How to debug a DLL or EXE

Print

Views:

Authored By

Michael Humphreys, Sandrine Auriol and Tim Gustafson

Abstract

This article shows two techniques to debug a User Defined DLL or a User Defined EXE.
- The first one is to write "debugged" values in a text file.
- The second one is to attach the Visual Studio debugger to a running process. It is shown here using Microsoft Visual Studio Community 2017.

Contents

Downloads

Article Attachments (https://downloads.zemax.com/Zemax-portal/knowledge_articles/KA-01636/Downloads/Lambertian_DebugText.zip) (http://downloads.zemax.com/Zemax-portal/knowledge_articles/KA-01636/Downloads/Lambertian_DebugText.zip)

This article shows how to debug User Defined DLL or EXE created in Visual Studio in C, C++ or C#. The steps to create those User Defined features are not detailed here.

Debug using a text file

One way to debug a User Defined DLL or a User Defined EXE is to write "debugged" values in a text file. Let's add some debugging lines to the user-defined scattering function {Zemax}\DLL\SurfaceScatter\Lambertian.c. The modified file called Lambertian_DebugText.cpp can be found in the attachments.

First, to write in a text file, the following libraries are added at the top of the code:

```
 #include <fstream>
#include <sstream>
#include <iostream>
#include <shlobj.h>
using namespace std;
```

Then a "DebugBool" parameter is defined:

```
 int __declspec(dllexport) APIENTRY UserParamNames(char *data)
    {
    ...
    //debug
    if (i == 3) strcpy(data, "DebugBool");
    return 0;
    }
```

Depending on the value of the "DebugBool" parameter, a file is created and values can be written in that file for further analysis:

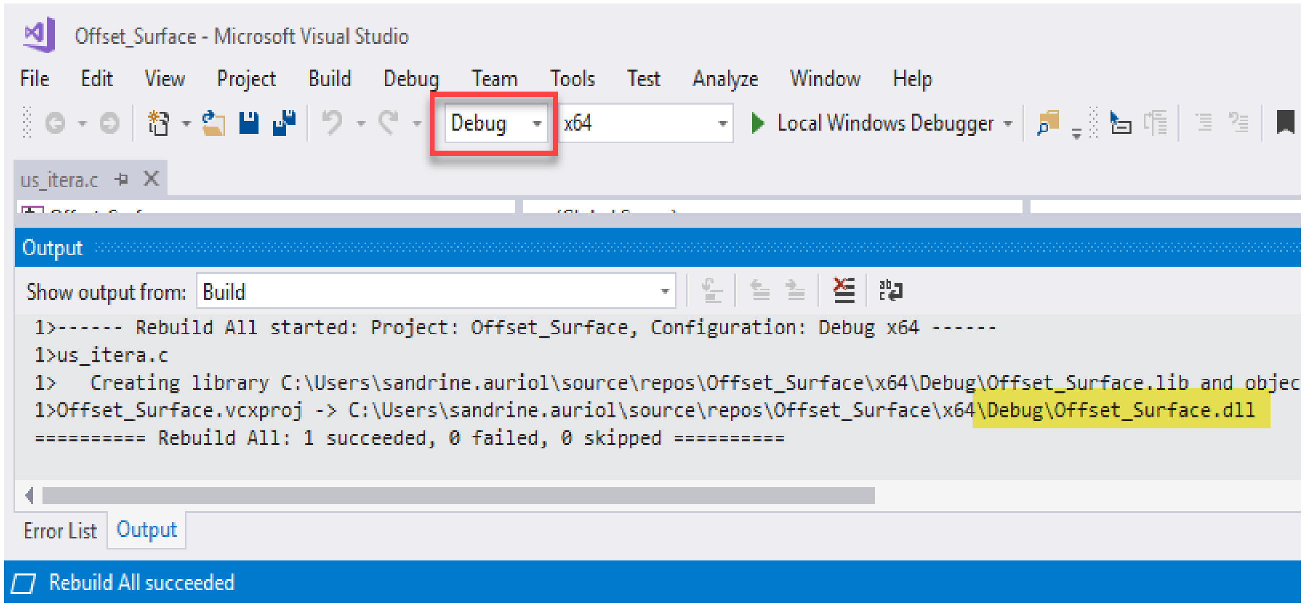
```
      bool debug = data[54] != 0;

      ...

      CHAR my_documents[MAX_PATH];
     fstream debuglog;
     string filepath;
     if (debug)
     {
         SHGetFolderPath(NULL, CSIDL_PERSONAL, NULL, SHGFP_TYPE_CURRENT, my_documents);
         filepath = string(my_documents) + "\\Zemax\\DLL\\SurfaceScatter\\debug.txt";
         debuglog.open(filepath, ios::out | ios::app); // debuglog.close();
         debuglog.precision(12);
     }

      ...

      debuglog << debugged_value << "\t";
     debuglog.close();
```

As can be seen, writing a text file to debug is an easy method.

Debug using Visual Studio debugger

Compile the DLL or EXE in Debug Mode

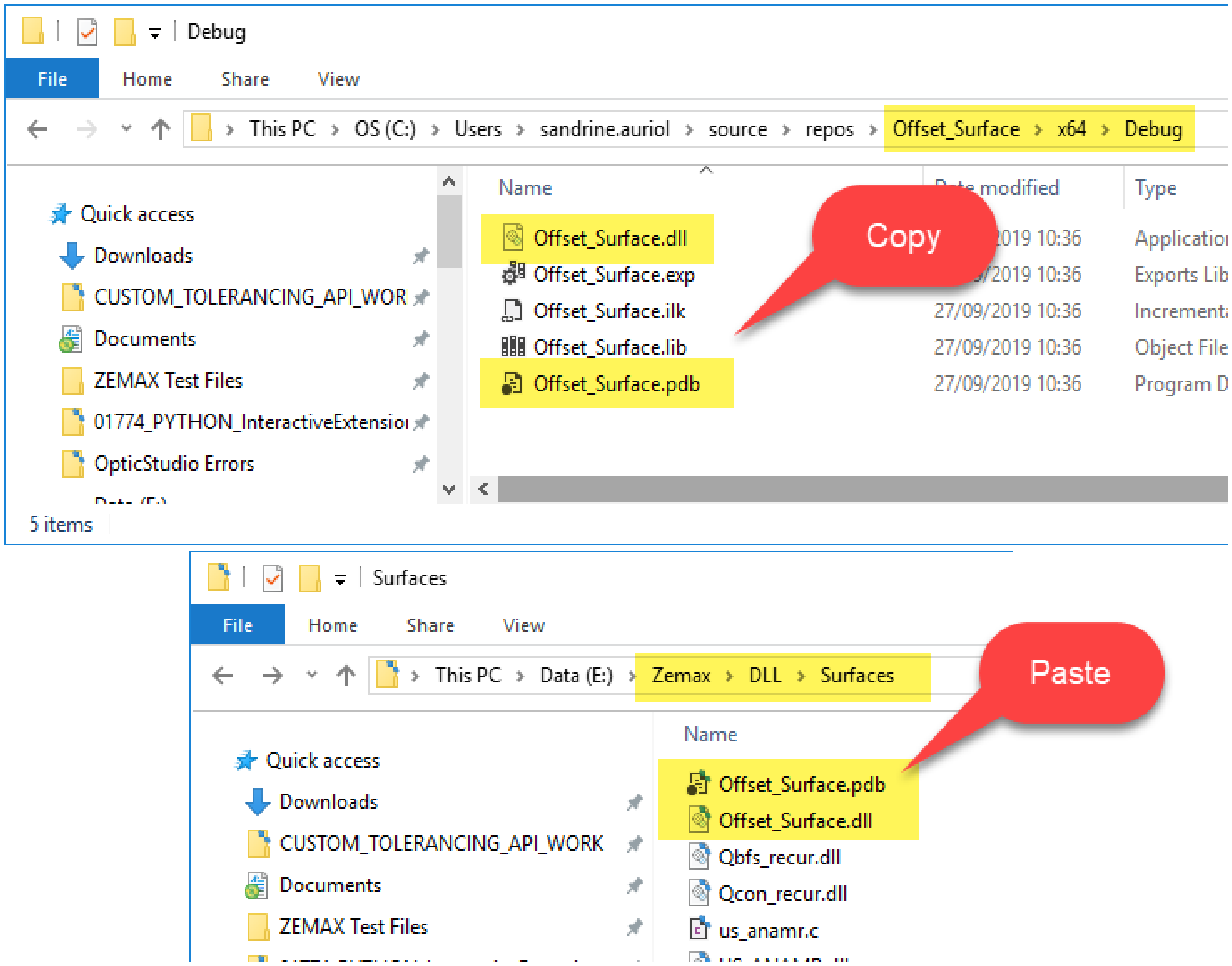
Let's see how to compile a DLL or EXE in Debug Mode. For example, let's compile a user defined surface (DLL). For more details, see the article "How to compile a User-Defined DLL" (~/Knowledge-Base/kb-article/?ka=KA-01787).


Offset_Surface - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug x64 Local Windows Debugger
us_itera.c
Output
Show output from: Build
Error List Output

```
1>------ Rebuild All started: Project: Offset_Surface, Configuration: Debug x64 ------
1>us_itera.c
1>   Creating library C:\Users\sandrine.auriol\source\repos\Offset_Surface\x64\Debug\Offset_Surface.lib and objec
1>Offset_Surface.vcxproj -> C:\Users\sandrine.auriol\source\repos\Offset_Surface\x64\Debug\Offset_Surface.dll
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

Rebuild All succeeded

Once compiled, 2 files need to be copied from Project\x64\Debug to {Zemax}\DLL\Surfaces: the DLL and the PDB file. Normally (when using the DLL in release mode), only the DLL file is copied. But the PDB file (Program database) is needed here because it stores debugging information about programs.


Debug
File Home Share View
This PC › OS (C:) › Users › sandrine.auriol › source › repos › Offset_Surface › x64 › Debug
Quick access
Downloads
CUSTOM_TOLERANCING_API_WOR
Documents
ZEMAX Test Files
01774_PYTHON_InteractiveExtensio
OpticStudio Errors
Copy
5 items
Surfaces
File Home Share View
This PC › Data (E:) › Zemax › DLL › Surfaces
Quick access
Downloads
CUSTOM_TOLERANCING_API_WORK
Documents
ZEMAX Test Files
Paste

Name	Date modified	Type
Offset_Surface.dll	27/09/2019 10:36	Applicatio
Offset_Surface.exp	27/09/2019 10:36	Exports Lib
Offset_Surface.ilk	27/09/2019 10:36	Increment
Offset_Surface.lib	27/09/2019 10:36	Object File
Offset_Surface.pdb	27/09/2019 10:36	Program D

Name
Offset_Surface.pdb
Offset_Surface.dll
Qbfs_recur.dll
Qcon_recur.dll
us_anamr.c

Every time the DLL is modified and recompiled, it will have to be manually copied again from Project\x64\Debug to {Zemax}\DLL\Surfaces.

To avoid this, the "Project\x64\Debug" output folder can be changed to {Zemax}\DLL\Surfaces:

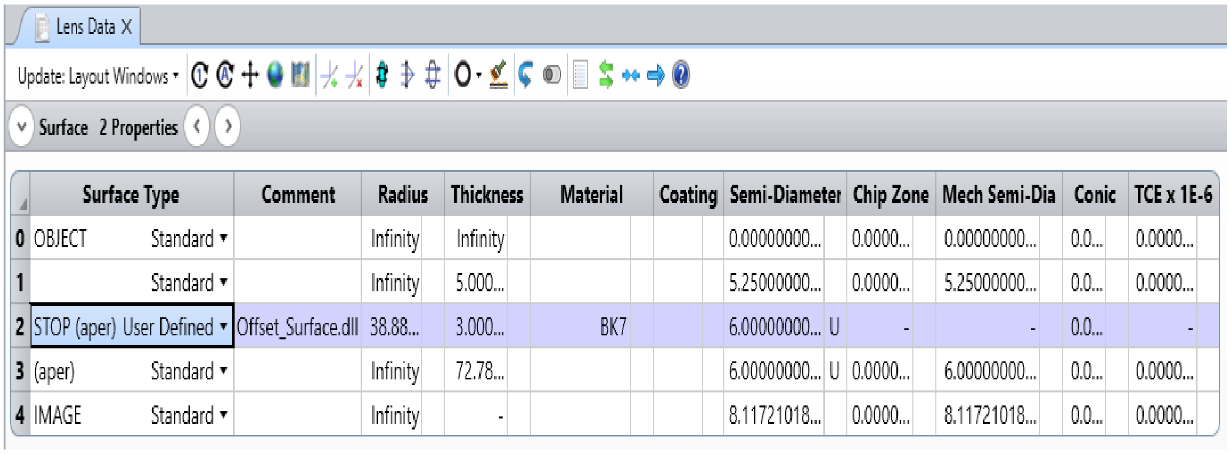
In C++: in Visual Studio, go to Solution Explorer. Select the project. Right-click, go to Properties. The project Property pages opens. Under Configuration Properties, select General. Change the Target Name.

In C#: in Visual Studio, go to Project Preferences > Build > Output directory.

Back to OpticStudio

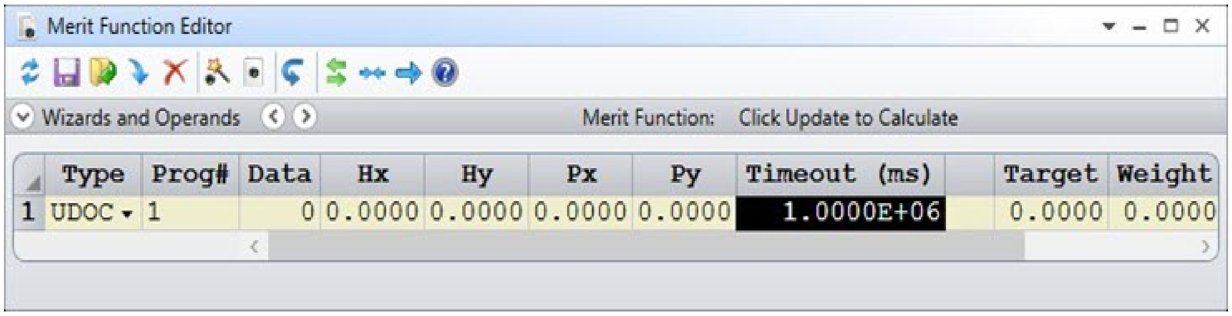
Open OpticStudio and select the User Defined Surface or Operand.

- User Defined Surface DLL: change Surface 2 to a User Defined Surface. Surface 2 is using the DLL in Debug mode:



	Surface Type		Comment	Radius	Thickness	Material	Coating	Semi-Diameter	Chip Zone	Mech Semi-Dia	Conic	TCE x 1E-6
0	OBJECT	Standard ▾		Infinity	Infinity			0.00000000...	0.0000...	0.00000000...	0.0...	0.0000...
1		Standard ▾		Infinity	5.000...			5.25000000...	0.0000...	5.25000000...	0.0...	0.0000...
2	STOP (aper)	User Defined ▾	Offset_Surface.dll	38.88...	3.000...	BK7		6.00000000... U	-	-	0.0...	-
3	(aper)	Standard ▾		Infinity	72.78...			6.00000000... U	0.0000...	6.00000000...	0.0...	0.0000...
4	IMAGE	Standard ▾		Infinity	-			8.11721018...	0.0000...	8.11721018...	0.0...	0.0000...

- User Defined Operand UDOC: select the UDOC operand in the Merit Function.
  Make sure to set the Timeout (ms) to a high value so that the UDOC can be debugged in Visual Studio without having OpticStudio throw a timeout error.



	Type	Prog#	Data	Hx	Hy	Px	Py	Timeout (ms)	Target	Weight
1	UDOC ▾	1	0	0.0000	0.0000	0.0000	0.0000	1.0000E+06	0.0000	0.0000

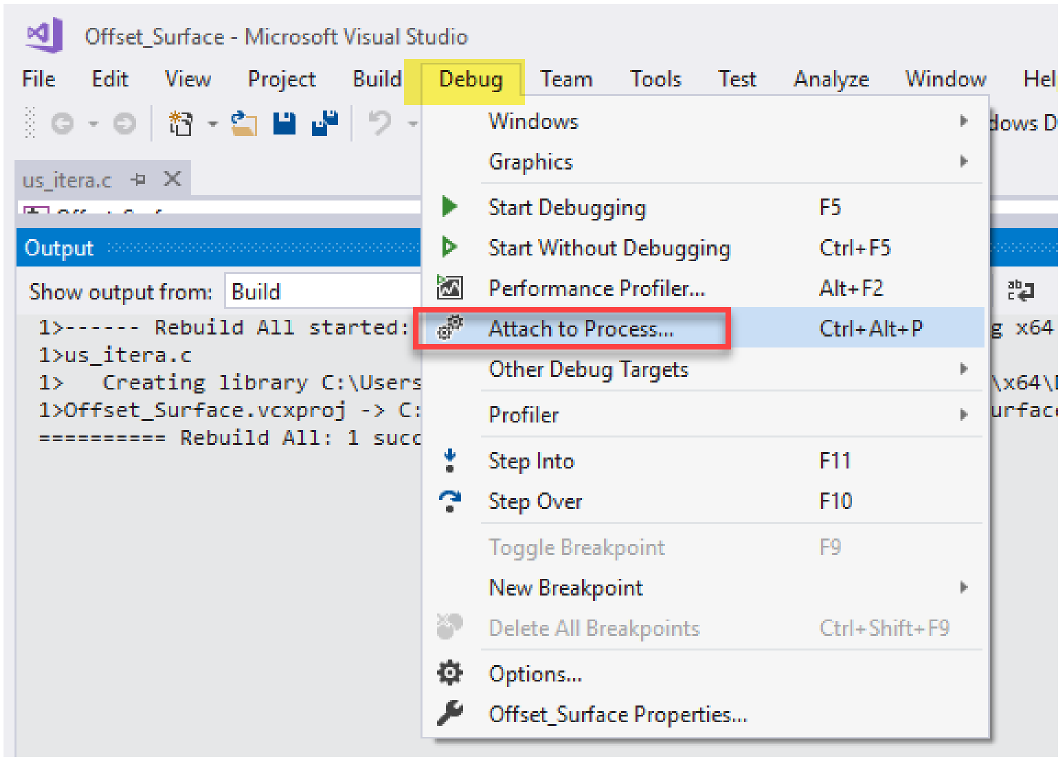
At this point, do not update the system or the Merit Function yet. An update will execute the user defined surface or UDOC. If there are errors in the code, it may cause a crash.

Attach to Process

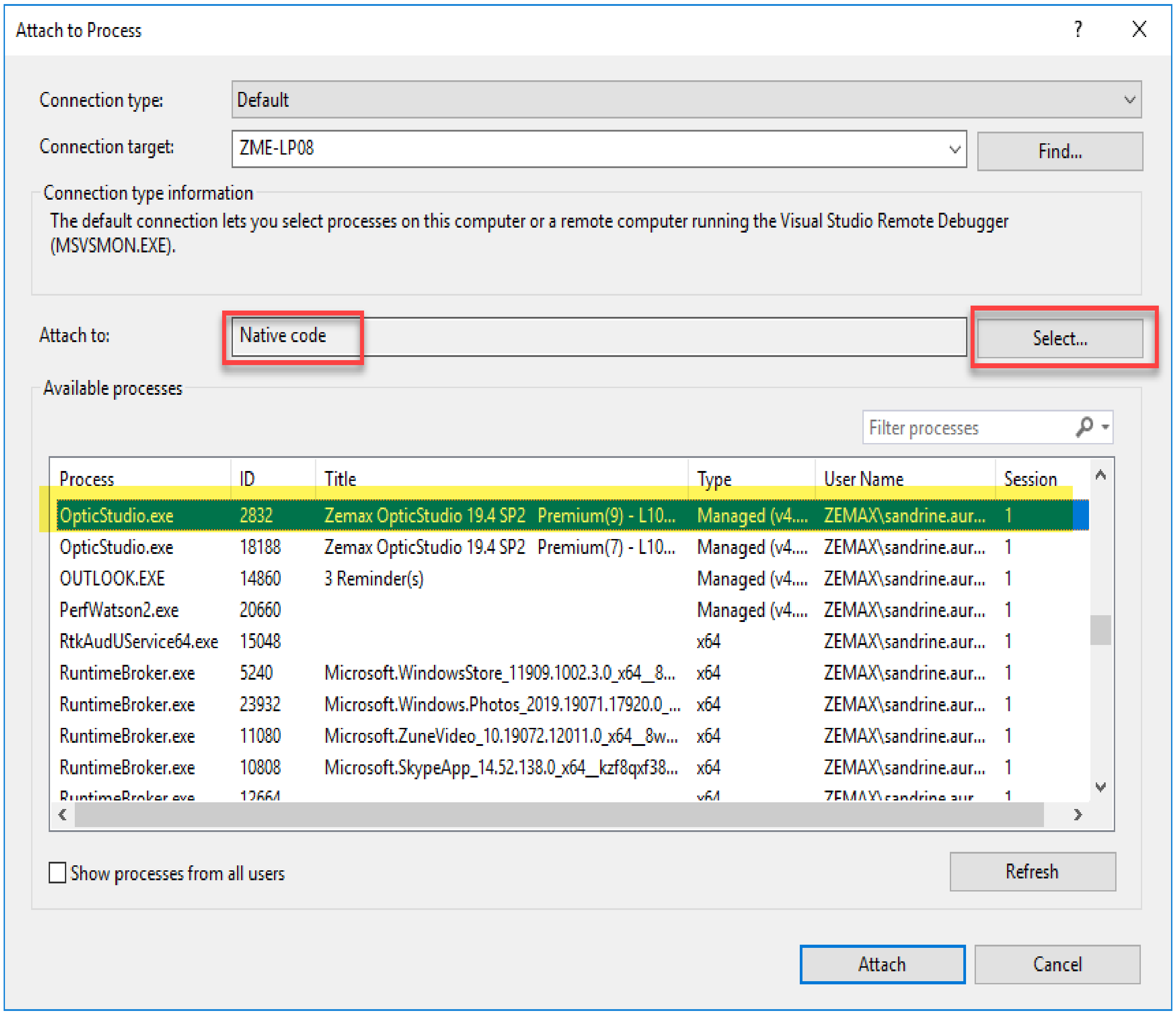
Back to Visual Studio, everything is ready to start debugging. If the user extension is a DLL, the debugger can be attached to the OpticStudio.exe process. If the user extension is an EXE, the debugger can be attached to our application process.

DLL Method – Attach to OpticStudio

The debugger can be attached to the OpticStudio running process :



```
1>------ Rebuild All started:
1>us_itera.c
1>   Creating library C:\Users
1>Offset_Surface.vcxproj -> C:
========== Rebuild All: 1 succ
```



Connection type: Default

Connection target: ZME-LP08

Connection type information
The default connection lets you select processes on this computer or a remote computer running the Visual Studio Remote Debugger (MSVSMON.EXE).

Attach to: Native code

Available processes

Process	ID	Title	Type	User Name	Session
OpticStudio.exe	2832	Zemax OpticStudio 19.4 SP2 Premium(9) - L10...	Managed (v4....	ZEMAX\sandrine.aur...	1
OpticStudio.exe	18188	Zemax OpticStudio 19.4 SP2 Premium(7) - L10...	Managed (v4....	ZEMAX\sandrine.aur...	1
OUTLOOK.EXE	14860	3 Reminder(s)	Managed (v4....	ZEMAX\sandrine.aur...	1
PerfWatson2.exe	20660		Managed (v4....	ZEMAX\sandrine.aur...	1
RtkAudUService64.exe	15048		x64	ZEMAX\sandrine.aur...	1
RuntimeBroker.exe	5240	Microsoft.WindowsStore_11909.1002.3.0_x64__8...	x64	ZEMAX\sandrine.aur...	1
RuntimeBroker.exe	23932	Microsoft.Windows.Photos_2019.19071.17920.0_...	x64	ZEMAX\sandrine.aur...	1
RuntimeBroker.exe	11080	Microsoft.ZuneVideo_10.19072.12011.0_x64__8w...	x64	ZEMAX\sandrine.aur...	1
RuntimeBroker.exe	10808	Microsoft.SkypeApp_14.52.138.0_x64__kzf8qxf38...	x64	ZEMAX\sandrine.aur...	1
RuntimeBroker.exe	12664		x64	ZEMAX\sandrine.aur	1

☐ Show processes from all users

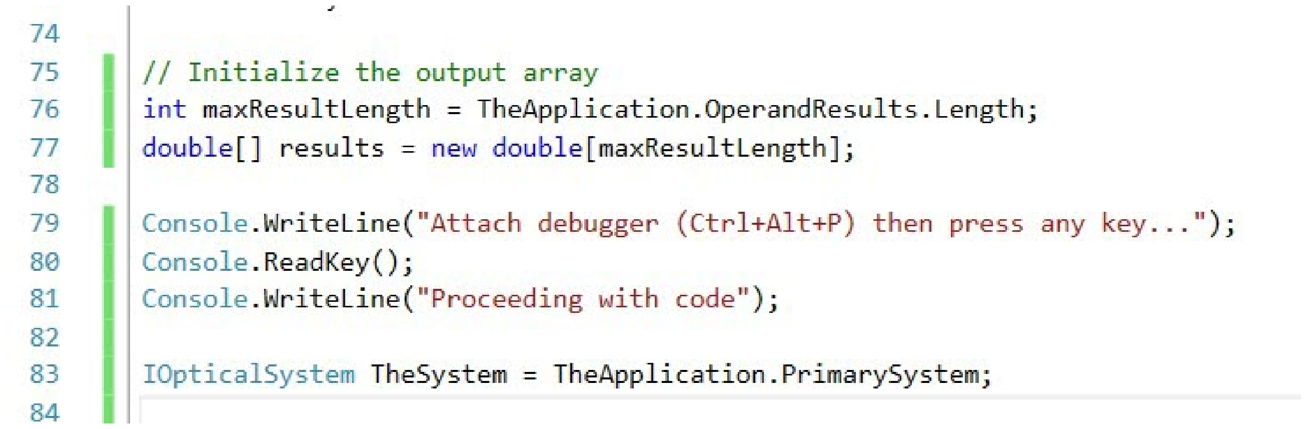
The Attach to box will select the type of code to be debugged.

- In C++ it will be **Native code**
- In C# it will be **Managed code**

In Available Processes, make sure to select the appropriate OpticStudio process. It may be easier to have only one OpticStudio instance open.

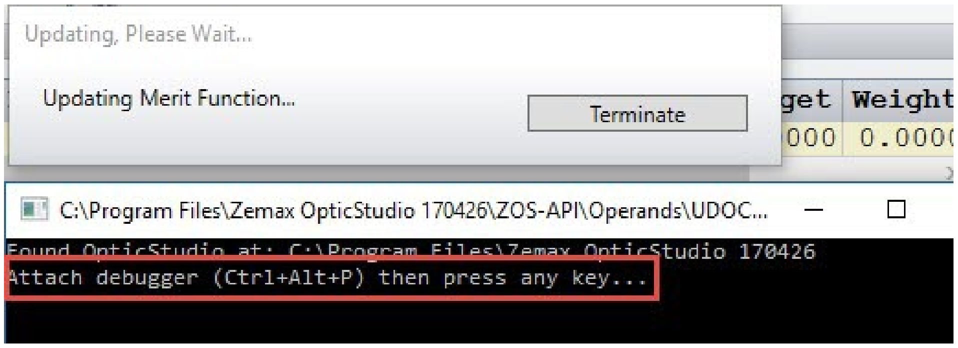
EXE Method – Attach Directly To UDOC

To debug a UDOC, the UDOC application itself is attached to the process rather than OpticStudio. Therefore, a “pause” is set in the UDOC code. It can be a `Console.ReadKey()` for C# or `system(“pause”)` for C++ to pause the actual execution.
In C#, the code might look like below:



```
// Initialize the output array
int maxResultLength = TheApplication.OperandResults.Length;
double[] results = new double[maxResultLength];

Console.WriteLine("Attach debugger (Ctrl+Alt+P) then press any key...");
Console.ReadKey();
Console.WriteLine("Proceeding with code");

IOpticalSystem TheSystem = TheApplication.PrimarySystem;
```

If the Merit Function is updated, a Command Line Window will open and display a message :

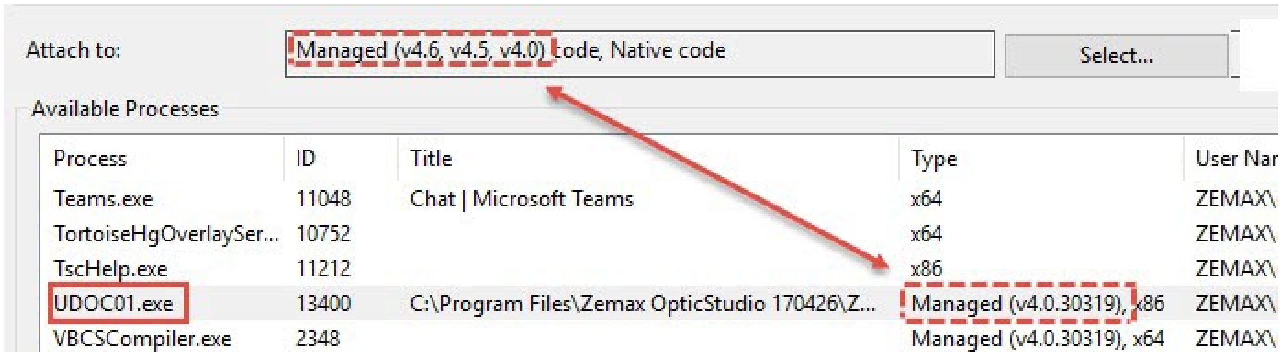


Updating, Please Wait...

Updating Merit Function...

```
C:\Program Files\Zemax OpticStudio 170426\ZOS-API\Operands\UDOC...
Found OpticStudio at: C:\Program Files\Zemax OpticStudio 170426
Attach debugger (Ctrl+Alt+P) then press any key...
```

Now, the UDOC can be attached to the debugger. Make sure to have the correct code selected for the UDOC. Below, "Managed code" is selected as the code is in C++.



Attach to: Managed (v4.6, v4.5, v4.0) code, Native code

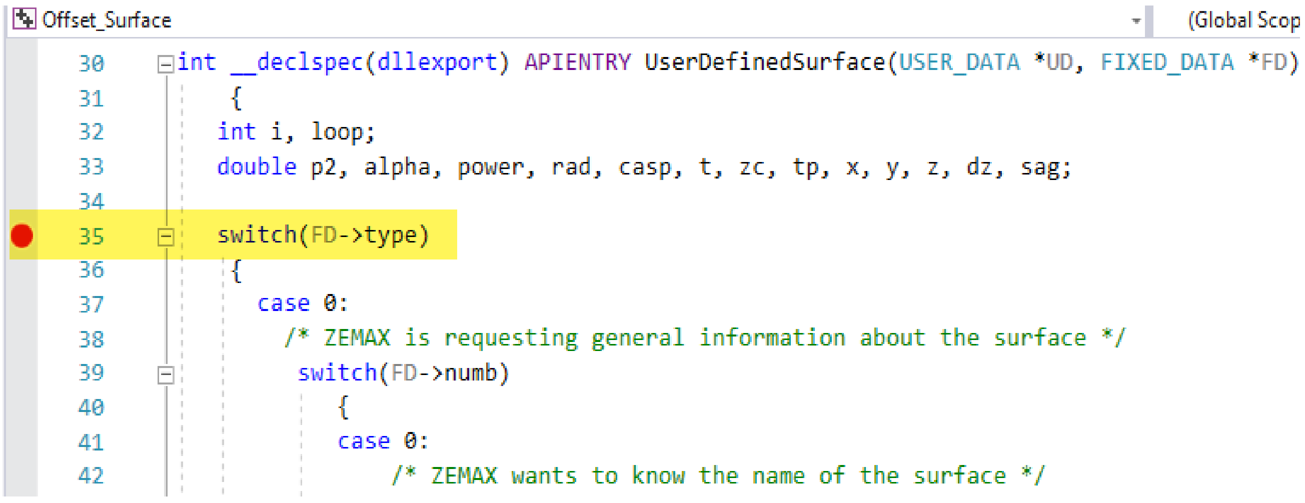
Available Processes

Process	ID	Title	Type	User Nar
Teams.exe	11048	Chat | Microsoft Teams	x64	ZEMAX\
TortoiseHgOverlaySer...	10752		x64	ZEMAX\
TscHelp.exe	11212		x86	ZEMAX\
UDOC01.exe	13400	C:\Program Files\Zemax OpticStudio 170426\Z...	Managed (v4.0.30319), x86	ZEMAX\
VBCSCompiler.exe	2348		Managed (v4.0.30319), x64	ZEMAX\

Note that with a UDOC, the process needs to be attached every time the Merit Function is updated.

Set Breakpoints

Then let's set a few breakpoints inside the DLL code to investigate the values being passed.

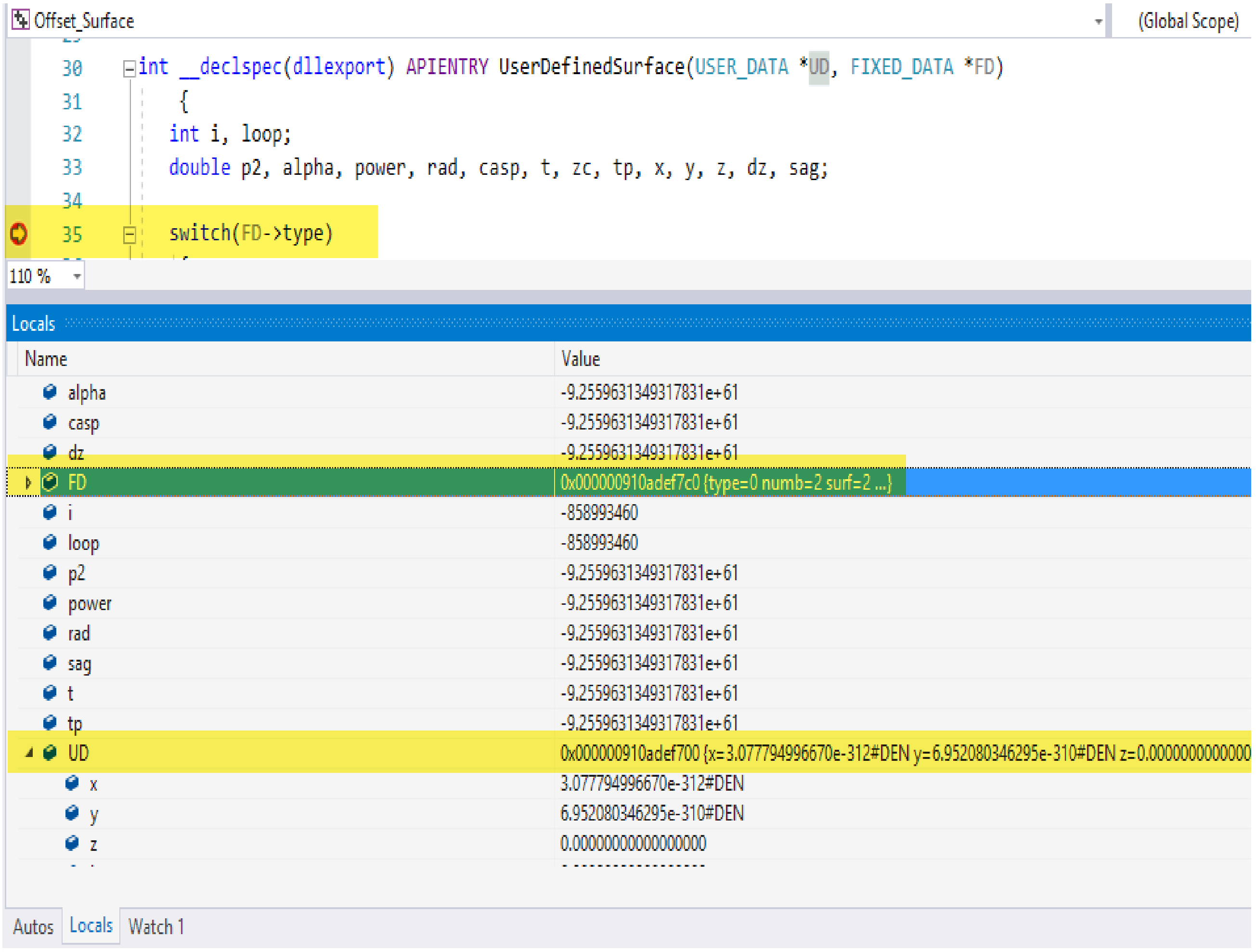

Offset_Surface (Global Scop

```
int __declspec(dllexport) APIENTRY UserDefinedSurface(USER_DATA *UD, FIXED_DATA *FD)
 {
int i, loop;
double p2, alpha, power, rad, casp, t, zc, tp, x, y, z, dz, sag;

switch(FD->type)
 {
  case 0:
   /* ZEMAX is requesting general information about the surface */
    switch(FD->numb)
     {
     case 0:
         /* ZEMAX wants to know the name of the surface */
```

Run the DLL/EXE

In OpticStudio, clicking on the User Defined Surface or updating the Merit Function with the UDOC operand will run the DLL or EXE and start the debugger. Back to Visual Studio, a yellow arrow is displayed on the breakpoint. It shows the code that will be executed next.

The window DEBUG > WINDOWS > LOCALS displays the values being passed between the DLL and OpticStudio. In that window, two types of data structures are passed: User Data (UD) and Fixed Data (FD).


Offset_Surface (Global Scope)

```
int __declspec(dllexport) APIENTRY UserDefinedSurface(USER_DATA *UD, FIXED_DATA *FD)
 {
int i, loop;
double p2, alpha, power, rad, casp, t, zc, tp, x, y, z, dz, sag;

switch(FD->type)
```

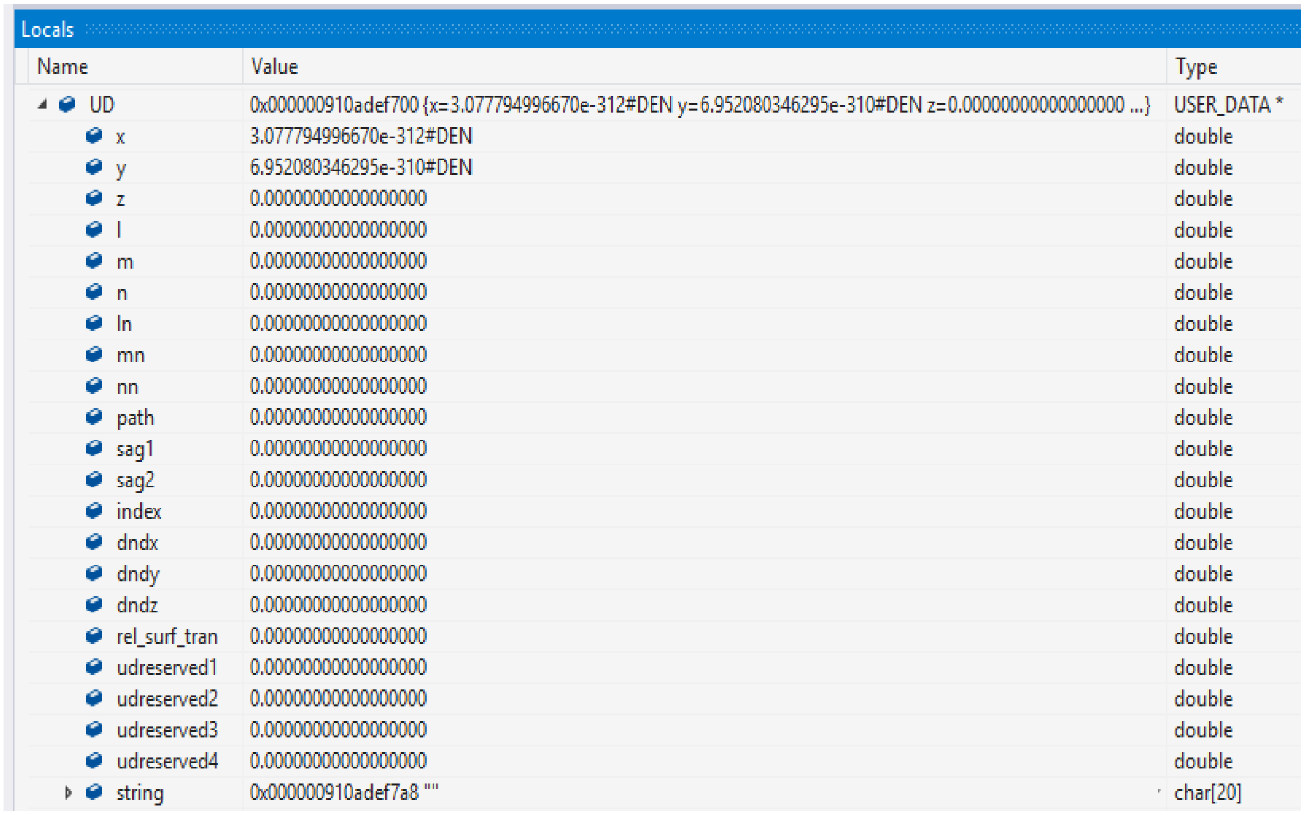
110 %

Locals

Name	Value
alpha	-9.2559631349317831e+61
casp	-9.2559631349317831e+61
dz	-9.2559631349317831e+61
FD	0x000000910adef7c0 {type=0 numb=2 surf=2 ...}
i	-858993460
loop	-858993460
p2	-9.2559631349317831e+61
power	-9.2559631349317831e+61
rad	-9.2559631349317831e+61
sag	-9.2559631349317831e+61
t	-9.2559631349317831e+61
tp	-9.2559631349317831e+61
UD	0x000000910adef700 {x=3.077794996670e-312#DEN y=6.952080346295e-310#DEN z=0.0000000000000
x	3.077794996670e-312#DEN
y	6.952080346295e-310#DEN
z	0.0000000000000000

User Data UD

It contains the rays x, y, z locations on a plane tangent to the surface vertex (the DLL will compute and return them for the real surface), the ray direction cosines l, m, n and the surface normals ln, mn, nn (which are computed and returned) and other data like index, transmission, path length.

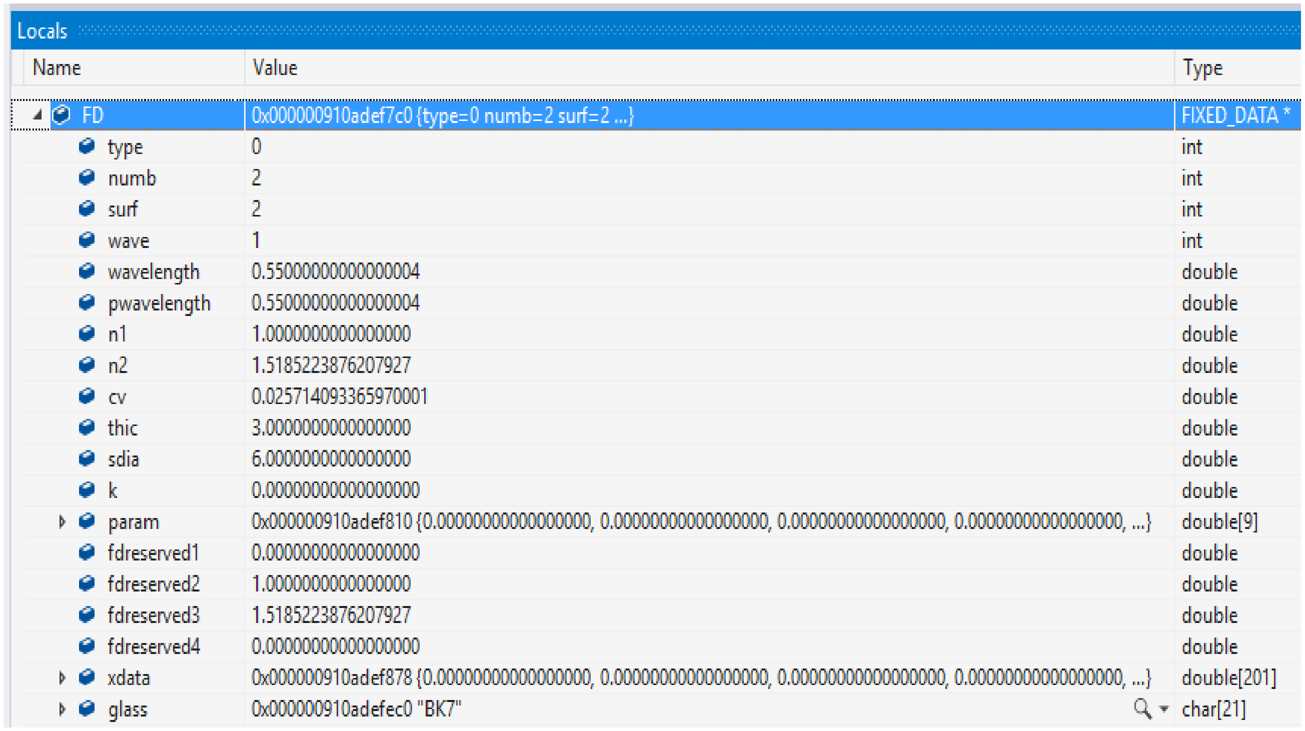


Locals

Name	Value	Type
UD	0x000000910adef700 {x=3.077794996670e-312#DEN y=6.952080346295e-310#DEN z=0.00000000000000000 ...}	USER_DATA *
x	3.077794996670e-312#DEN	double
y	6.952080346295e-310#DEN	double
z	0.00000000000000000	double
l	0.00000000000000000	double
m	0.00000000000000000	double
n	0.00000000000000000	double
ln	0.00000000000000000	double
mn	0.00000000000000000	double
nn	0.00000000000000000	double
path	0.00000000000000000	double
sag1	0.00000000000000000	double
sag2	0.00000000000000000	double
index	0.00000000000000000	double
dndx	0.00000000000000000	double
dndy	0.00000000000000000	double
dndz	0.00000000000000000	double
rel_surf_tran	0.00000000000000000	double
udreserved1	0.00000000000000000	double
udreserved2	0.00000000000000000	double
udreserved3	0.00000000000000000	double
udreserved4	0.00000000000000000	double
string	0x000000910adef7a8 ""	char[20]

Fixed Data FD

It mainly contains data the DLL cannot change: the surface data in the Lens Data Editor (radius, conic, parameters, etc.), the ray wavelength, polarization state and Type and Numb parameters indicating what data OpticStudio wants from a given call to the DLL.

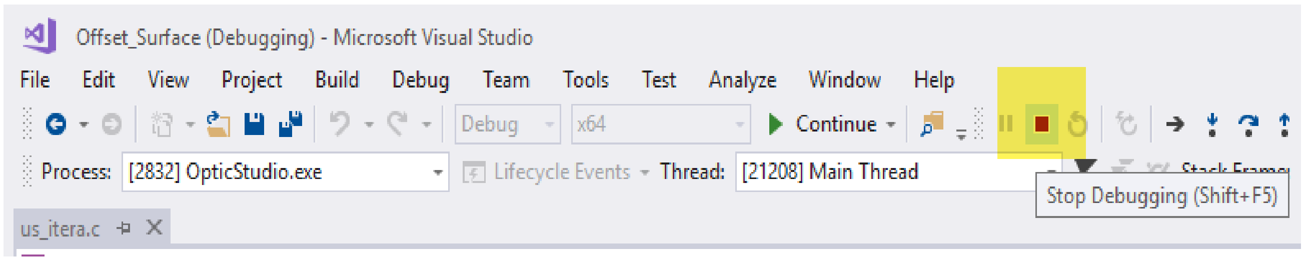

Locals

Name	Value	Type
FD	0x000000910adef7c0 {type=0 numb=2 surf=2 ...}	FIXED_DATA *
type	0	int
numb	2	int
surf	2	int
wave	1	int
wavelength	0.55000000000000004	double
pwavelength	0.55000000000000004	double
n1	1.0000000000000000	double
n2	1.5185223876207927	double
cv	0.025714093365970001	double
thic	3.0000000000000000	double
sdia	6.0000000000000000	double
k	0.00000000000000000	double
param	0x000000910adef810 {0.00000000000000000, 0.00000000000000000, 0.00000000000000000, 0.00000000000000000, ...}	double[9]
fdreserved1	0.00000000000000000	double
fdreserved2	1.0000000000000000	double
fdreserved3	1.5185223876207927	double
fdreserved4	0.00000000000000000	double
xdata	0x000000910adef878 {0.00000000000000000, 0.00000000000000000, 0.00000000000000000, 0.00000000000000000, ...}	double[201]
glass	0x000000910adefec0 "BK7"	char[21]

For example, FD -> n2 shows the index of refraction of N_BK7 being passed from OpticStudio to the DLL.

Stop the debugger

To stop the debugger, click on the red Square in Visual Studio.


Offset_Surface (Debugging) - Microsoft Visual Studio
File
Edit
View
Project
Build
Debug
Team
Tools
Test
Analyze
Window
Help
Debug
x64
Continue
Process: [2832] OpticStudio.exe
Lifecycle Events
Thread: [21208] Main Thread
Stack Frame
Stop Debugging (Shift+F5)
us_itera.c

Distributing the DLL/EXE in Release Mode

After debugging the DLL/EXE, the configuration can be changed back from Debug to Release. The libraries for Debug configurations are not redistributable while the libraries for Release configurations are redistributable.

Two techniques to debug a User Defined feature were demonstrated. The 1st one showed how to write a debug text file. The 2nd one showed how to use Visual Studio Debugger in the case of a DLL (User Defined Surface) and an EXE (User Defined Operand).
Both methods can be applied for any User Defined capabilities that OpticStudio has: DLL, User Defined Operands, ZOS-API User Extensions and ZOS-API User-Analysis.

References

- Attach to Running Processes with the Visual Studio Debugger: https://msdn.microsoft.com/en-us/library/3s68z0b3.aspx (https://msdn.microsoft.com/en-us/library/3s68z0b3.aspx)
- Navigating through Code with the Debugger: https://msdn.microsoft.com/en-us/library/y740d9d3.aspx (https://msdn.microsoft.com/en-us/library/y740d9d3.aspx)

Keywords: ZOS-API, Programming, EXE, DLL, User Defined