPara.SamplingDistance;
Globals.DataDisplay.ShowDocument(optimizedTransmission, "Optimized Transmission");

SamplingParameters sParaOutPutField = new SamplingParameters();
sParaOutPutField = new SamplingParameters(design.ConstraintSpecification.SamplingPoints,
                                          design.ConstraintSpecification.SamplingDistanceOut);
//design.ConstraintSpecification.
ComplexField cfOutputField = design.DoForwardProp();
//show optimized transmission
ComplexAmplitude resultField = new ComplexAmplitude(cfOutputField);
resultField.SamplingDistance = sParaOutPutField.SamplingDistance;
Globals.DataDisplay.ShowDocument(resultField, "Optimized Output Field");

double conversionEff = design.GetConversionEfficiency(cfOutputField, sParaOutPutField);
double uniformityError = design.GetUniformityError(cfOutputField, sParaOutPutField);
//log conversion efficiency
Globals.DataDisplay.LogMessage("Conversion Efficiency = " + PhysicalUnits.FormatPhysicalUnit(conversionEff,
PhysicalProperty.Percentage));
Globals.DataDisplay.LogMessage("Uniformity Error = " + PhysicalUnits.FormatPhysicalUnit(uniformityError,
PhysicalProperty.Percentage));


Globals.DataDisplay.ShowDocument(design, "Modified Design (performed!)");

//define path for storing merit functions
string filenameMeritFunction = Path.Combine(pathofIFTAInputData, filenameMeritFunctionValues);
//check whether file already exist
if (File.Exists(filenameMeritFunction)) {
    File.Delete(filenameMeritFunction);
}

VL_Files.WriteLineToLogFile(filenameMeritFunction, "Conversion Efficiency = " +
PhysicalUnits.FormatPhysicalUnit(conversionEff, PhysicalProperty.Percentage));
VL_Files.WriteLineToLogFile(filenameMeritFunction, "Uniformity Error = " +
PhysicalUnits.FormatPhysicalUnit(uniformityError, PhysicalProperty.Percentage));
```

After the design was finished, the module will automatically perform an analysis. For this purpose, the optimized result field is calculated and output, similar to the option *Show Output Field* in the IFTA document.



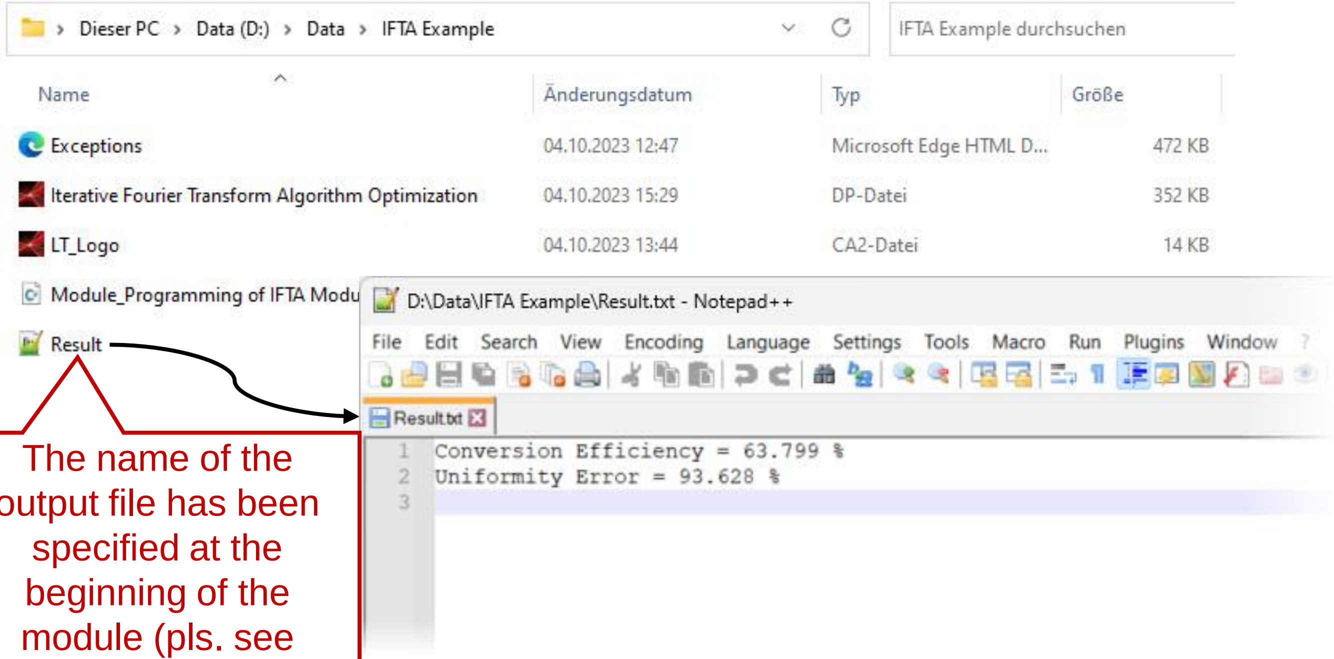
The *DesignAlgorithm Handler* provides various performance parameters which can be output: Here, just the *Conversion Efficiency* and *Uniformity Error* are used.

The Module – Export Output Data

```
//read sampling parameters from design document
SamplingParameters sPara = new SamplingParameters();
sPara = new SamplingParameters(design.ConstraintSpecification.SamplingPoints,
                               design.ConstraintSpecification.SamplingDistance);

if (design.Transmission != null) {
    //show initial transmission
    ComplexAmplitude initialTransmission = new ComplexAmplitude(design.Transmission);
    initialTransmission.SamplingDistance = sPara.SamplingDistance;
    Globals.DataDisplay.ShowDocument(initialTransmission, "Initial Transmission");
}

//perform design
IterationDataOutput iterationDataOutput = new IterationDataOutput();
WorkerThread worker = new WorkerThread("Design", null, null, null);
design.DoAllSelectedSteps(iterationDataOutput, worker);


//show optimized transmission
ComplexAmplitude optimizedTransmission = new ComplexAmplitude(design.Transmission);
optimizedTransmission.SamplingDistance = sPara.SamplingDistance;
Globals.DataDisplay.ShowDocument(optimizedTransmission, "Optimized Transmission");

SamplingParameters sParaOutPutField = new SamplingParameters();
sParaOutPutField = new SamplingParameters(design.ConstraintSpecification.SamplingPoints,
                                          design.ConstraintSpecification.SamplingDistanceOut);
//design.ConstraintSpecification.
ComplexField cfOutputField = design.DoForwardProp();
//show optimized transmission
ComplexAmplitude resultField = new ComplexAmplitude(cfOutputField);
resultField.SamplingDistance = sParaOutPutField.SamplingDistance;
Globals.DataDisplay.ShowDocument(resultField, "Optimized Output Field");

double conversionEff = design.GetConversionEfficiency(cfOutputField, sParaOutPutField);
double uniformityError = design.GetUniformityError(cfOutputField, sParaOutPutField);
//log conversion efficiency
Globals.DataDisplay.LogMessage("Conversion Efficiency = " + PhysicalUnits.FormatPhysicalUnit(conversionEff,
PhysicalProperty.Percentage));
Globals.DataDisplay.LogMessage("Uniformity Error = " + PhysicalUnits.FormatPhysicalUnit(uniformityError,
PhysicalProperty.Percentage));


Globals.DataDisplay.ShowDocument(design, "Modified Design (performed!)");

//define path for storing merit functions
string filenameMeritFunction = Path.Combine(pathofIFTAInputData, filenameMeritFunctionValues);
//check whether file already exist
if (File.Exists(filenameMeritFunction)) {
    File.Delete(filenameMeritFunction);
}

VL_Files.WriteLineToLogFile(filenameMeritFunction, "Conversion Efficiency = " +
PhysicalUnits.FormatPhysicalUnit(conversionEff, PhysicalProperty.Percentage));
VL_Files.WriteLineToLogFile(filenameMeritFunction, "Uniformity Error = " +
PhysicalUnits.FormatPhysicalUnit(uniformityError, PhysicalProperty.Percentage));
```

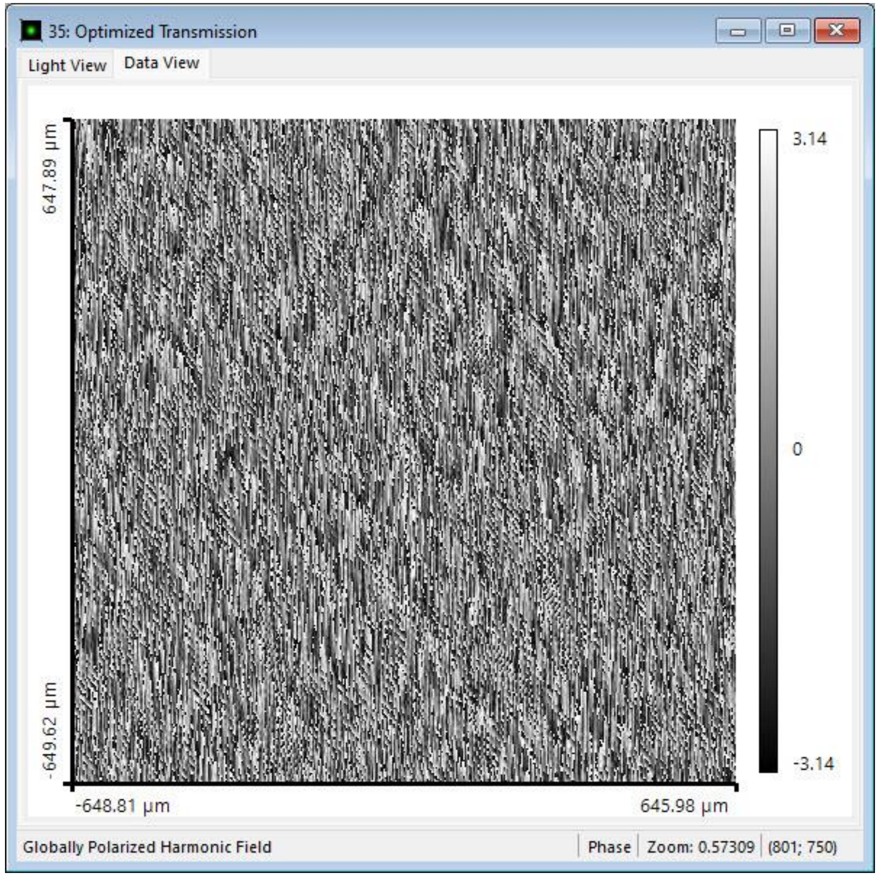
Finally, the resulting performance parameters are output in the *Messages* tab as well as exported in a text file.

```
[2023-10-04 15:38:54] Compile successful
[2023-10-04 15:38:54] Module started
[2023-10-04 15:39:20] Total Time: 00:00:24.9593766
[2023-10-04 15:39:21] Conversion Efficiency = 63.799 %
[2023-10-04 15:39:21] Uniformity Error = 93.628 %
```

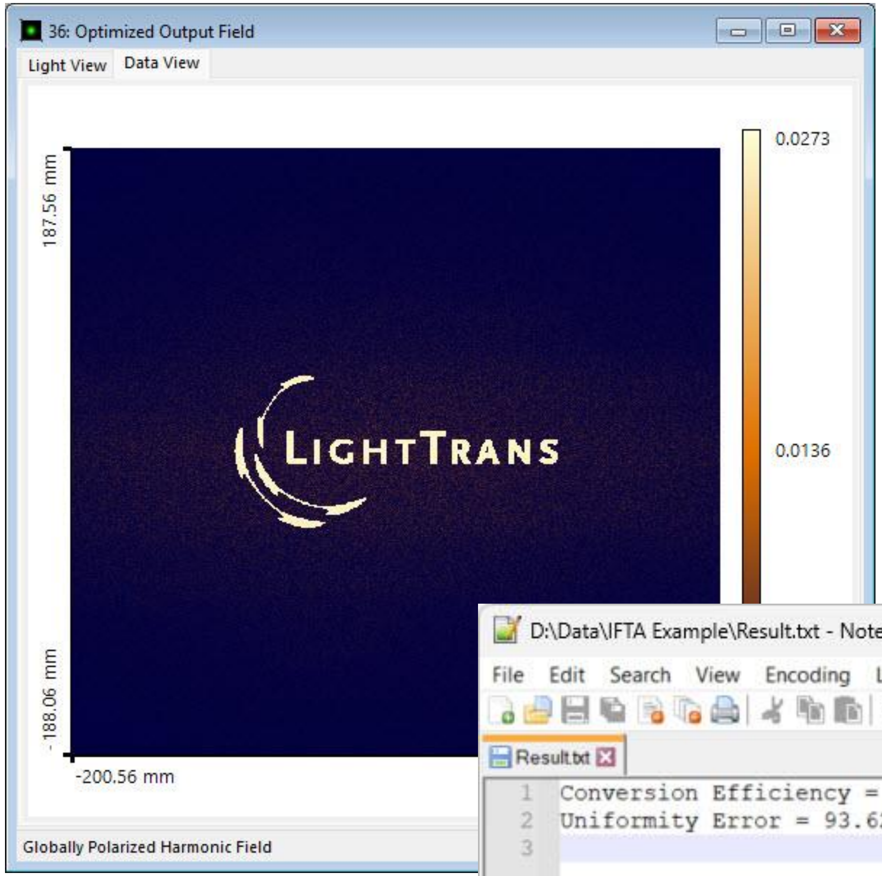


The name of the output file has been specified at the beginning of the module (pls. see page 5).

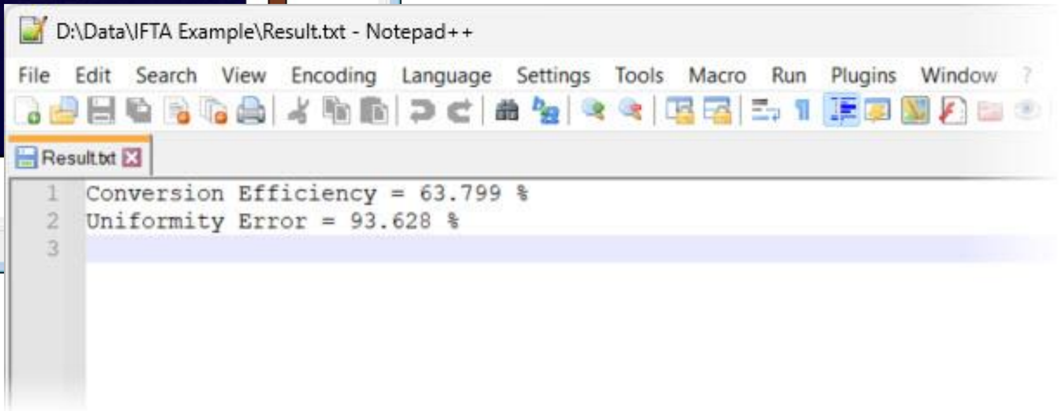
Results – Output Original Field



optimized transmission



optimized output field



performance parameters

Document Information

title	IFTA optimization per Module
document code	SWF.0048
document version	1.0
software edition	VirtualLab Fusion Standard
software version	2023.2 (Build 1.242)
category	Feature Use Case
further reading	- Design of a Diffractive Diffuser to Generate a LightTrans Mark