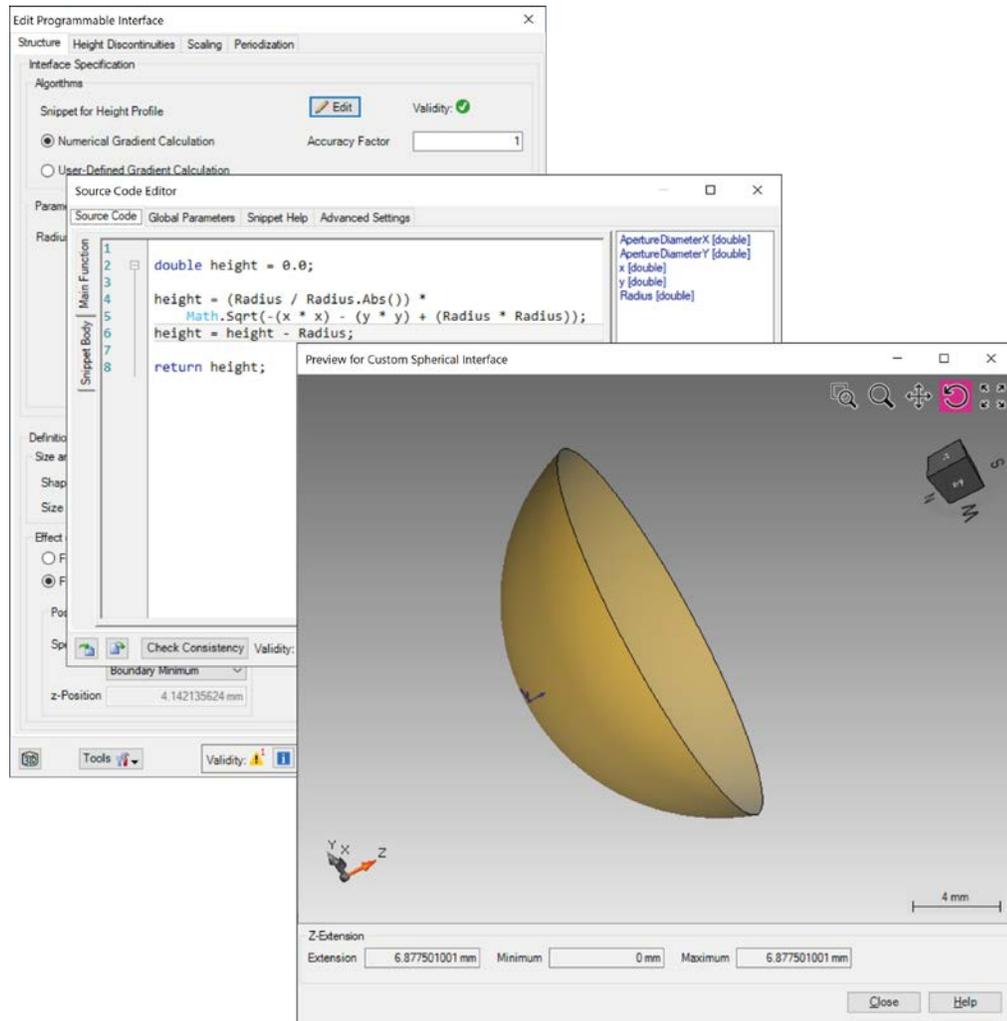


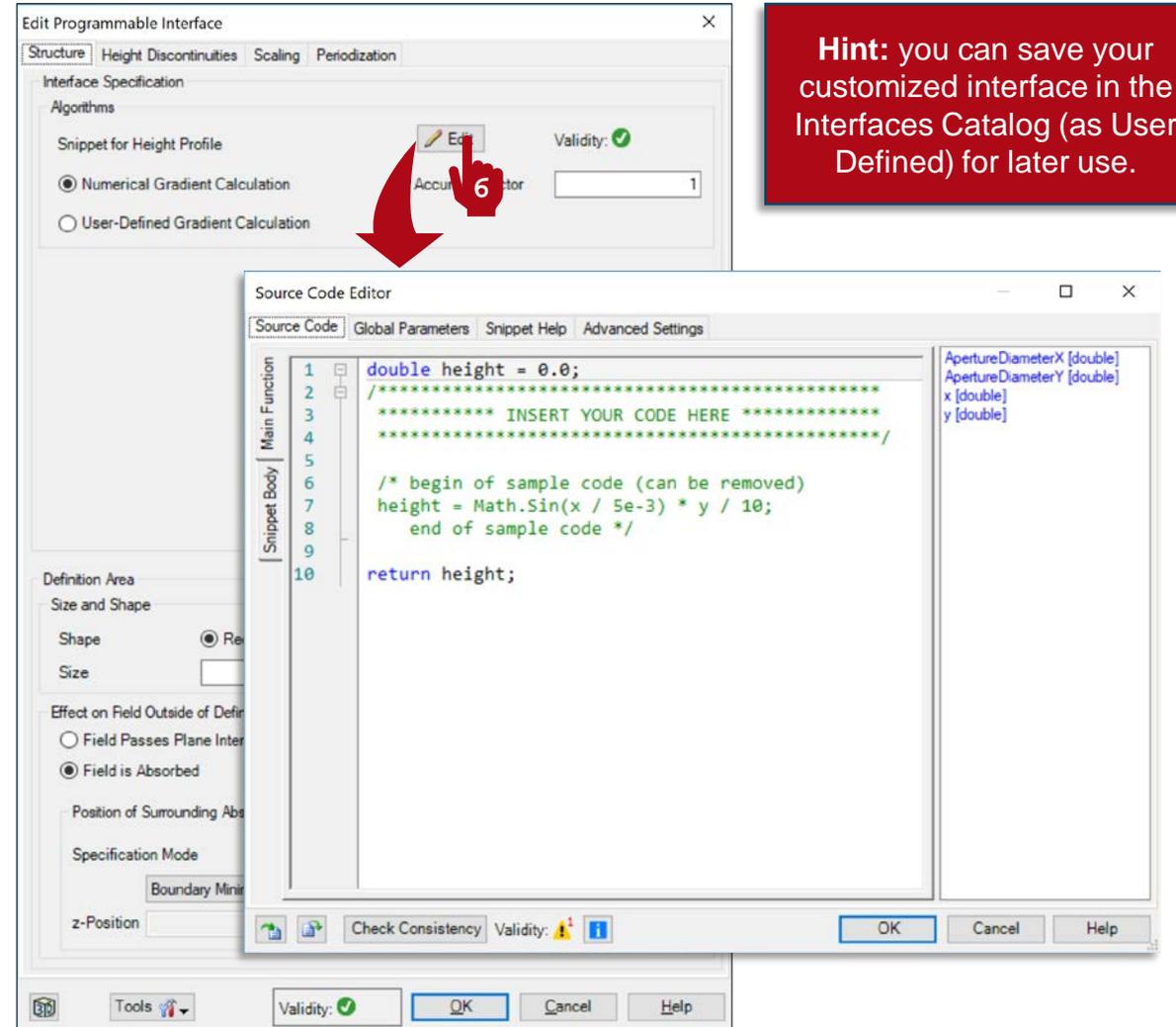
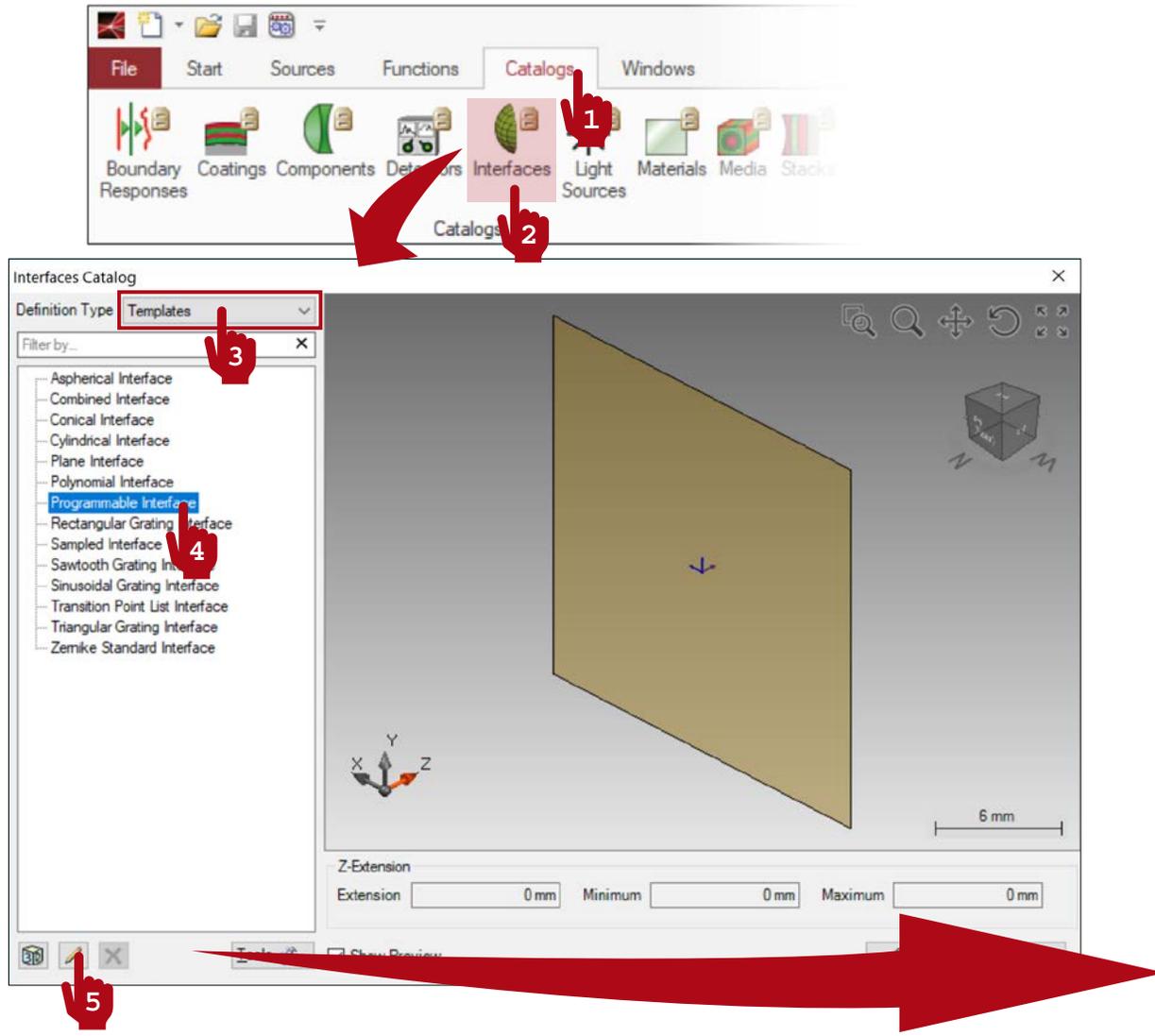
How to Work with the Programmable Interface & Example (Spherical Surface)

Abstract



Providing maximum versatility for your optical simulations is one of our most fundamental objectives. In this document we show you how to program custom surfaces: that is, how to define a height function $h(x, y)$ that describes a 3D surface with respect to the x, y coordinates which span the parametrization plane. These surfaces can then be used to configure the optical components in your system. Conical surfaces, being some of the most fundamental, are of course provided as a default template in VirtualLab; in this tutorial, however, we use a spherical surface as a simple programming example.

Where to Find the Programmable Interface: Catalog



Hint: you can save your customized interface in the Interfaces Catalog (as User Defined) for later use.

Where to Find the Programmable Interface: Components

Step 1: In the **Optical Setup View #1 (Optical Setup)**, the **Optical Interface Sequence** component is highlighted.

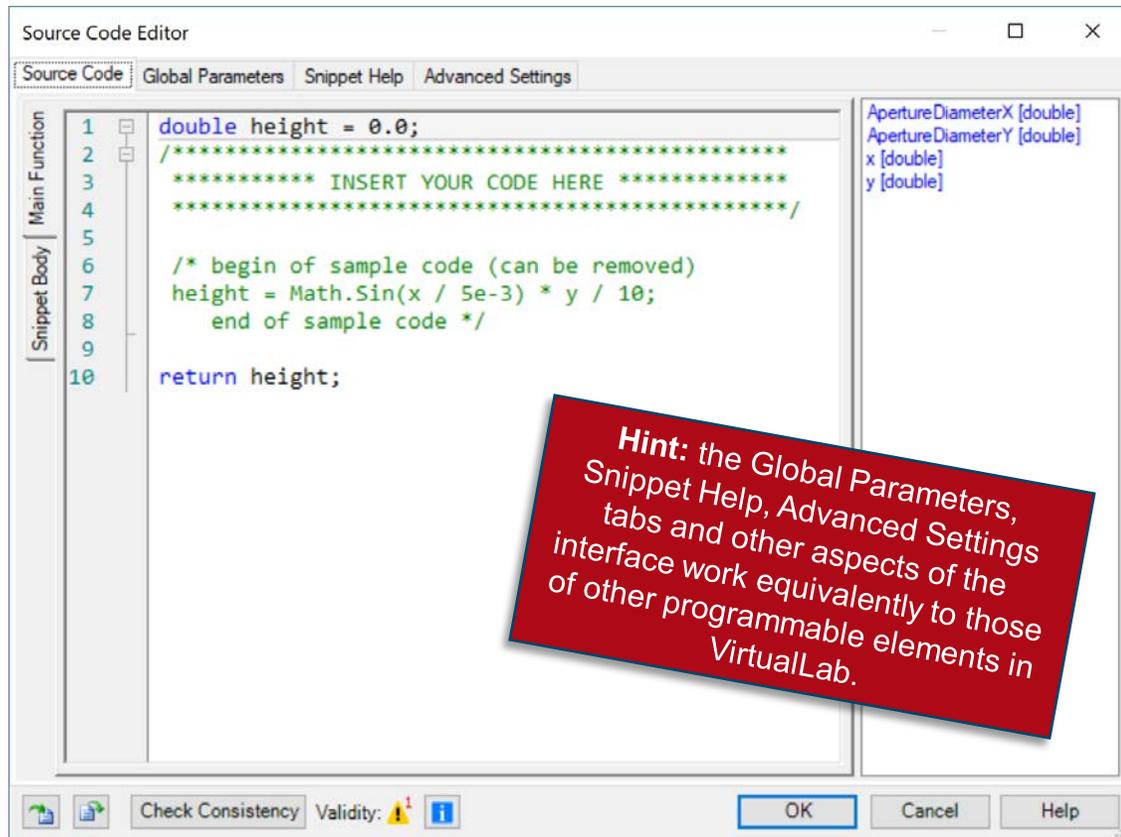
Step 2: In the **Edit Optical Interface Sequence** window, the **Programmable Interface** component is selected from the list.

Step 3: In the **Edit Programmable Interface** window, the **Edit** button is clicked.

Source Code Editor:

```
1 double height = 0.0;
2 .....
3 .....** INSERT YOUR CODE HERE **.....
4 .....
5 .....
6 /* begin of sample code (can be removed)
7 height = Math.Sin(x / 5e-3) * y / 10;
8 end of sample code */
9
10 return height;
```

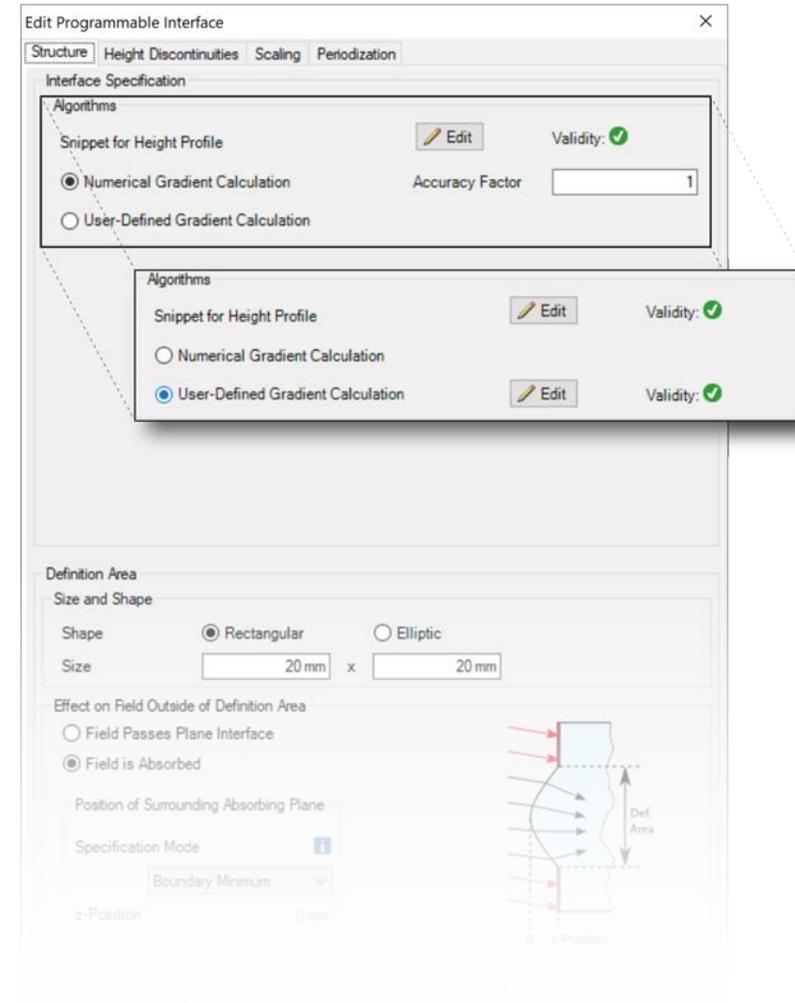
Writing the Code



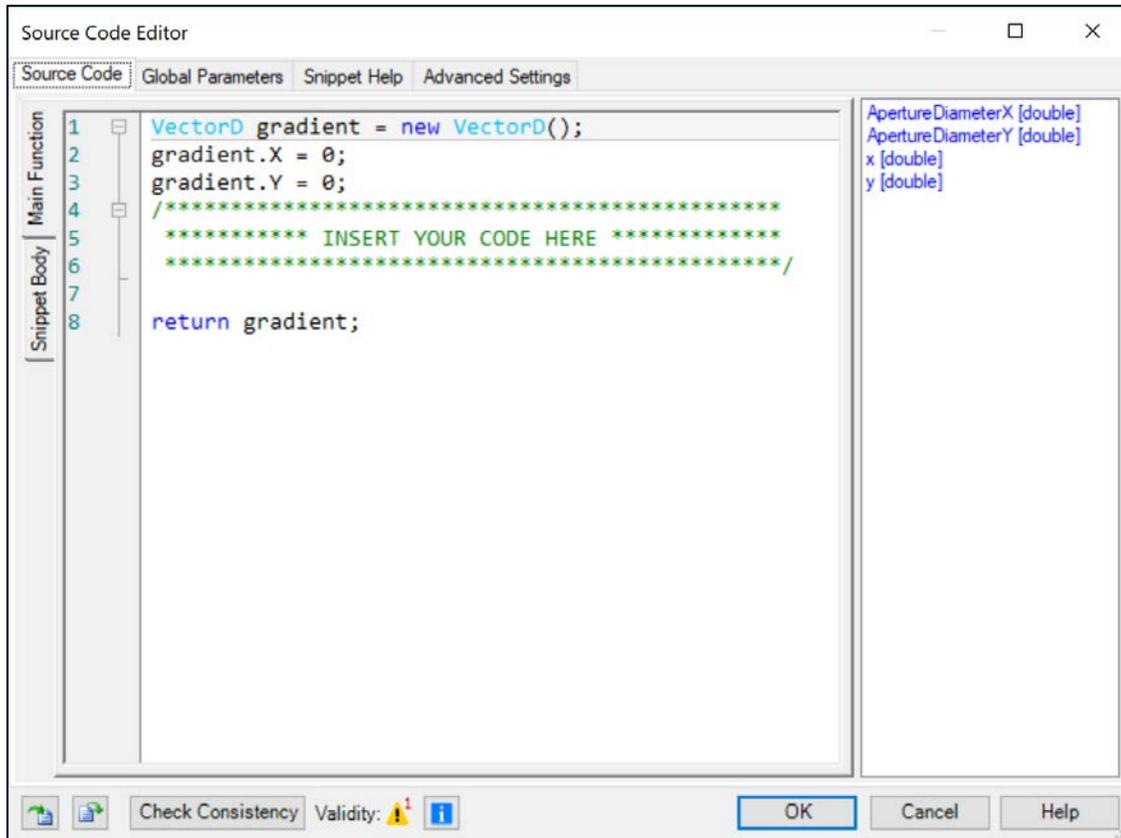
- The panel on the right shows a list of available independent parameters.
- x and y represent the independent variables, the 2D coordinates spanning a plane.
- The interval where x and y are defined is given by `ApertureDiameterX` and `ApertureDiameterY` (both of them determined in the general configuration dialog of the interface).
- The code in the Main Function must return a `double` value per x , y point. This value represents the height at that point. The set of all these height values defines a 3D surface.
- Use the Snippet Body to group parts of the code in support functions.

Definition of the Surface Gradient

- An accurate calculation of the gradient of the interface is fundamental in an optical simulation.
- The Programmable Interface in VirtualLab allows for two different modes of definition of the gradient: numerical, with adjustable accuracy (automatically carried out by the software) or analytical (programmed additionally by the user).



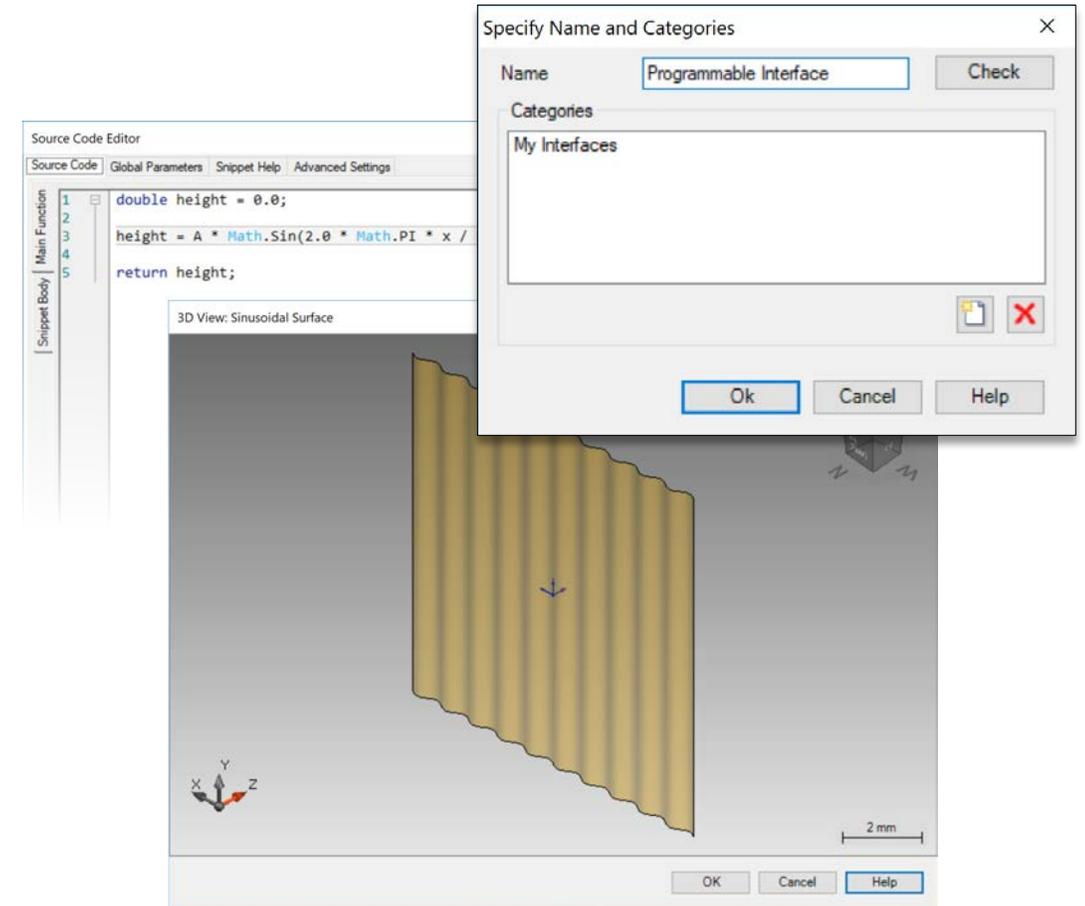
User-Defined Surface Gradient



- The panel on the right shows the same list of available independent variables.
- This time, the code in the Main Function must return a **VectorD**—a vector with two **double** (real-valued) coordinates. Use the Snippet Body to group parts of the code in support functions.
- Using the **analytical definition** of the gradient where possible makes for the **more accurate** alternative. The consistency of the code is checked by the software, but the user must make sure that the function entered for the gradient coincides with the corresponding interface!

Output of the Programmable Interface

- The output is a 3D surface which can be used to define the interface of an actual optical component in a system.
- The custom interface can be programmed directly within the real component where it is required.
- Or, alternatively, it can be saved in the catalog for later use.



Programming a Spherical Surface

The Spherical Surface

A sphere is defined as the locus of all the points x, y, z located at the same distance R from a common centre (which we shall place at the origin)

$$x^2 + y^2 + z^2 = R^2 . \quad (1)$$

Then, to transform Eq. (1) into an expression that generates the height function $h(x, y)$ of a spherical surface

$$h(x, y) = \pm \sqrt{R^2 - (x^2 + y^2)} . \quad (2)$$

It is also possible to straight-forwardly compute the gradient of the surface

$$\frac{\partial h(x, y)}{\partial x} = \mp x [R^2 - (x^2 + y^2)]^{-1/2} \quad \text{and} \quad \frac{\partial h(x, y)}{\partial y} = \mp y [R^2 - (x^2 + y^2)]^{-1/2} \quad (3)$$

Where to Find the Programmable Interface: Catalog

The image illustrates the steps to find and edit a Programmable Interface in a software application. It consists of two main panels: the 'Interfaces Catalog' and the 'Edit Programmable Interface' dialog.

Interfaces Catalog:

- Step 1:** The 'Catalogs' tab is selected in the top menu bar.
- Step 2:** The 'Interfaces' icon is highlighted in the 'Catalogs' toolbar.
- Step 3:** The 'Definition Type' dropdown menu is set to 'Templates'.
- Step 4:** The 'Programmable Interface' option is selected in the list of interface types.
- Step 5:** The 'Add' icon (a plus sign) is clicked at the bottom of the catalog window.

Edit Programmable Interface Dialog:

- Interface Specification:** The 'Numerical Gradient Calculation' algorithm is selected. The 'Accuracy Factor' is set to 1.
- Definition Area:** The 'Shape' is set to 'Rectangular' with a size of 20 mm x 20 mm.
- Effect on Field Outside of Definition Area:** The 'Field is Absorbed' option is selected.
- Position of Surrounding Absorbing Plane:** The 'Specification Mode' is set to 'Boundary Minimum' and the 'z-Position' is 0 mm.

A large red arrow points from the 'Add' icon in the 'Interfaces Catalog' to the 'Edit Programmable Interface' dialog, indicating the transition from selection to editing.

Where to Find the Programmable Interface: Components

The image shows a three-step process to locate the Programmable Interface component in an optical software package:

- Step 1:** In the **Optical Setup View #1 (Optical Setup)***, the **Components** tree on the left is expanded to show **Optical Interface Sequence**. A red arrow points from this component to the next window.
- Step 2:** In the **Edit Optical Interface Sequence** window, a table lists the sequence components. A red arrow points from the **Programmable Interf...** entry to the next window.
- Step 3:** In the **Edit Programmable Interface** window, the **Structure** tab is active, showing configuration options for the interface.

Index	Distance	Position	Type	Homogeneous Medium	Comment
1	0 mm	0 mm	Programmable Interf...	Air in Homogeneous...	Enter

Edit Programmable Interface - Structure Tab

Interface Specification

Algorithms

Snippet for Height Profile Edit Validity: ✓

Numerical Gradient Calculation Accuracy Factor:

User-Defined Gradient Calculation

Definition Area

Size and Shape

Shape: Rectangular Elliptic

Size: x

Effect on Field Outside of Definition Area

Field Passes Plane Interface

Field is Absorbed

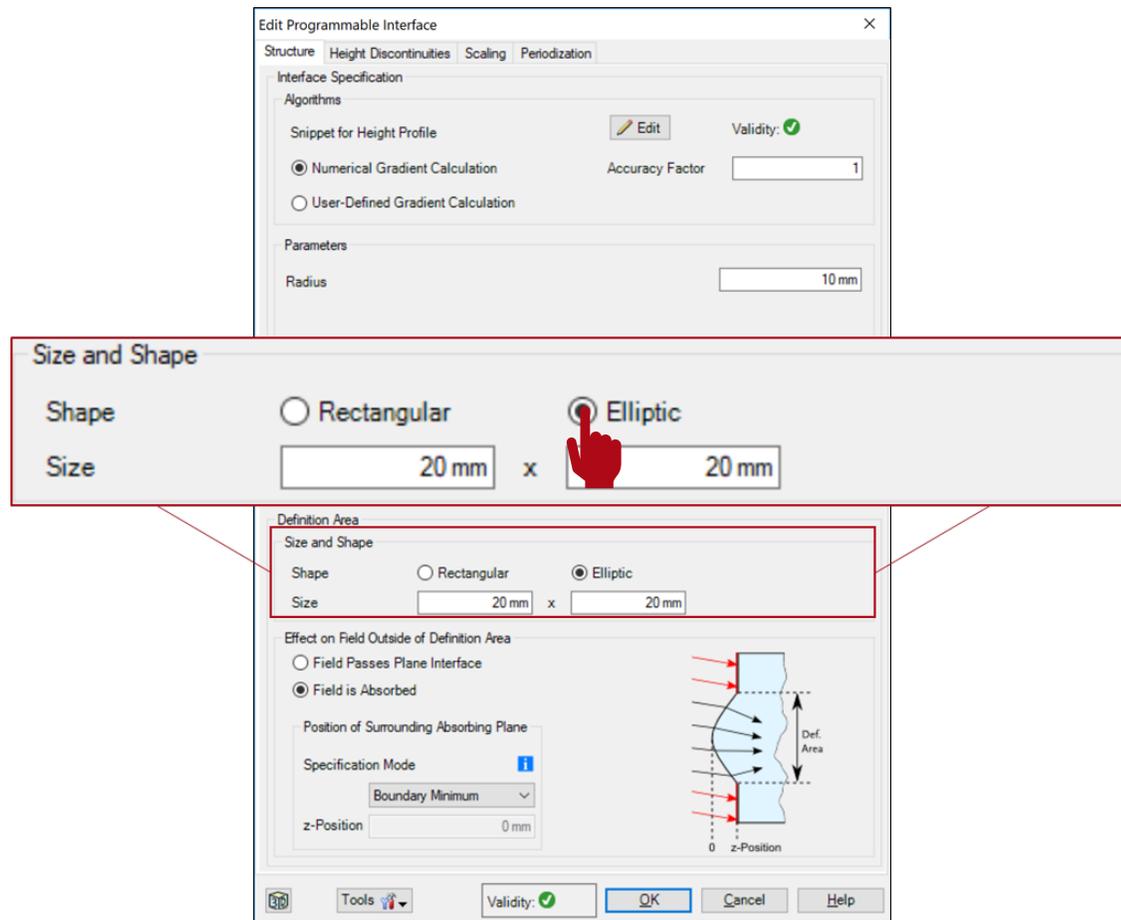
Position of Surrounding Absorbing Plane

Specification Mode:

z-Position:

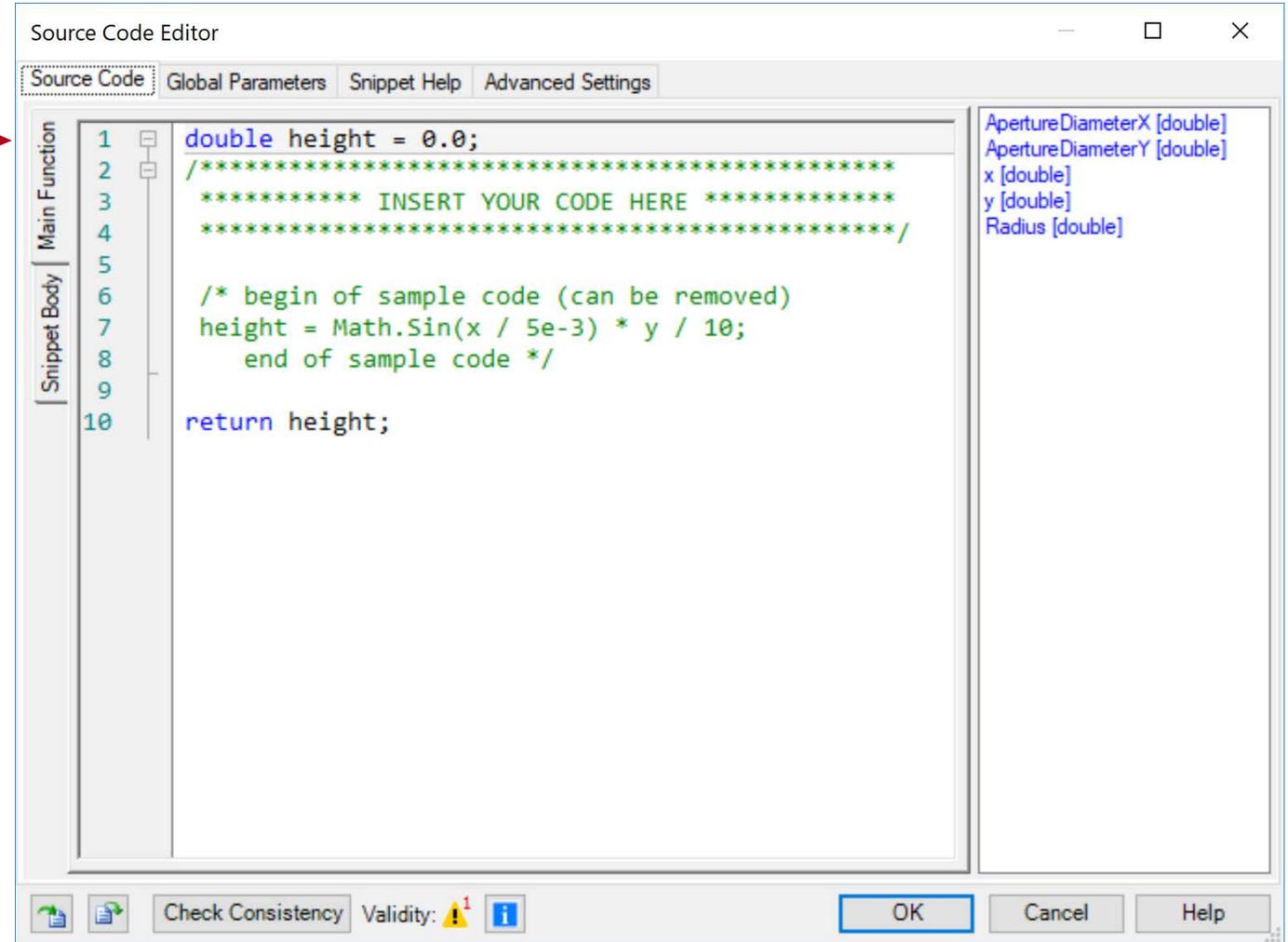
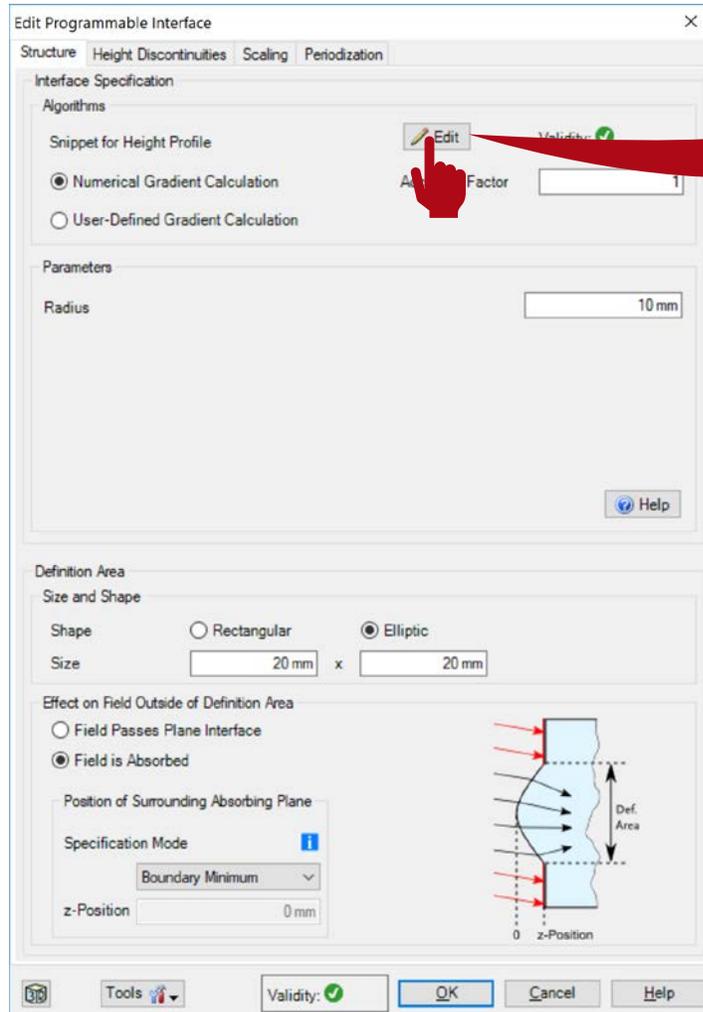
The diagram shows a rectangular definition area (Def. Area) with light rays entering from the left. The z-axis is shown at the bottom, with the origin (0) at the left edge of the area.

Setting Up the Area of Definition of the Surface



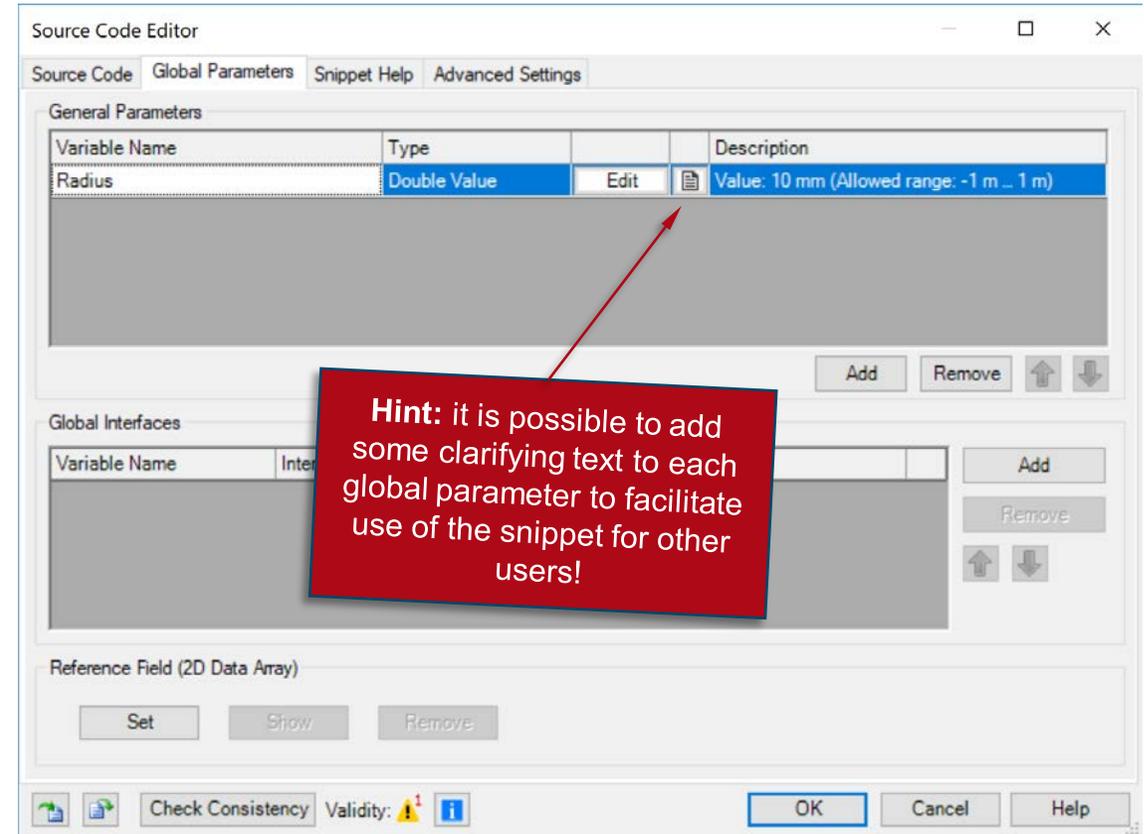
- In the configuration dialog previous to entering the programming interface, the user can define the shape and size of the area of definition (parametrization) of the surface.
- In this example we use a circular aperture.
- Bear in mind that there is a natural limit to the area of definition of a spherical surface, given by its diameter, outside of which the surface is not defined!

Entering the Programming Interface

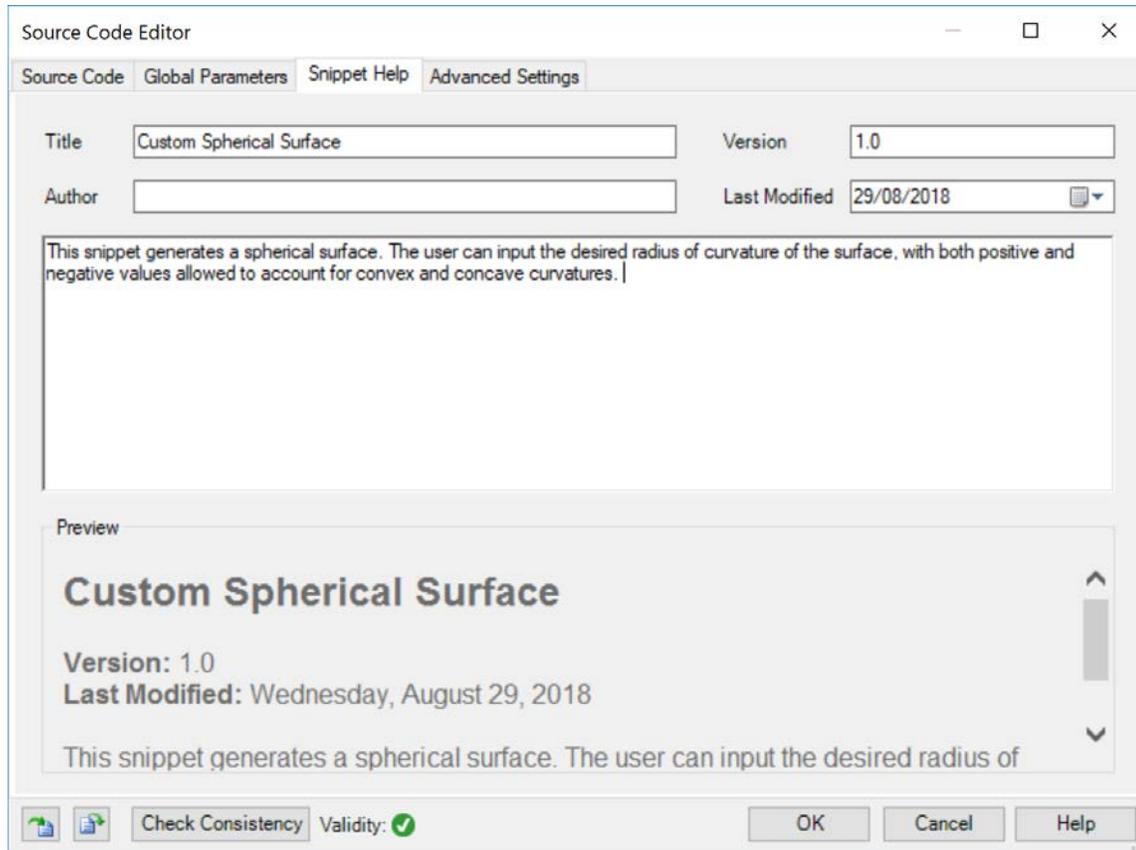


Programmable Interface: Global Parameters

- Once you have triggered open the Edit dialog (Source Code Editor), go to the Global Parameters tab.
- There, Add and Edit one parameter:
 - `double` Radius = 10 mm (-1 m, 1 m):
the radius of curvature of the spherical surface.
- Use the button with the small “notes” icon to add some explanation to your custom global parameters.



Programmable Interface: Snippet Help



- **Optional:** you can use the Snippet Help tab to write instructions, clarifications, and some additional data 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 Interface: Snippet Help

Source Code Editor

Source Code Global Parameters Snippet Help Advanced Settings

Title: Custom Spherical Surface Version: 1.0

Author:

This snippet generates a spherical surface. The user can input negative values allowed to account for convex and concave surfaces.

Preview

Custom Spherical Surface

Version: 1.0
Last Modified: Wednesday, August 29, 2018

This snippet generates a spherical surface.

Check Consistency Validity: ✓

Edit Programmable Interface

Structure Height Discontinuities Scaling Periodization

Interface Specification

Algorithms

Snippet for Height Profile Edit Validity: ✓

Numerical Gradient Calculation

User-Defined Gradient Calculation Edit Validity: ✓

Parameters

Radius: 10 mm

Definition Area

Size and Shape

Shape: Rectangular Elliptic

Size: 20 mm x 20 mm

Effect on Field Outside of Definition Area

Field Passes Plane Interface

Field is Absorbed

Position of Surrounding Absorbing Plane

Specification Mode: Boundary Minimum

z-Position: -9.99999974 mm

Help

Snippet Help

Custom Spherical Surface

Version: 1.0
Last Modified: Wednesday, August 29, 2018

This snippet generates a spherical surface. The user can input the desired radius of curvature, with both positive and negative values allowed to account for convex and concave surfaces.

PARAMETER	DESCRIPTION
Radius	The radius of curvature of the spherical surface. It can be positive or negative to define a convex or a concave surface respectively.

Close

Programmable Interface: Writing the Code

Declaration of output variable given by default

Source Code Editor

Source Code Global Parameters Snippet Help Advanced Settings

```
1 double height = 0.0;
2
3 height = (Radius / Radius.Abs()) * // Use correct sign. } Eq. (2)
4   Math.Sqrt((Radius * Radius) - (x * x) - (y * y));
5 height = height - Radius; // Offset surface so that central point is at zero height.
6
7 // Hint: an alternative way to compute the sign is to use
8 // MathFunctions.Sign(Radius) instead of (Radius / Radius.Abs()).
9
10
11 return height;
```

ApertureDiameterX [double]
ApertureDiameterY [double]
x [double]
y [double]
Radius [double]

Check Consistency Validity:

OK Cancel Help

Default global parameters/variables

Global parameter defined by user in Global Parameters tab

Offset the height function so that $h(0, 0) = 0$. Even if this is not implemented in the code, the surface will still be automatically offset to fulfil this condition!

Are there errors in your code?

Export Snippet to save your work!

Programmable Interface: User-Defined Gradient

The image displays two windows from a software application. The left window, titled "Edit Programmable Interface", shows the "User-Defined Gradient Calculation" option selected under the "Algorithms" section. A red hand icon with the number "1" points to this option, and another red hand icon with the number "2" points to the "Edit" button next to it. A red arrow points from this button to the right window.

The right window, titled "Source Code Editor", shows the following C# code in the "Main Function" section:

```
1 VectorD gradient = new VectorD();
2
3 gradient.X = -(Radius / Radius.Abs()) * (x) /
4   (Math.Sqrt((Radius * Radius) - (x * x) - (y * y)));
5 gradient.Y = -(Radius / Radius.Abs()) * (y) /
6   (Math.Sqrt((Radius * Radius) - (x * x) - (y * y)));
7
8 // Hint: an alternative way to compute the sign is to use
9 // MathFunctions.Sign(Radius) instead of (Radius / Radius.Abs()).
10
11 return gradient;
12
```

A blue bracket on the right side of the code groups lines 3 and 4, with a blue arrow pointing to a parameter list on the right:

```
ApertureDiameterX [double]
ApertureDiameterY [double]
x [double]
y [double]
Radius [double]
```

The parameter list is labeled "Eq. (3)".

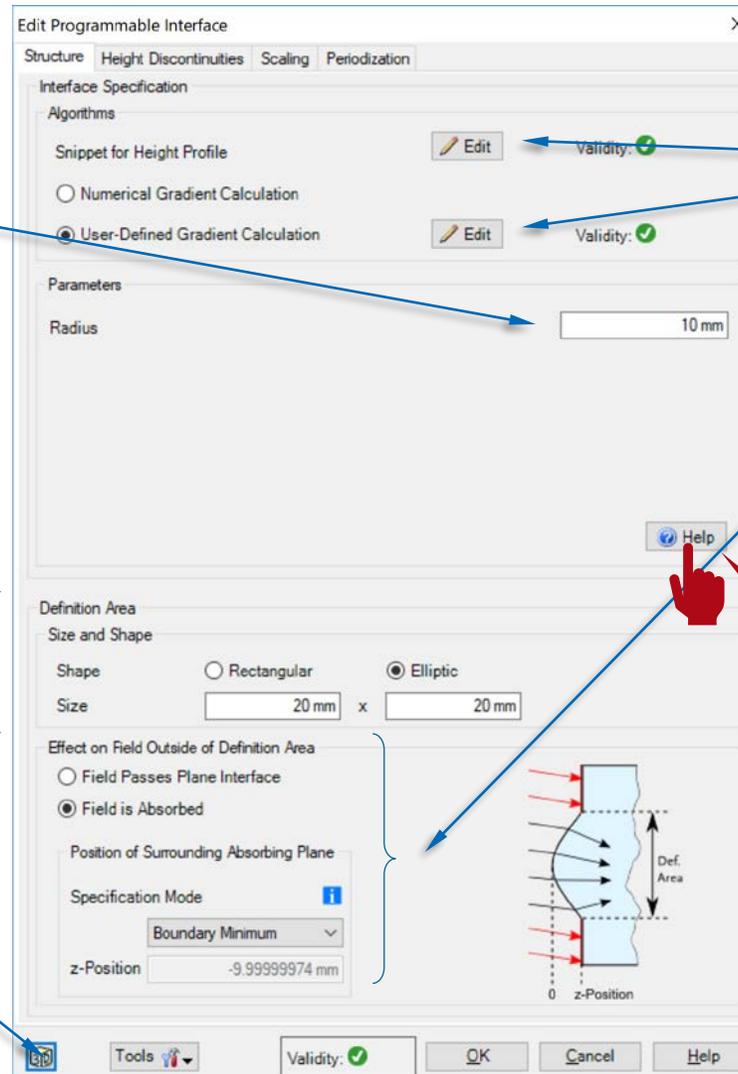
At the bottom of the "Source Code Editor" window, there is a "Check Consistency" button, a "Validity: ✓" indicator, and "OK", "Cancel", and "Help" buttons.

Programmable Interface: Using Your Snippet

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

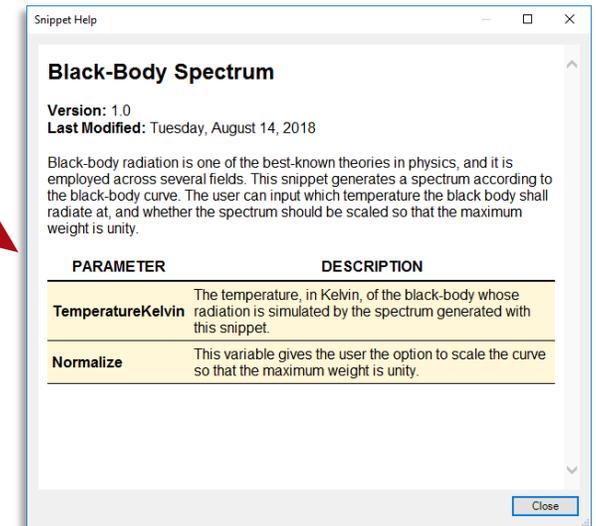
You can modify the area of definition (aperture) of your surface. Bear in mind possible constraints (as already pointed out in the case of the spherical surface).

You can view the surface you have defined here

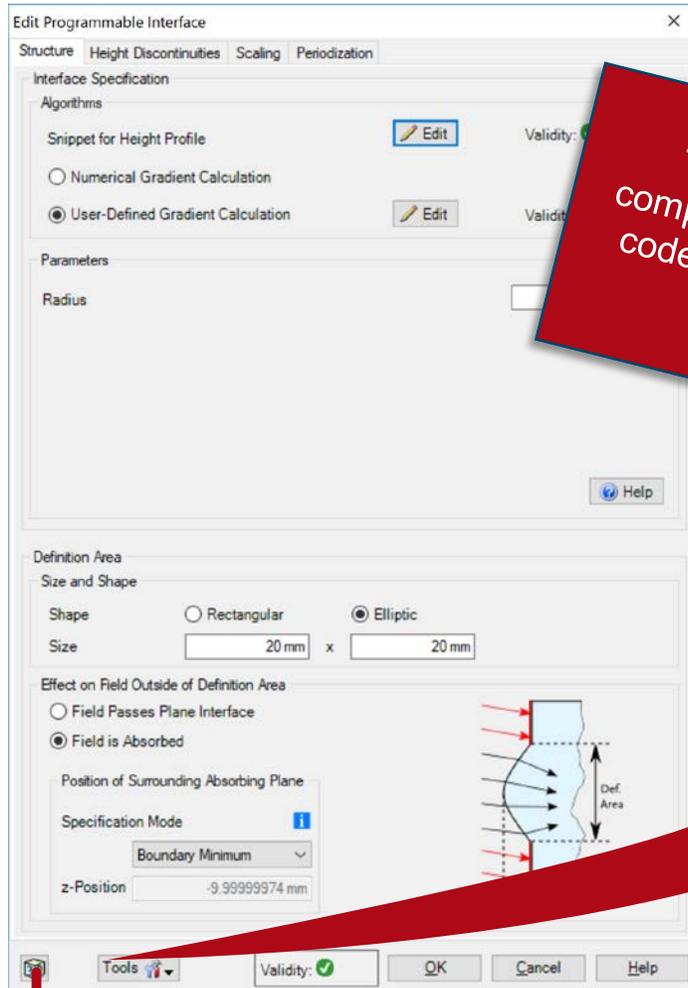


Modify your snippet by clicking on *Edit*

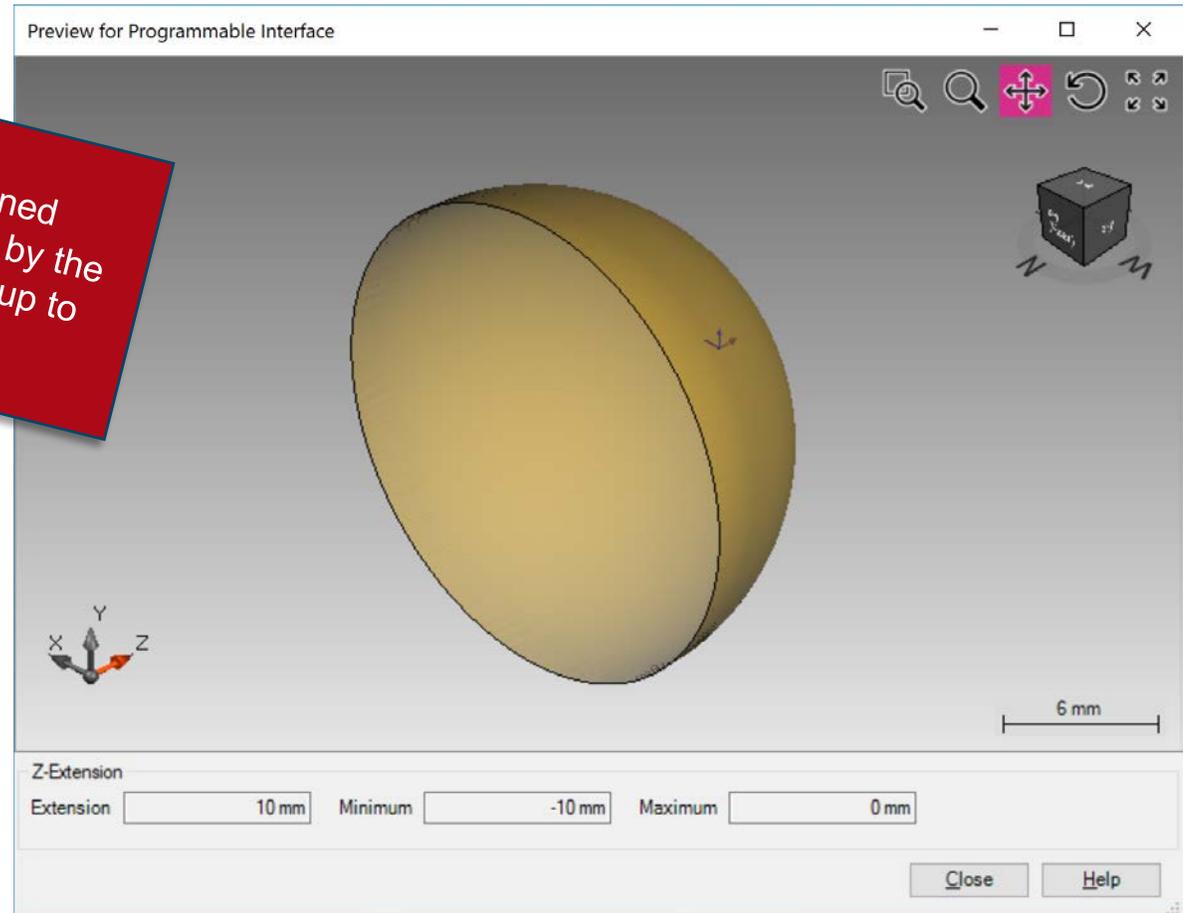
With respect to which reference point in the surface is the surrounding plane defined? And what happens when a part of the EM field meets this surrounding plane?



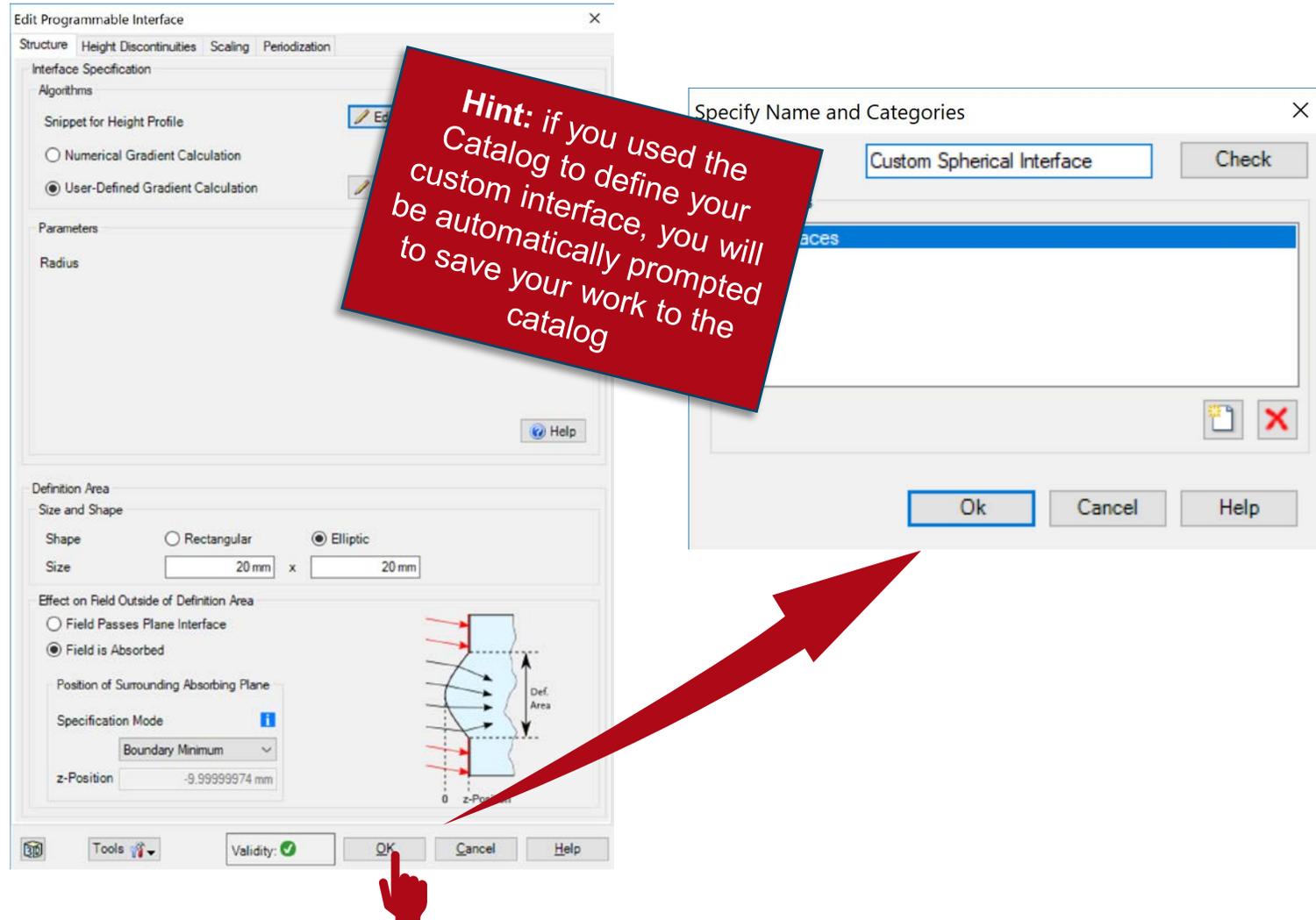
Output of Programmable Interface



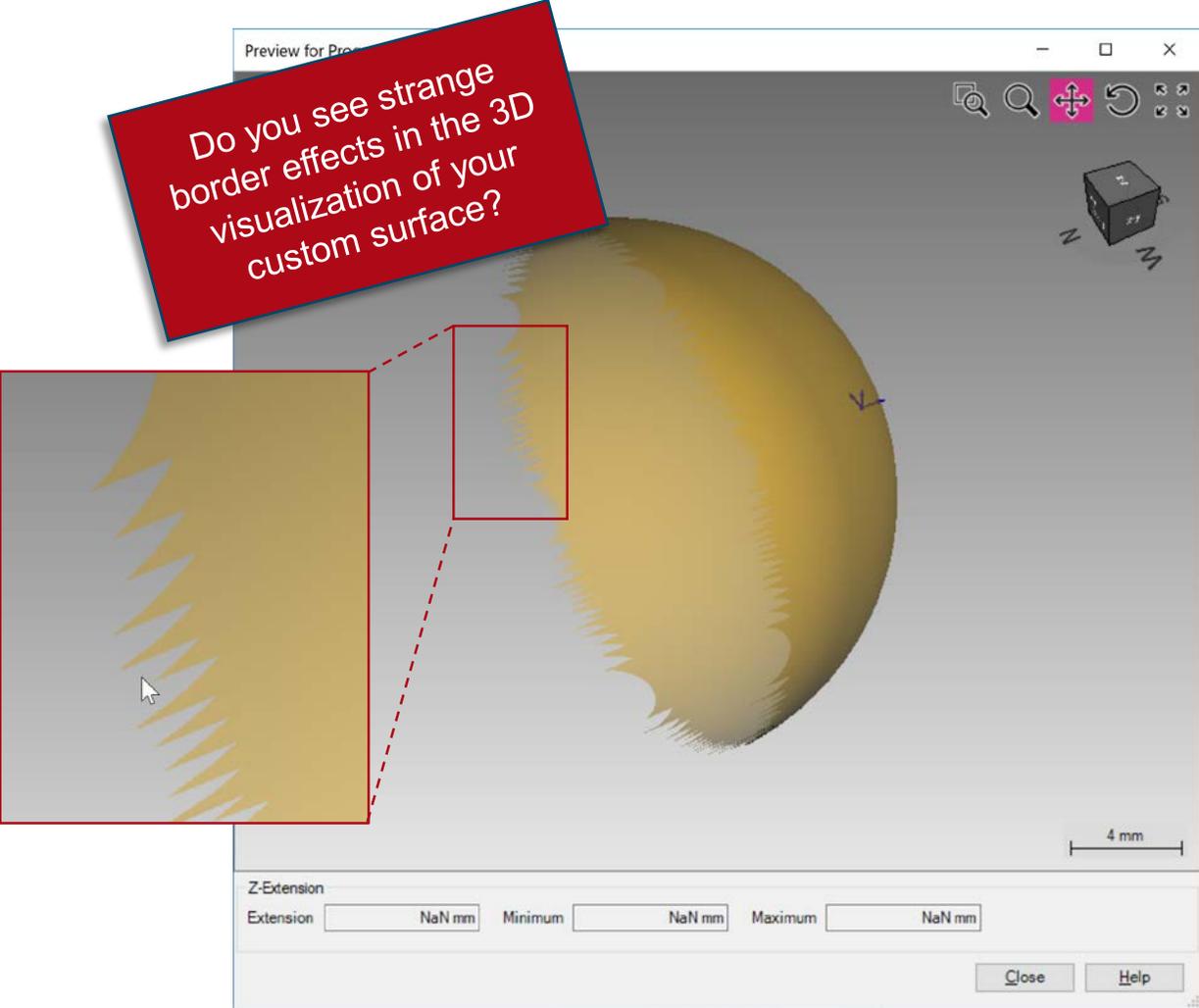
The surface is defined completely analytically by the code—**full accuracy** (up to double precision)



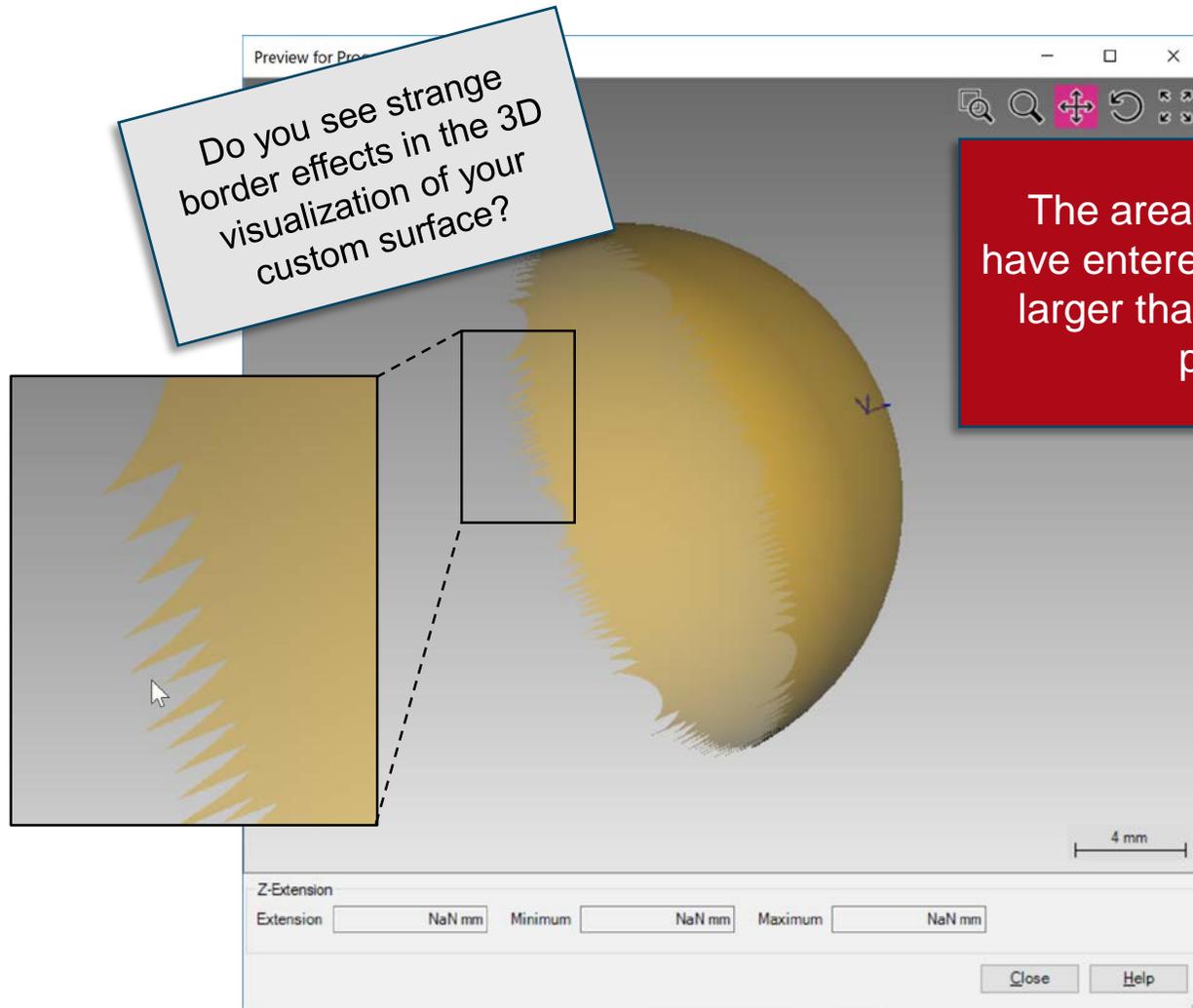
Saving the Custom Interface to the Catalog



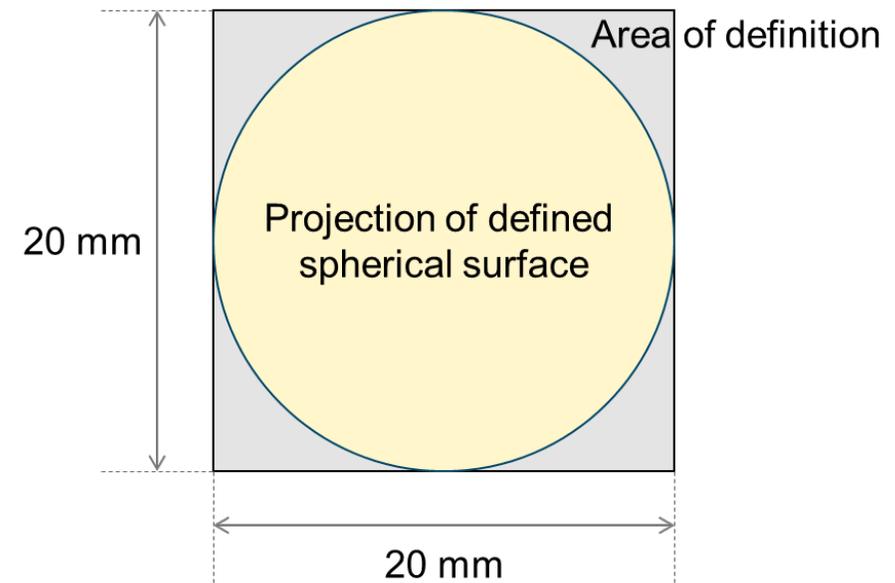
Troubleshooting Tips



Troubleshooting Tips



The area of definition you have entered for the surface is larger than mathematically possible!



Troubleshooting Tips

Do you see strange border effects in the 3D visualization of your custom surface?

The area of definition you have entered for the surface is larger than mathematically possible!

In this particular example, make sure you have selected an "Elliptic" shape, or reduce the size of the area of definition .

Definition Area

Size and Shape

Shape Rectangular Elliptic

Size x

Effect on Field Outside of Definition Area

Field Passes Plane Interface

Field is Absorbed

Size and Shape

Shape Rectangular Elliptic

Size x

Z-Extension

Extension Minimum Maximum

Close Help

Validity:

OK Cancel Help

Test the Code!

Main Function (Height Profile)

```
double height = 0.0;

height = (Radius / Radius.Abs()) * // Use correct sign.
    Math.Sqrt((Radius * Radius) - (x * x) - (y * y));
height = height - Radius; // Offset surface so that central point is at zero height.

// Hint: an alternative way to compute the sign is to use
// MathFunctions.Sign(Radius) instead of (Radius / Radius.Abs()).

return height;
```

Test the Code!

Main Function (Gradient)

```
VectorD gradient = new VectorD();

gradient.X = -(Radius / Radius.Abs()) * (x) /
    (Math.Sqrt((Radius * Radius) - (x * x) - (y * y)));
gradient.Y = -(Radius / Radius.Abs()) * (y) /
    (Math.Sqrt((Radius * Radius) - (x * x) - (y * y)));

// Hint: an alternative way to compute the sign is to use
// MathFunctions.Sign(Radius) instead of (Radius / Radius.Abs()).

return gradient;
```

Document Information

title	How to Work with the Programmable Interface & Example (Spherical Surface)
document code	CZT.0096
version	1.0
toolbox(es)	Starter Toolbox
VL version used for simulations	7.4.0.49
category	Feature Use Case
further reading	<ul style="list-style-type: none">- Customizable Help for Programmable Elements- Programmable Light Source, Function, Interface and Medium- Programming a Sinusoidal Surface