The simplest way to debug Fluent UDF

Debugging Fluent UDF is always the pain point for users. Intermediate variables can't be inspected and the source can't be traced step by step. User's progress remains stagnant especially because of inexplicable errors. Here, we recommend a UDF debugging tool that can directly inspect intermediate variables and trace the UDF source step by step.

- 1. Install Visual Studio. C++ and C# are recommended to be installed together. For 64 bit,
 - X64 compiler should be installed.

🕼 Microsoft Visual Studio 2008 Setup - Options Page					- • 💌			
Viscosoft Visual Studio Team System 2008 Maintenance Mode								
Select features to install:	Feature descr	iption:						
Koround't Visual Studio Team, System 2008 Good Language Tools Goo	Microsoft Visual Studio Team System 2008 Team Suite Microsoft Visual Studio Team System 2008 Team Suite is an expansion of the Visual Studio product line to include new lifecycle tools for project managers, architects, developers, testers, and operations managers to manage the software development process. Eeature instal path: C:\Program Files Microsoft Maual Studio 9.0\ Brogse							
Contemprise Tools	Volume	Disk Size	Available	Required	Remaining			
X Uortuscator Community Eattion X Toals for Redistributing Applications X Team Developer and Tester Tools X Team Database Edition X Microsoft SQL Server 2005 Express Edition X Crystal Reports Basic for Visual Studio 200	C:	40.0 GB	24.5 GB	1.0 GB	23.4 GB			
Restore <u>D</u> efaults			< Previous	Update	Cancel			

2. Download the latest tool version from website:

https://vcudfstudio.bitbucket.io/download_en.html



3. Install the tool using administrator right.



4. Start VC++ UDF Studio launcher and select VC/FLUENT versions to be run, then pressure OK button.



5. Read a case, then click the "Start Visual Studio" menu.



6. Edit source code and press "compile" button until no errors reported.

00	libudf - Microsoft Visual Studio		- • ×				
Ei	e <u>E</u> dit <u>V</u> iew VAssist <u>X</u> <u>P</u> roject	<u>a</u> uild <u>D</u> ebug Tea <u>m</u> D <u>a</u> ta V <u>C</u> ++UdfStudio <u>T</u> ools Ar <u>c</u> hitecture Te <u>s</u> t A <u>n</u> alyze <u>W</u> indow <u>H</u> el	p				
1	57 • 🖼 • 😂 🖬 🕼 🕺 🖎 🖄 👘) - 🔍 - 💭 - 🖏 🕨 Debug - 🔹 x64 - 🔹 🖄	• 🖏 🕾 🍹				
10	3 % 💺 🖉 🌾 🐺 📜 😫 (ଅନ୍ୟନ୍ୟ&଼ର <mark>,</mark> (⊞)୪ ଛିଛି ାା ା <u>ଜ</u> ିଆ :					
-	🙀 Solution Explorer 🚽 🕂 udf_source.cpp 🗙						
Sen	🕾 😰 🖻 🖧	♦ udf_source.cpp • ⊕ ♦ G:\Carrier-proj\testUdfMinGW\source\udf_source.cpp	- CGo				
er E	Solution 'libudf' (1 project)	(Global Scope) +					
xplo	VcUdfStudio2d_serial	Message0("Test for Calling UDF function!\n"); //call FLUENT U	DF function +				
P	Header Files	}	^ _				
	🗀 Resource Files						
읕	 Source Files 	DEFINE_EXECUTE_ON_UNLOADING(unload,libudf) //execute w	hen unload				
8	udf_source.cpp	{					
		Message0("Now unloading the udf library!\n"); //call FLUENT	UDF function				
		}					
			E				
		// TODO: put DEFINE_XXX_ODF macros nere					
		SDEFINE_ON_DEMAND(debug)					
		int aaa=123:					
		int bbb =345:					
		int ccc=aaa+bbb;					
		}					
			v				
	💐 S 🏹 T 🥞 Cl 🖓 R	100 % • 4 10	Þ				
	Output		- 4 ×				
	how output from: Build • 2 with a state of the state of t						
1) 1)%:13							
1/ould succeeded.							
	1>Time Elapsed 00:00:16.24 ====================================	failed, 0 up-to-date, 0 skipped ===========	*				
	4		Þ				
Bu	ild succeeded	Ln 16 Col 1 Ch 1	INS				

7. Set a breakpoint (F9) and press "debug" button to enter debug mode.



8. Run corresponding subroutine or macro. The program will stop at the breakpoint.



9. Run the code step-by-step (F10), then all intermediate variables can be inspected.



10. The unregistered tool can use 2d/3d serial version with maximum 3 UDF macros, which are enough for preliminary UDF programming. For more macros or parallel version, registration needed.