Debugging UDM

How to use a symbol server with the Visual Studio .NET debugger

It can be done by setting the **_NT_SYMBOL_PATH** global environment variable:

1. In Control Panel, double-click **System**
2. On the **Advanced** tab, click **Environment Variables**
3. Under System Variables, click New, and then add a variable as **_NT_SYMBOL_PATH**
4. Set the value of the variable to the UDM symbol server path:
   symsrv*symsrv.dll*c:\localcache* http://symbols.isis.vanderbilt.edu/symbols/

Note: the c:\localcache folder is used to cache the downloaded symbols.

Sharing the symbol files on a network for group development

Set the **_NT_SYMBOL_PATH** variable in each developer workstation to the following value:

symsrv*symsrv.dll*\\localserver\share*http://symbols.isis.vanderbilt.edu/symbols/

where \\localserver\share is a readable and writeable local network share.

Note: SymSrv always looks for the symbol file in the leftmost symbol store. If the right symbol file found in \\localserver\share it is used. If it is not there, SymSrv looks in the symbol store immediately to the right (the UDM symbol server in our case). If the file is there, it is copied to the left store and opened from there.

Get the UDM source code

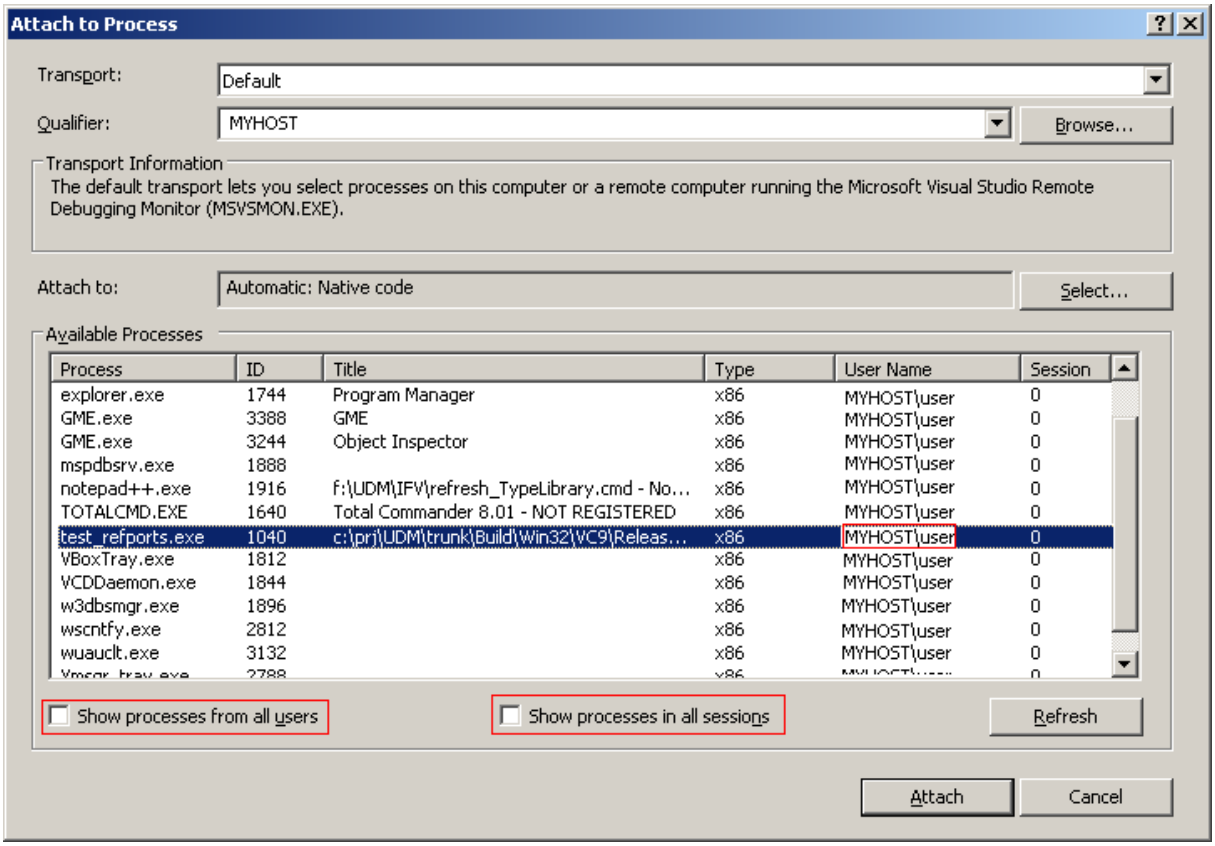
The UDM source code is available at http://repo.isis.vanderbilt.edu/downloads?tool=UDM

Find the source code that corresponds to your UDM version, download it and unpack it to a local folder.

While debugging, step through your code until you step into the UDM code. At this point a dialog will pop up asking where the source files are and you can attach the matching files to debug process.

Attach to a running process for a command-line UDM interpreter

1. On the **Debug** menu, select **Attach to Process**.
2. In the **Attach to Process** dialog box, find your process in the **Available Processes list** (select the **Show processes from all users** check box if the process is running under a different user account or select the **Show processes in all sessions** check box if you are connected through Remote Desktop Connection)
3. Click **Attach**.



Note: if there is no time to attach to the process you could place a cin.ignore(); or cin.get(); in your code to wait for a user input.

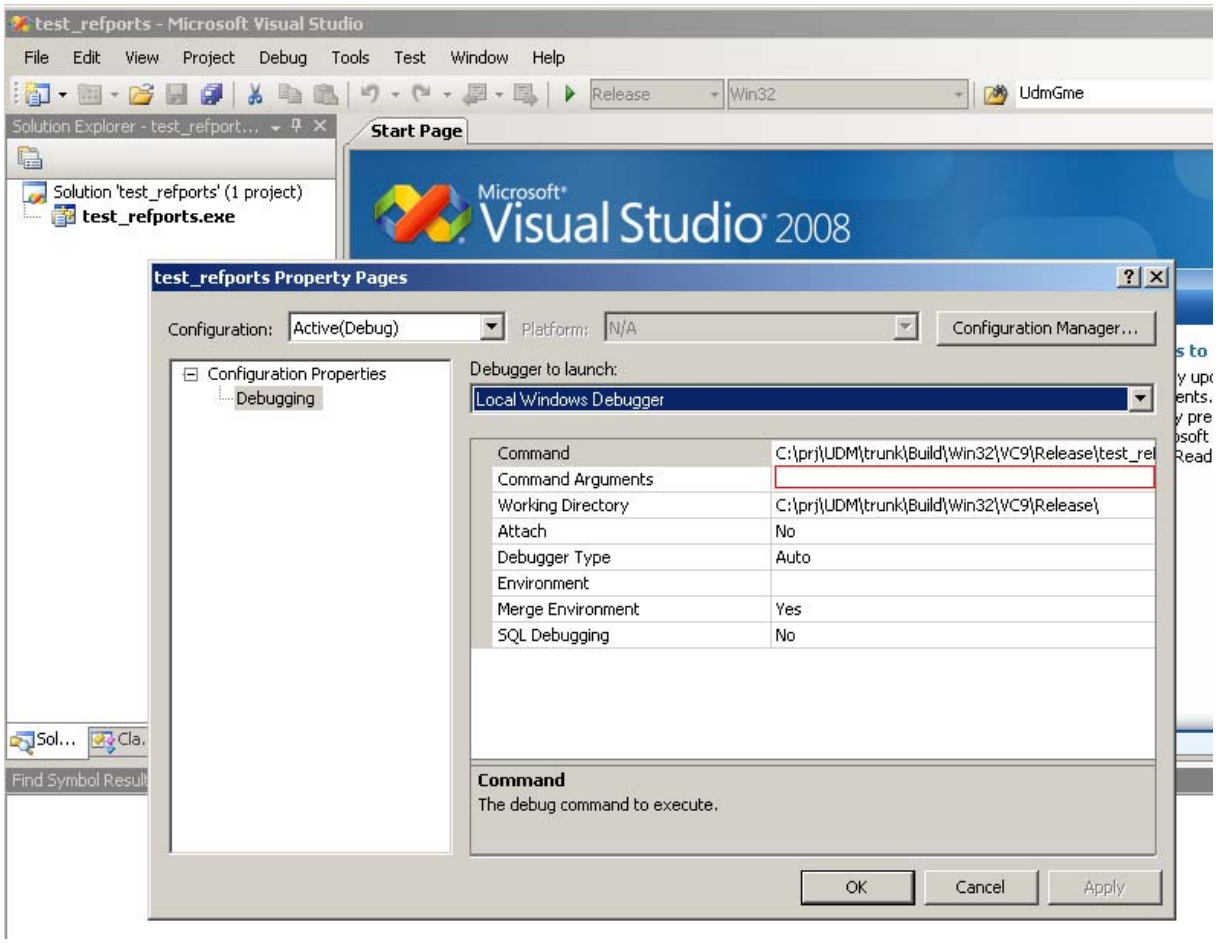
Start a debug session for a command-line UDM interpreter

In most cases there is no time to select the process because it terminates the execution very fast.

In this case we can use an other method to debug a command-line UDM interpreter:

1. Open **Visual Studio**
2. Choose **File** menu **Open**->**Project/Solution**
3. Locate and select your executable program you want to debug and open it.
4. Right click on [YourProgram].exe „project" and choose **Properties**.

5. In **Configuration Properties**->**Debugging**, **Command Arguments** can be used to specify the command line arguments.



6. Choose **Debug** menu, **New breakpoint** -> **Break at function** and type a function name.
7. Click **F5** to start debugging the program.
8. The program execution will break at the specified function and you can step through the UDM code.

Note: if we have a configured project that builds the interpreter we only have to specify the command line arguments and to create a breakpoint. After that we are ready to debug our interpreter.

Start a debug session for a live GME-based UDM interpreter

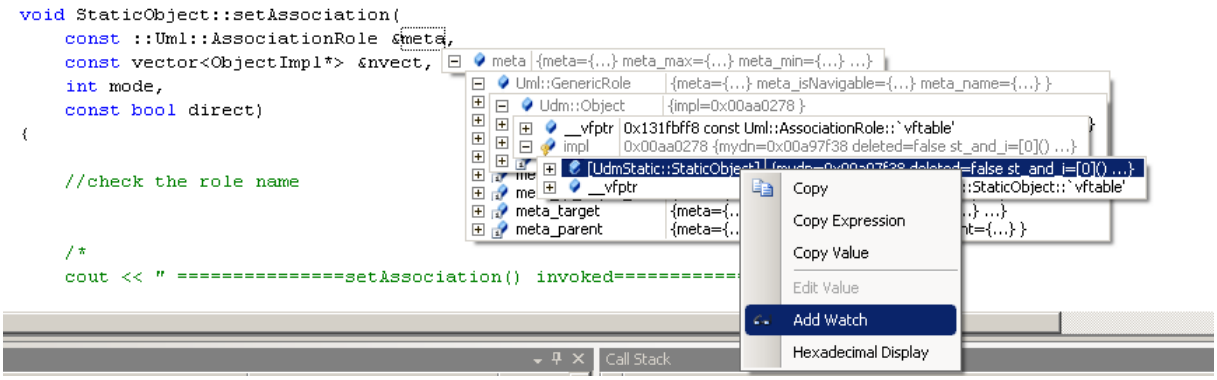
Debug your own project that depends on UDM, step through your code until you reach the UDM code. If the symbol server is set and the matching UDM code is available for use, you can debug the UDM interpreter.

Attach to a running process for a live GME-based UDM interpreter

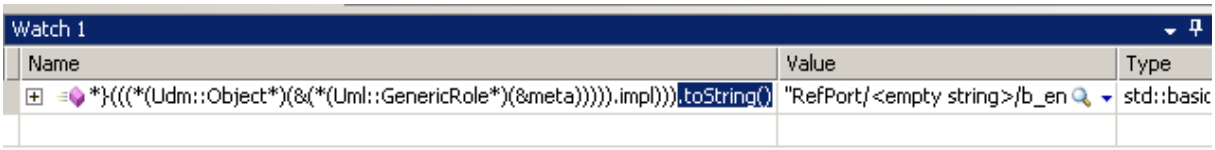
Please refer to „Attach to a running process for a command-line UDM interpreter" section.

Identifing particular UDM objects and get access to attribute data

Locate the object implementation, add to the watch window and call the toString() method.



The toString() method provides useful informations about the current object.



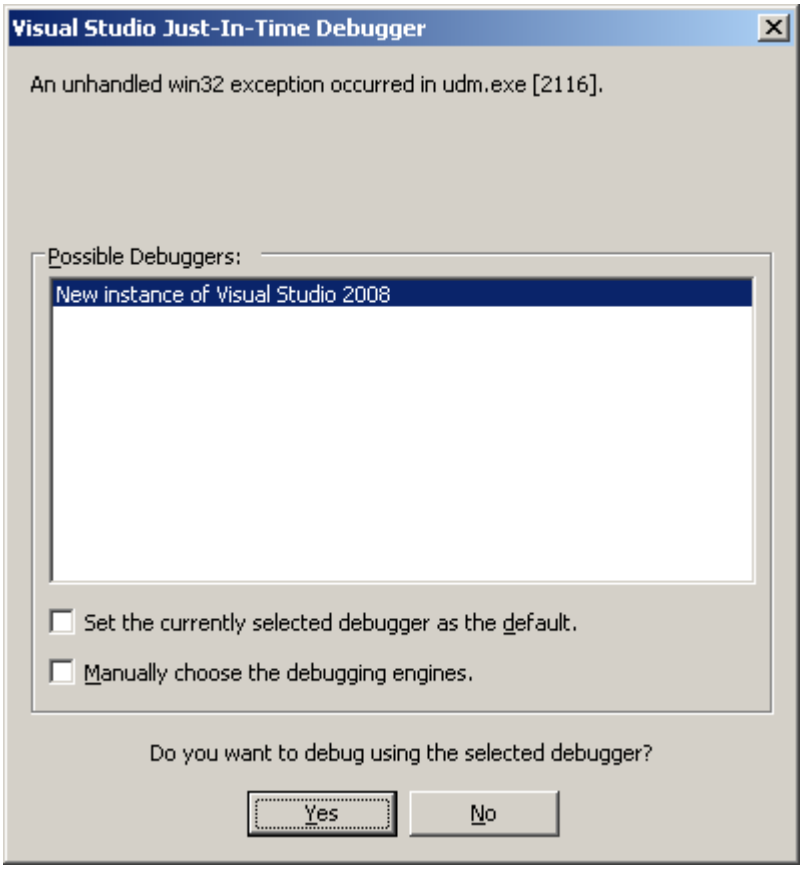
Package up stack traces and program data to submit a bug report for UDM

Just-In-Time debugging launches Visual Studio automatically when an exception or crash occurs in an application running outside Visual Studio. This enables you to test your application when Visual Studio is not running and begin debugging with Visual Studio when a problem occurs.

When a crash or exception occurs, you will see a dialog box appears with a message that looks something like this:

An unhandled exception occurred in yourprogram.exe[line number]

Choose the **New instace of Visual Studio** from the Possible Debuggers list and click Ok.



This will start a new debugging session in Visual Studio and you will see a new dialog box with the error description.

Click on the **Break** button to break the program's execution at the current point.

In **Debug** menu choose **Save Dump as** and save the minidump file.

Send the minidump file to analysis.