TYX Corporation

Productivity Enhancement Systems

Reference	TYX_0051_17
Revision	1.0
Document	dotNetDebug.doc
Date	June 21, 2005

How to debug Wcem.dll with .NET Studio

1. Studio Version:

This document is for Studio **1.17.0** and later. The pictures below have been made using version **1.31.1**.

The Subset used for the Atlas is **IEEE716.89/Paws**.

This document assumes that the user has some knowledge of **PAWS Developer's Studio** and **MSVS .NET**.

2. How to debug the Wcem.dll from the MSVS .NET compiler:

We can utilize Microsoft Visual .NET and debug our drivers.

The first step is to have an Atlas program and a device database.

We are going to walk through an example in order to make this easier to understand. Here is the source file for a small Atlas, device database and Busconfi example:

---------- Atlas1.atl ----------

```
 000000 BEGIN, ATLAS PROGRAM 'Simple Dynamic'      $

E900000 OUTPUT, C'START TEST'                      $

     10 SETUP, AC SIGNAL,

        VOLTAGE 5 V,

        CNX HI                                     $

 999999 TERMINATE, ATLAS PROGRAM 'Simple Dynamic'  $
```

---------- end of Atlas1.atl ----------

---------- Ddb1.ddb ----------

```
configuration Wcem_debugtest;
def, fnc, Dsp   ==      45;     ** Display
*****************************************************
begin dev FNG;
  cnx hi virtualpin;
  begin FNC = 10;
    control{
           voltage range 0 v to 140 v;
           }
    source ac signal;
  end;
end;
```

---------- end of Ddb1.atl ----------

---------- busconfi ----------

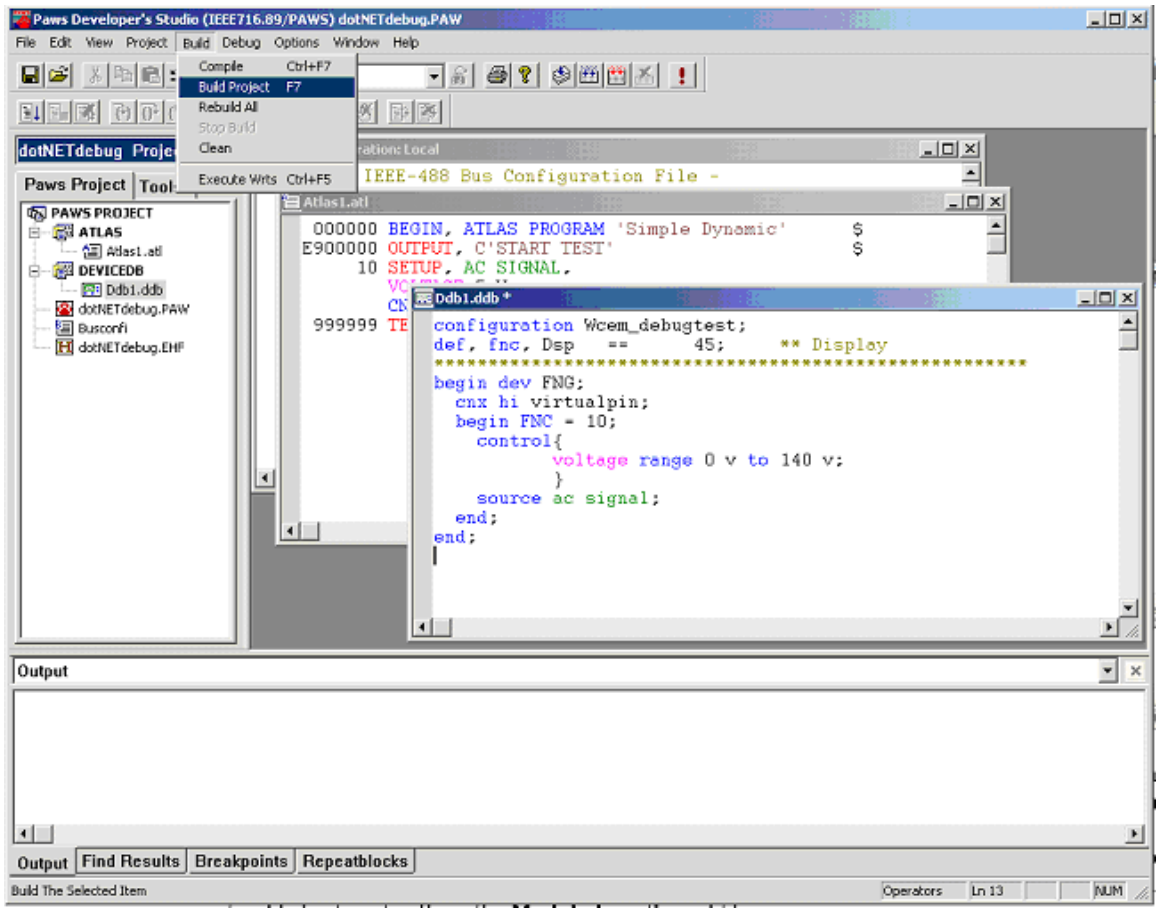
```
;       IEEE-488 Bus Configuration File -

"Channel"        2

FNG     BUS 2   MLA 11  MTA 11
```

---------- end of busconfi----------

Put those files in a Paws project called **dotNETdebug**. In our case, we will put the project under **C:\usr\paws\ dotNETdebug**.

You may place the **Busconfi** either in the local directory or you may place it in the station subdirectory in your Station.

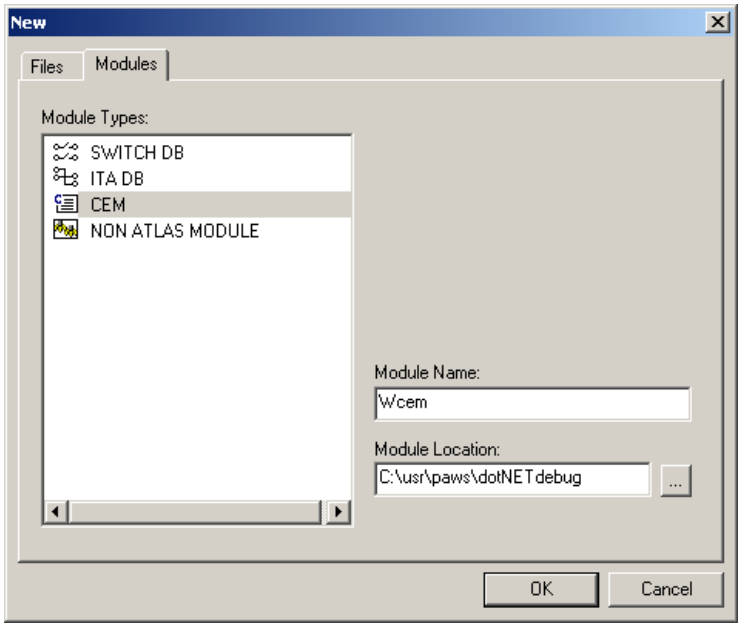
- Build the project, as shown below, under the TYX Studio.



- Now that the device database is built, you can create a **Wcem** module as shown below. You can call the module **Wcem** and redefine the path to locate the module in the .paw subfolder rather than in **.\Wcem**.

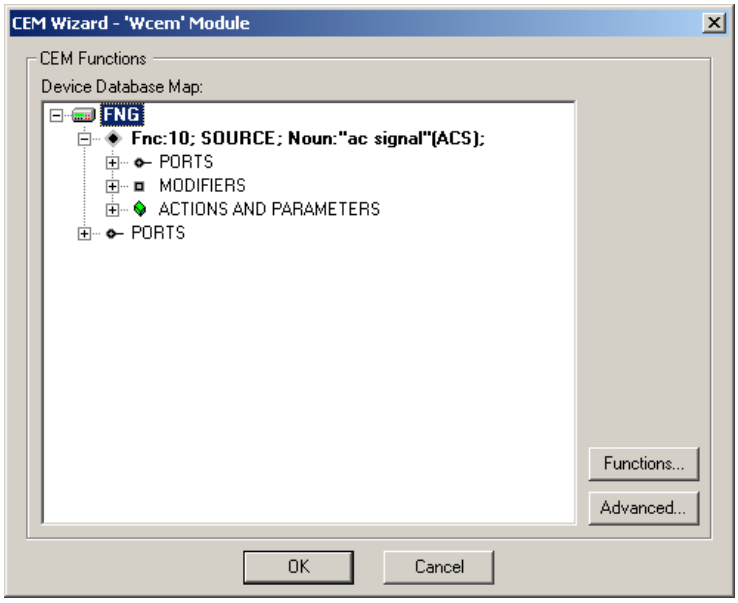
  Note: By default, when you add the name for the **Module Name**, the files will be put into Module name subfolder into the **.paw** folder, so you will need to delete the

module name folder extension that gets added automatically in the **Module Location** edit box.

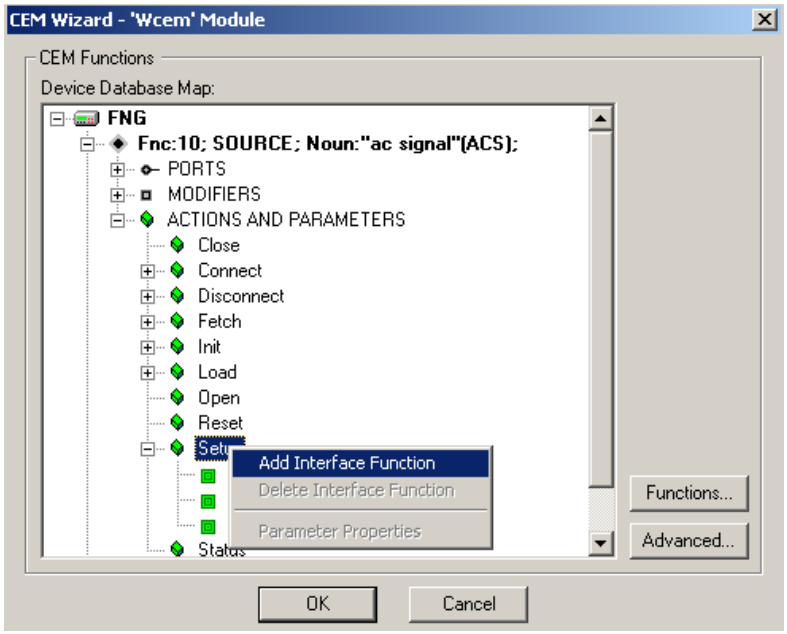


- You will then have access to the **Wcem Wizard** in order to create the source files that will allow us to build the **Wcem.dll**.

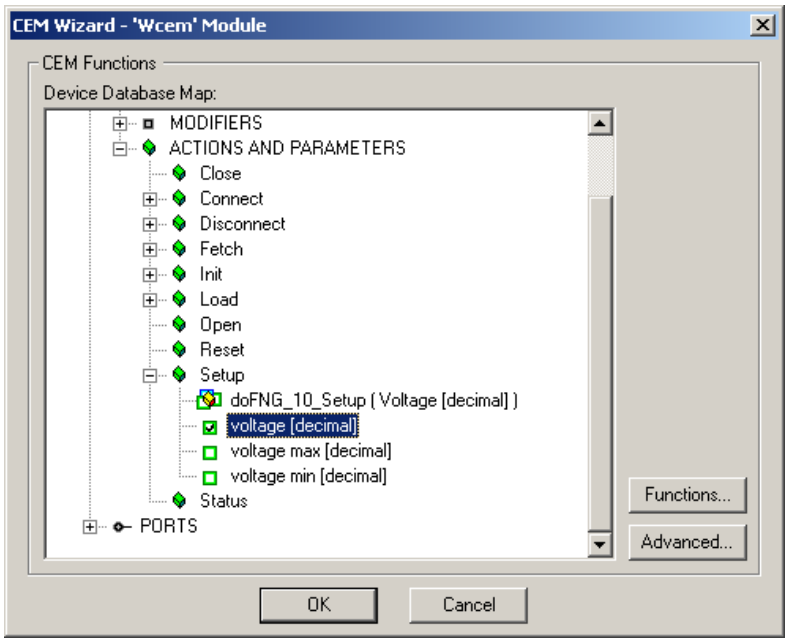
- Using the **Wcem Wizard…** under **View**, create the proper setup function associated to the Atlas **Setup** function. In the next steps you will see what actions to take.



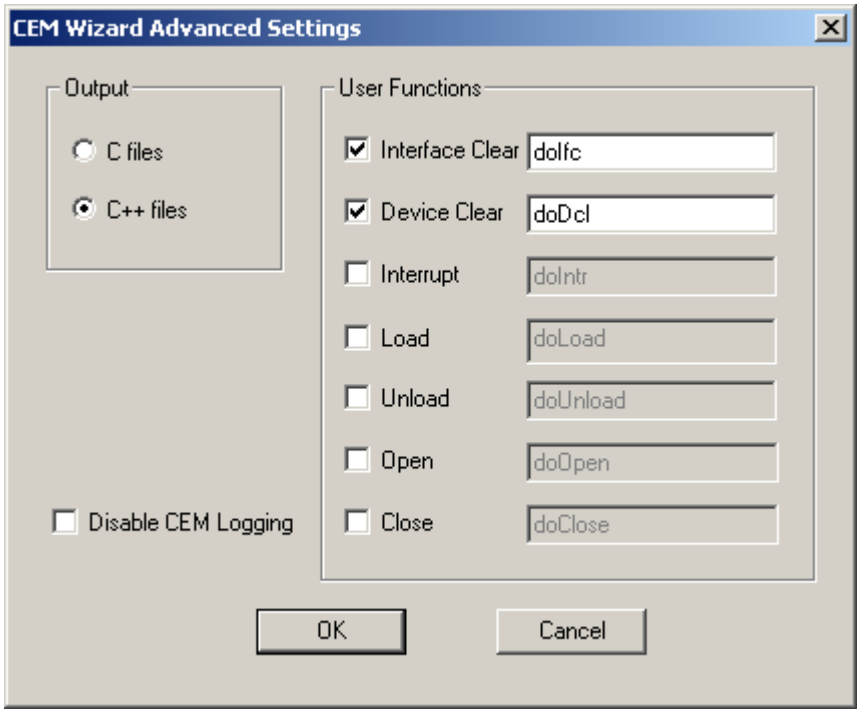
- Left-click on **Setup** under **ACTION AND PARAMETERS** and you should see pulled-down menu on which chose **Add Interface Function**:



- Then check the **voltage[decimal]** box.



- You may wish to add some features under **Advanced** such as shown below. This will be responsible for an additional **C++** file generated by the **Wcem Wizard** called **ctlr.c**.



- Once all the functions have been mapped properly, you should click on **OK**. The Wizard will generate a list of source files in the TYX Studio Project. In this case, the list of files is the following: **Wrapper.cpp**, **key.h**, **error.cpp**, **ctlr.cpp** and **FNG.cpp**.

- The file that will include the driver code that you might want to debug will be in this case **FNG.cpp**. We will add a Display function in the **doFNG_10_Setup** function in **FNG.cpp** as shown below:

```
#include "cem.h"

#include "key.h"


//BEGIN{DFW}:FNG:10:0

int doFNG_10_Setup (
```

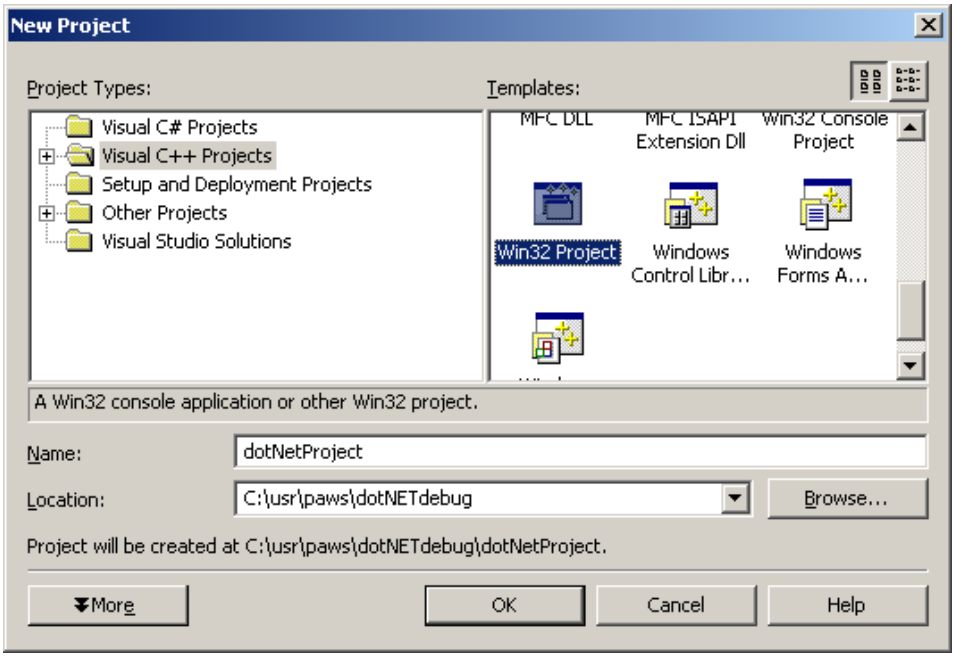
```
     // voltage [decimal]

     double VOLT)

     // Set the return to a non-negative integer to report
a success status.

     // Set the return to a negative integer to report an
error status.

//END{DFW}

{

     // CEM Logging Function

     userStubSETUP();



     // Please insert your CEM driver code here.

      Display("Entering doFNG_10_Setup\n");

     return 0;

}----------------------------------------------------------------------
```

Note: If you build the Wcem.dll from the Paws Studio, make sure that you either delete it or overwrite it with **Wcem.dll** generated by the MSVS .NET Studio, or it might cause some problems when debugging the **dll** from the MSVS .NET generated project.

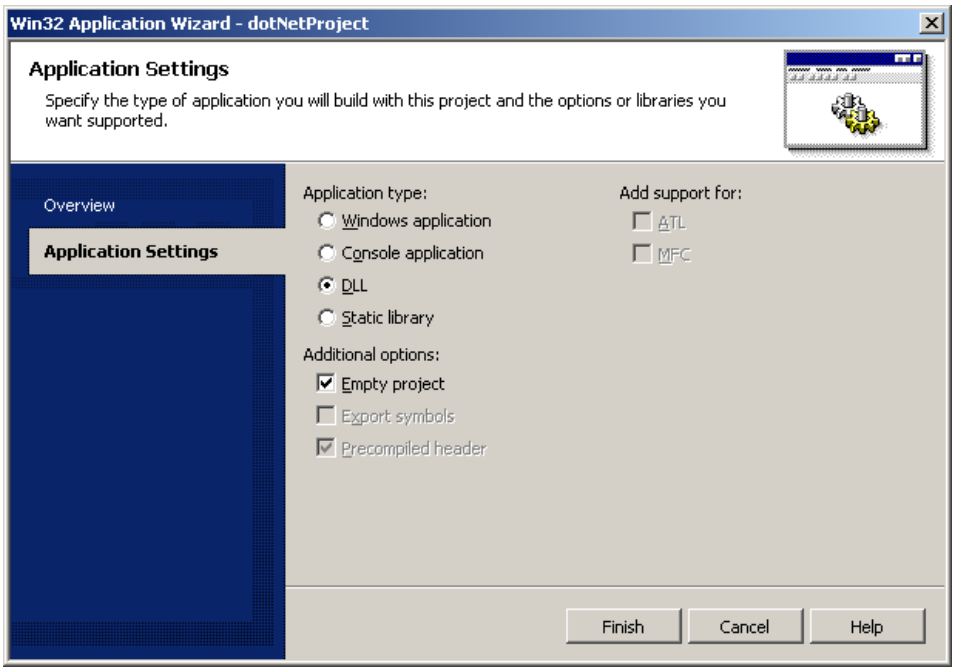
3. The MSVS .NET environment:

Now, we are ready to move on to the MS environment. This example uses the support of **MSVS .NET**.

- From the **MSVS .NET** click on **New Project** button and select **Win32 Project** from the **Templates** section. Fill in the **Name**, and the **Location** as shown below:

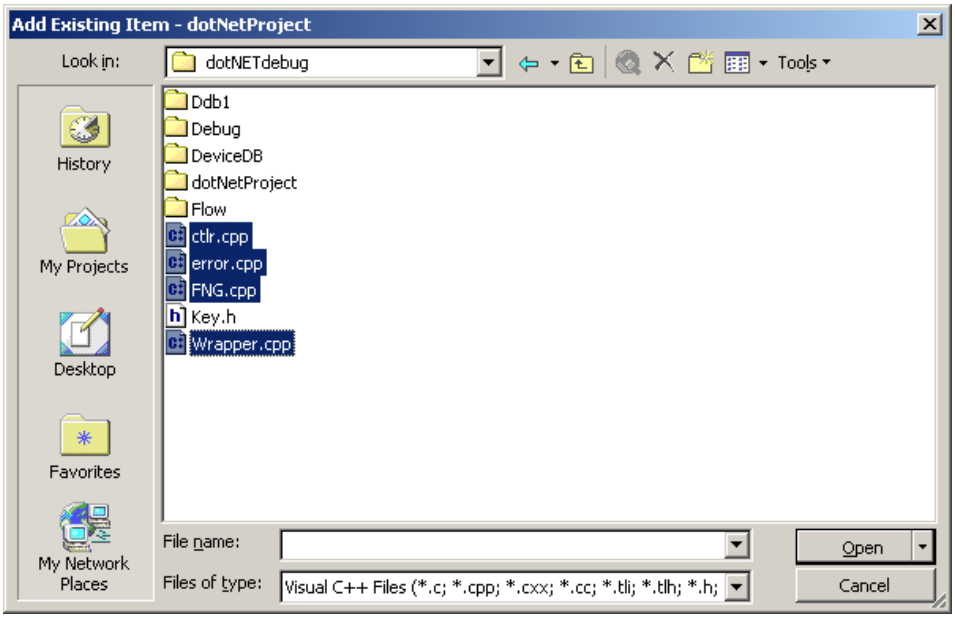


- After pressing **OK**, go to **Application Settings** and chose **DLL** under **Application type** and **Empty project** under **Additional options** as shown below:



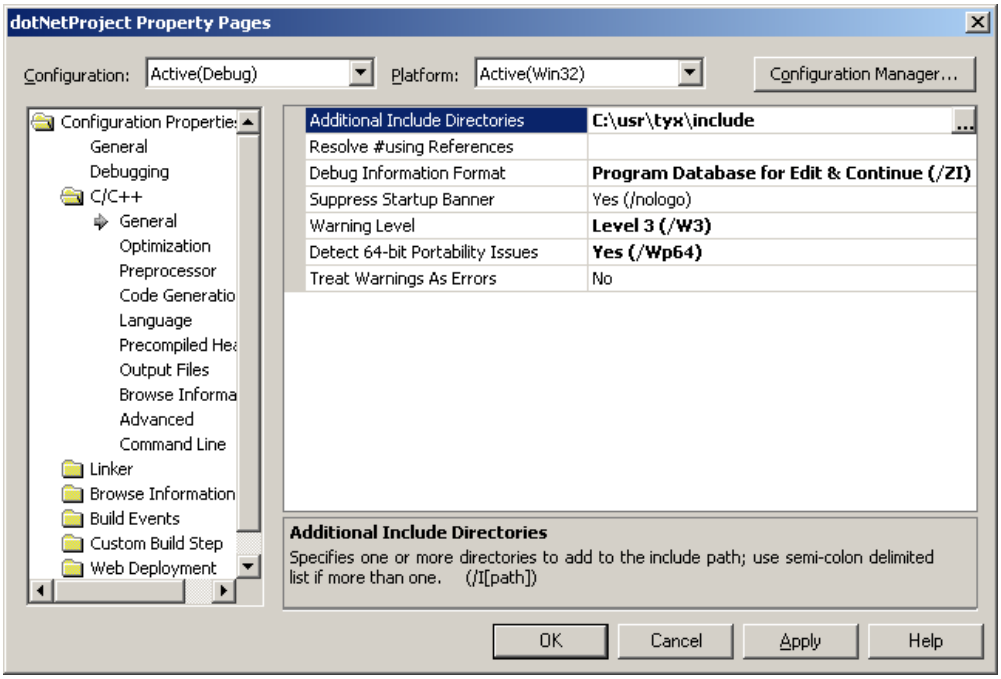
- Press **Finish**.

- Now, select **Project**, **Add Existing Item…**. This will open the window below. Select the TYX Cem C++ files one after another. Here, we will select **Wrapper.cpp**, **error.cpp**, **ctlr.cpp** and the **FNG.cpp** files.

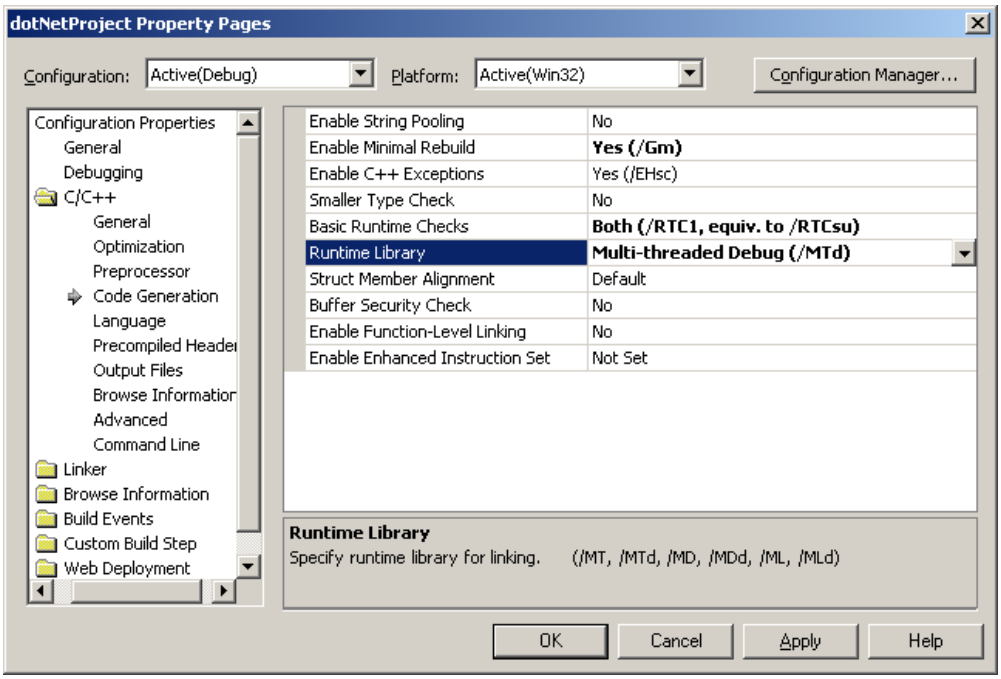


- Press **Open**.

- We now need to add basically all the settings that you would have in the TYX Wcem settings. The first thing will be to include the additional path of all the header files that you wish to include in your project. In this case, one that is unavoidable is the **cem.h** file from TYX. The path is usually **C:\usr\tyx\include**. This can be done going into **Project-> <name of the project> Properties…-> C/C++** in the **General** Category as shown below:

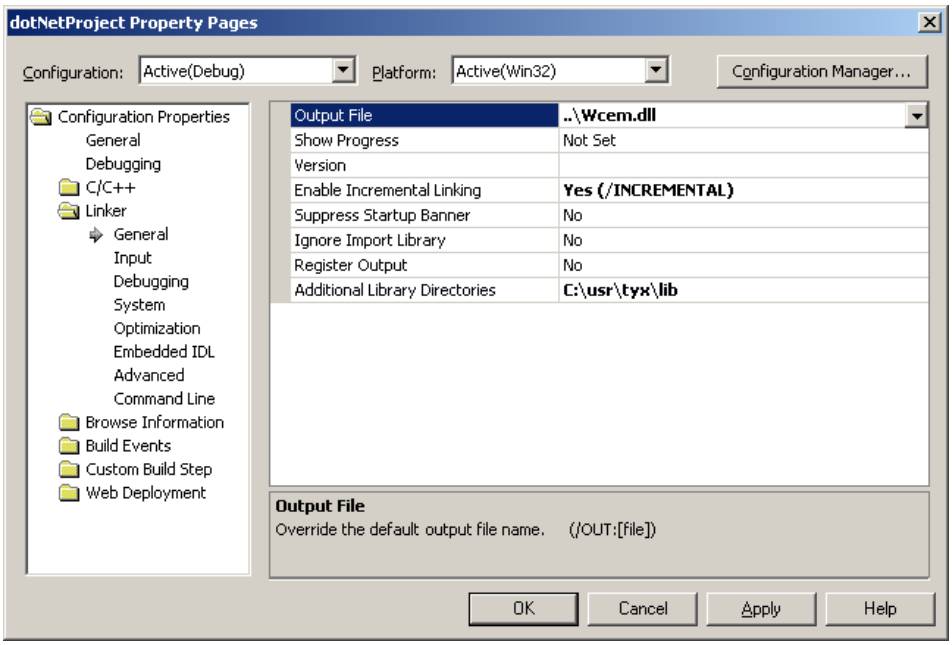


- **Note:** For greater safety, make sure that the **Runtime Library** option under **Code Generation** is set to **Multi-threaded Debug** as shown below:



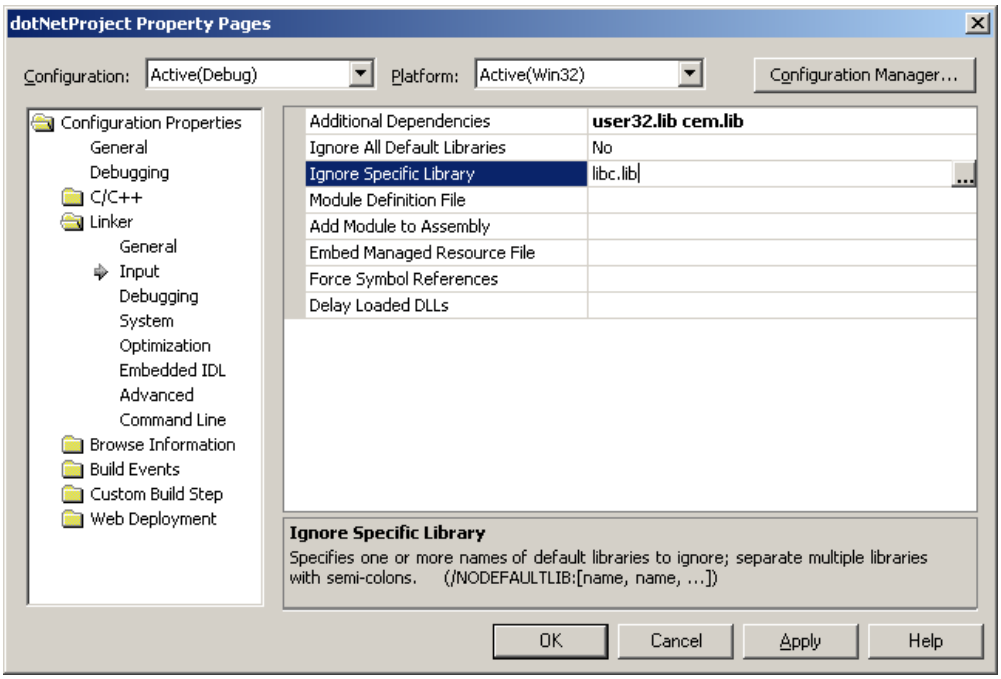
- Under the **General** Category in **Link**, You also need to specify the location of the **Wcem.dll** output file. This is where you need to be careful about making sure that it will not conflict with the one generated with the TYX studio. In this case, we will just locate the **Wcem.dll** generated here in the same directory as the one used by the TYX studio.

- We need to add the additional path for the **Additional Library Directories** as show below:

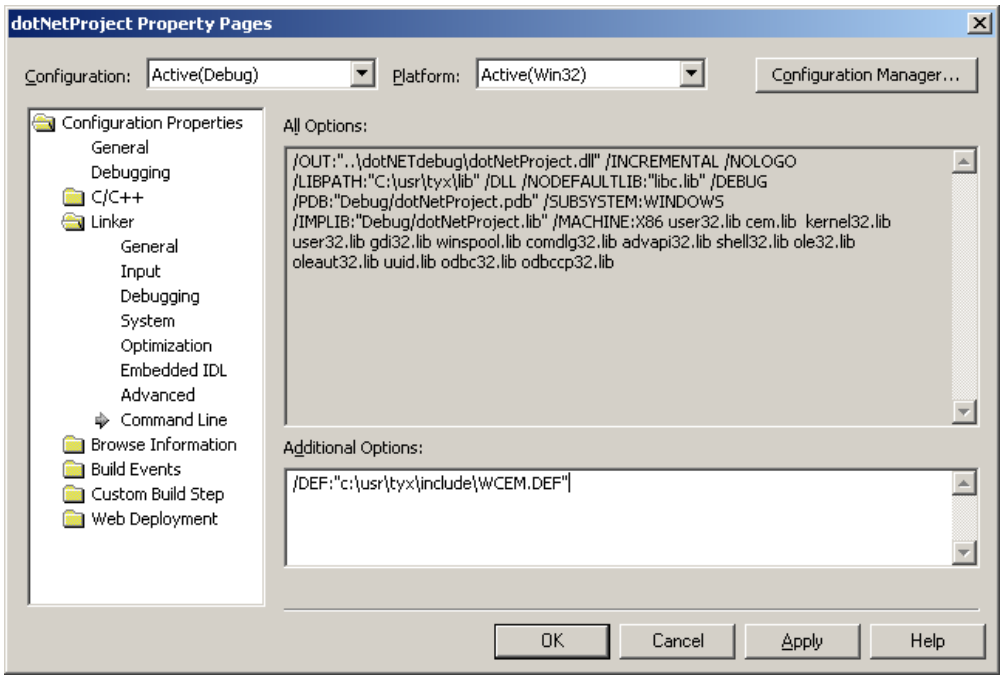


- Under the **Input** Category in **Linker**, we need to include all the libraries that will be addressed by our project. In this case, we need to add **cem.lib** and **user32.lib** as indicated below. You may wish to delete any additional libraries (if any appear) that aren't used, but it will not affect your project if you leave them there.

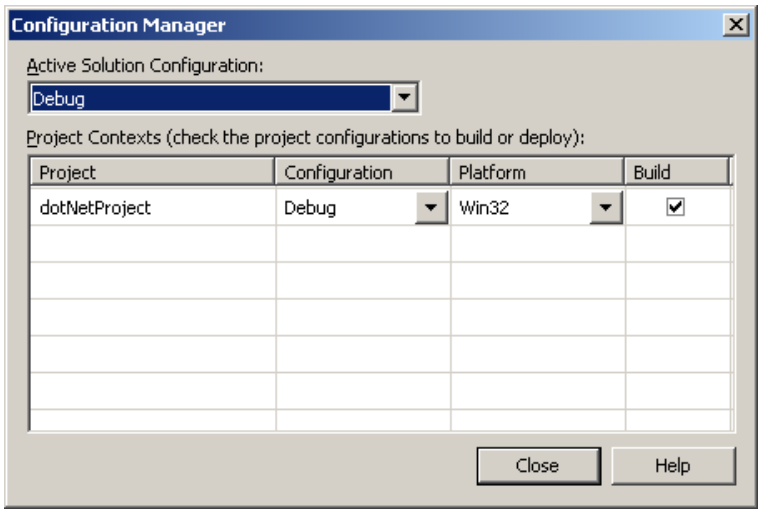
- We also need to ignore the **libc.lib** library to avoid redefinition warnings.



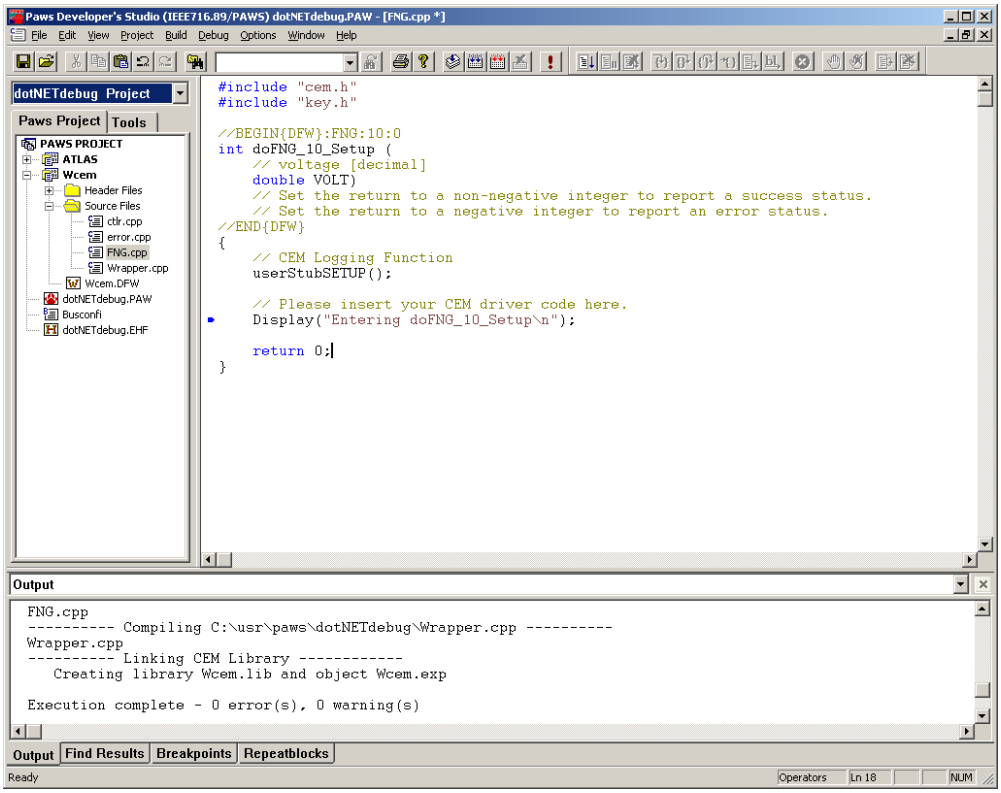
- Under the **Command Line** also in **Linker**, we need to add **/DEF:"C:\usr\tyx\include\WCEM.DEF"** as shown below. If we fail to do this, the **dll** will build, but the Wrts will fail to make proper use of the **Wcem.dll**.



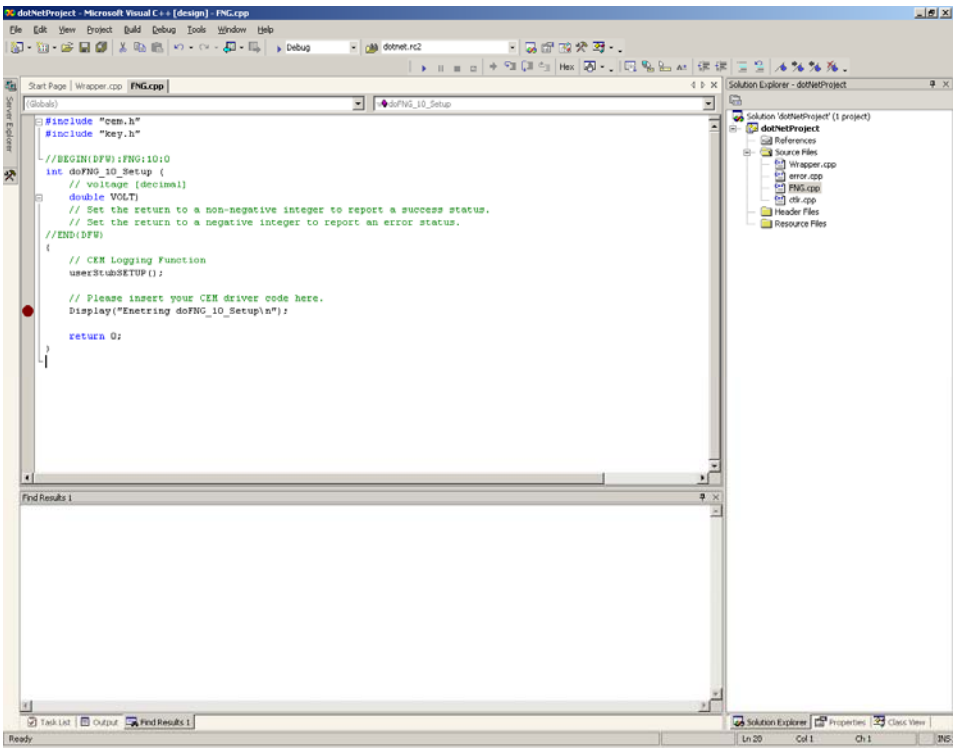
- You are now ready to build the project and start it in debug mode. Make sure that the **Active Solution Configuration** is the debug version.



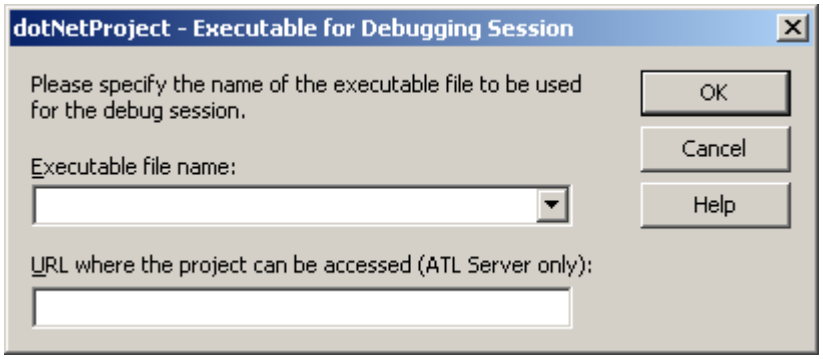
- Build the Paws project from PAWS Studio by pressing **F7**.



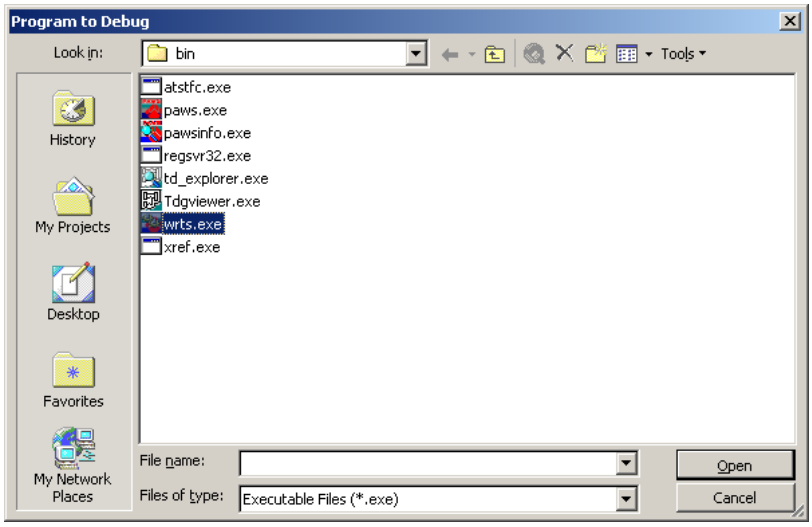
- You can place breakpoints in the **CPP** files that you have under the MS Studio, such as at the Display function in the **FNG.cpp** file.



- Now you can run the project from MS Studio in debug mode via **F5** or via **Debug->Start**. You will see the following window:

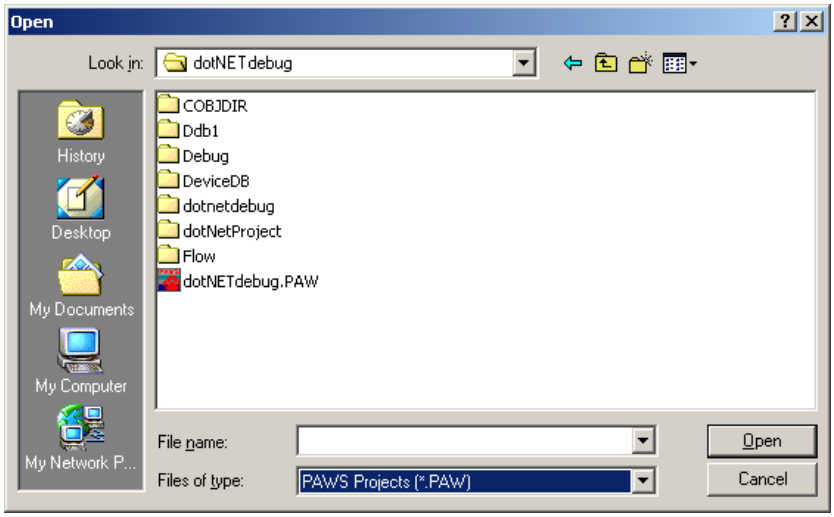


- That is the window that will determine which executable to launch that will load the **Wcem.dll**, which is the output of the MSVS .NET project. Click on the arrow next to the edit box and select **Browse**. Select **Wrts.exe**. This exe is usually located under **c:\usr\tyx\bin**

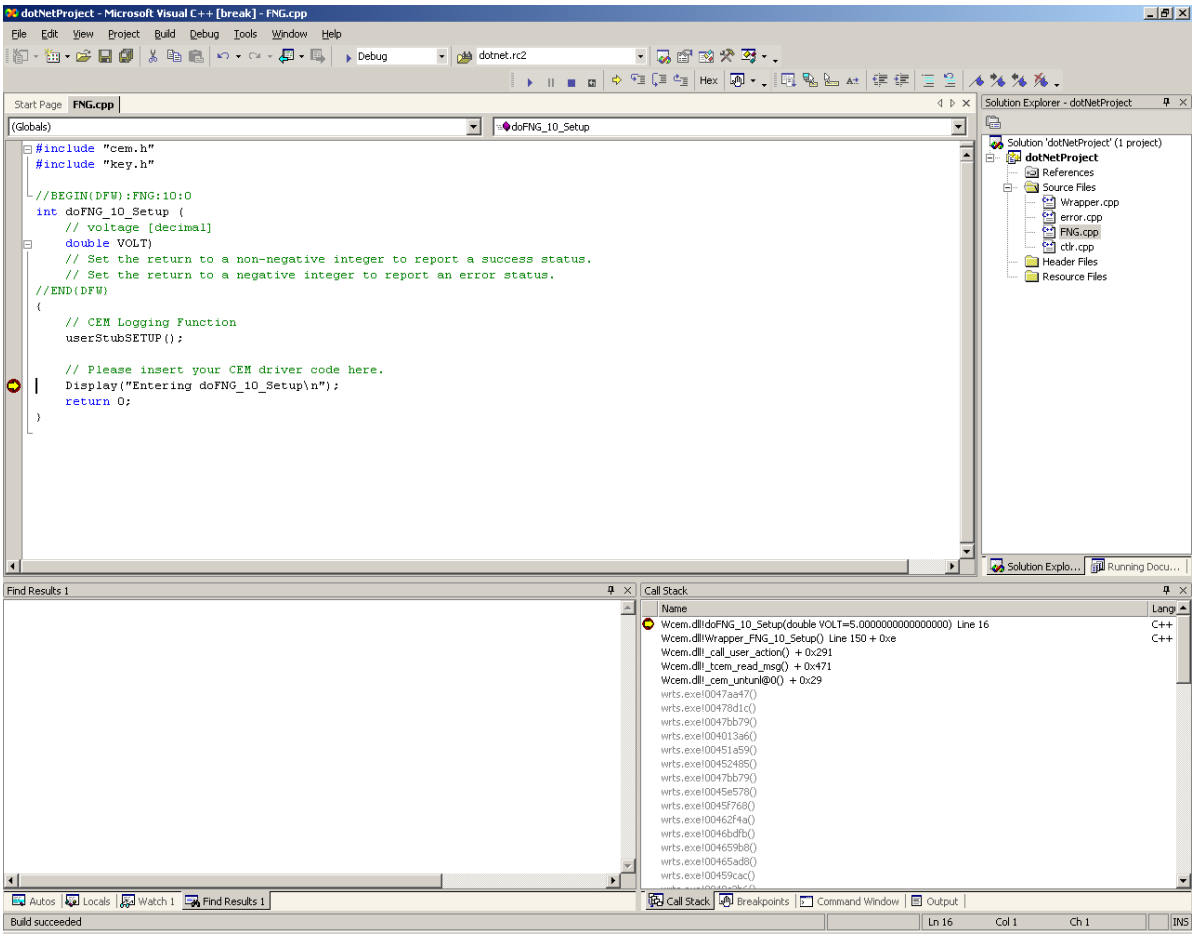


- Click on **Open,** and then **OK**.

- Once the Wrts started, you should select the **dotNETdebug.PAW** project as shown below:



- Run the project from the Wrts and MSVS .NET will stop at the breakpoint set in the **Setup** function as shown below. In this case, we placed a breakpoint in the **FNG.cpp** file.



- As we can see, the value specified in the **Atlas** for the **Voltage** value is visible.

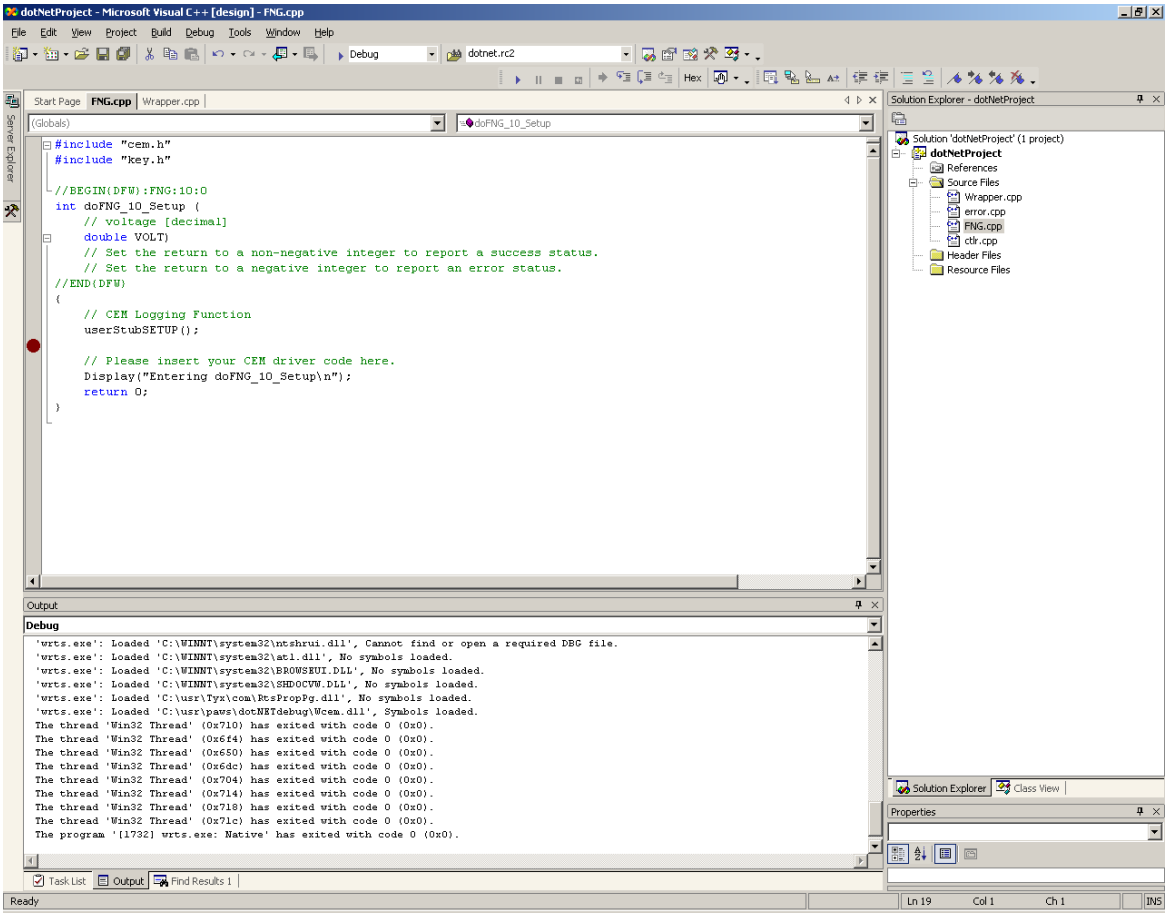
- **Note:** If you are having problem with the breakpoint, it will be because you will have built the Wcem.dll with the Paws Studio which has a release configuration and cannot be debugged. In order to overcome this problem, you need to rebuild the project from the MSVS .NET in order to overwritte the Wcem.dll generated by the Paws Studio.

4. How to debug the Atlas while you are debugging the Wcem.dll driver?

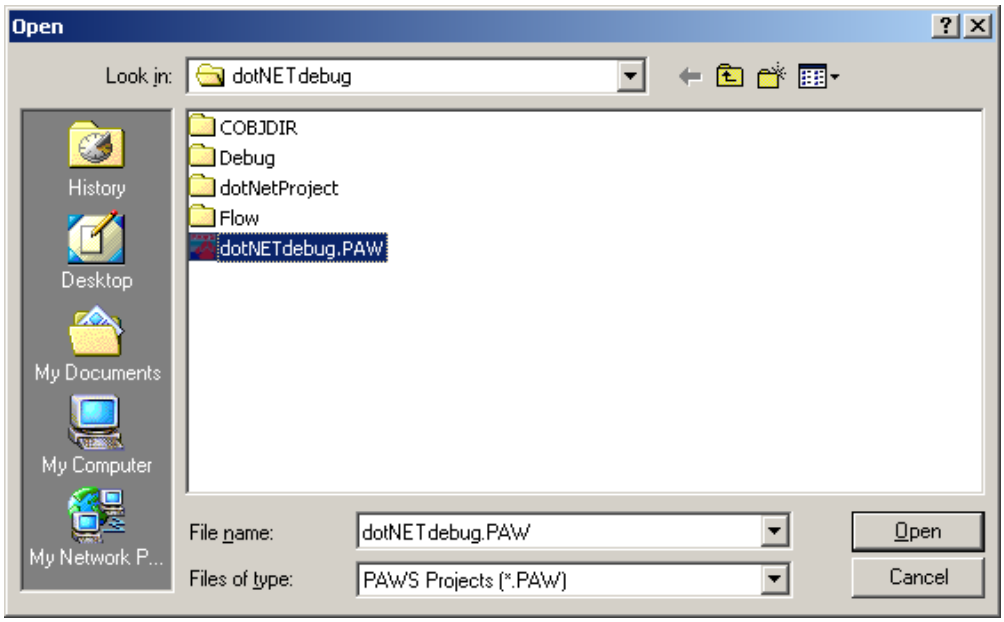
This is a simple procedure.

1. You need to build the **Wcem.dll** in debug mode.

2. You need to start the Wrts from the MSVS .NET with **F5**. This will allow to debug the **Wcem.dll**.

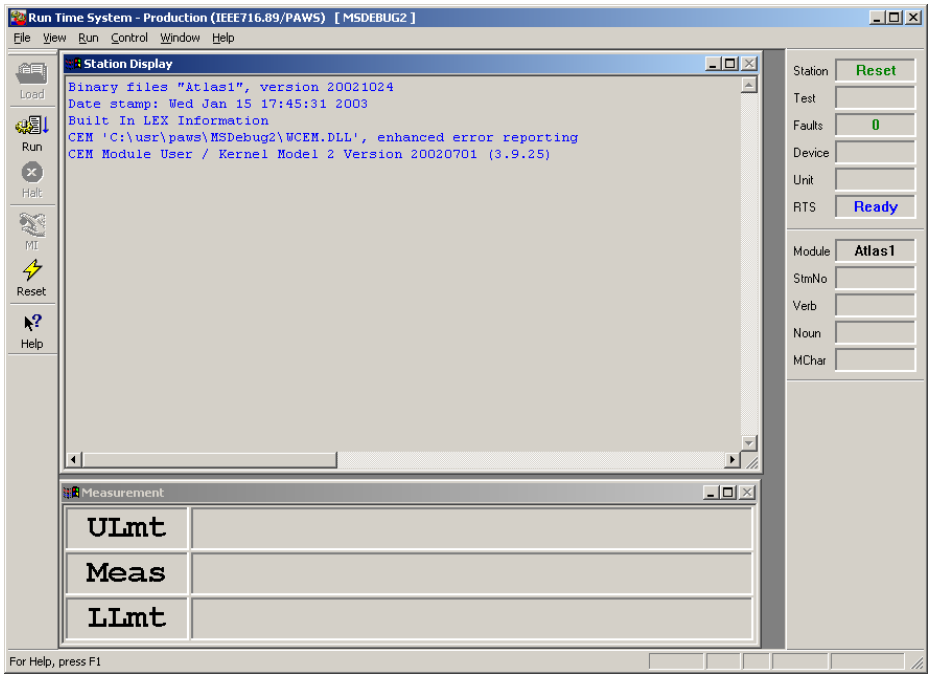


3. Load the project that you want to run in debug mode.

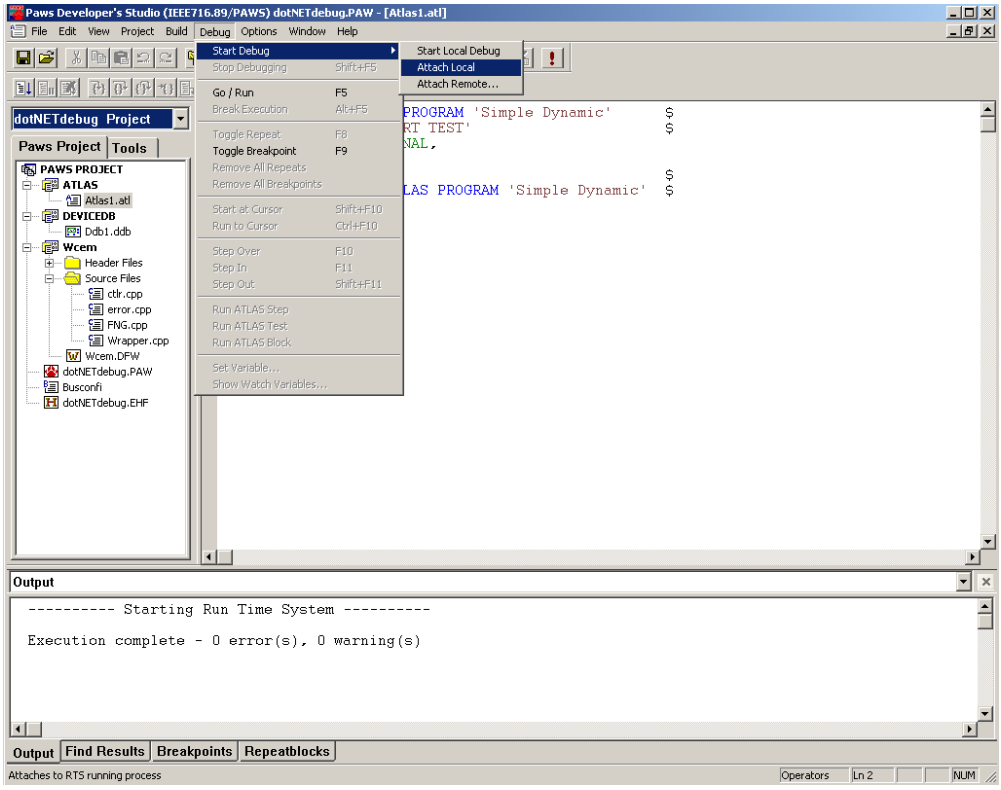


4. This will put the Wrts at the beginning of the TPS, ready to start.

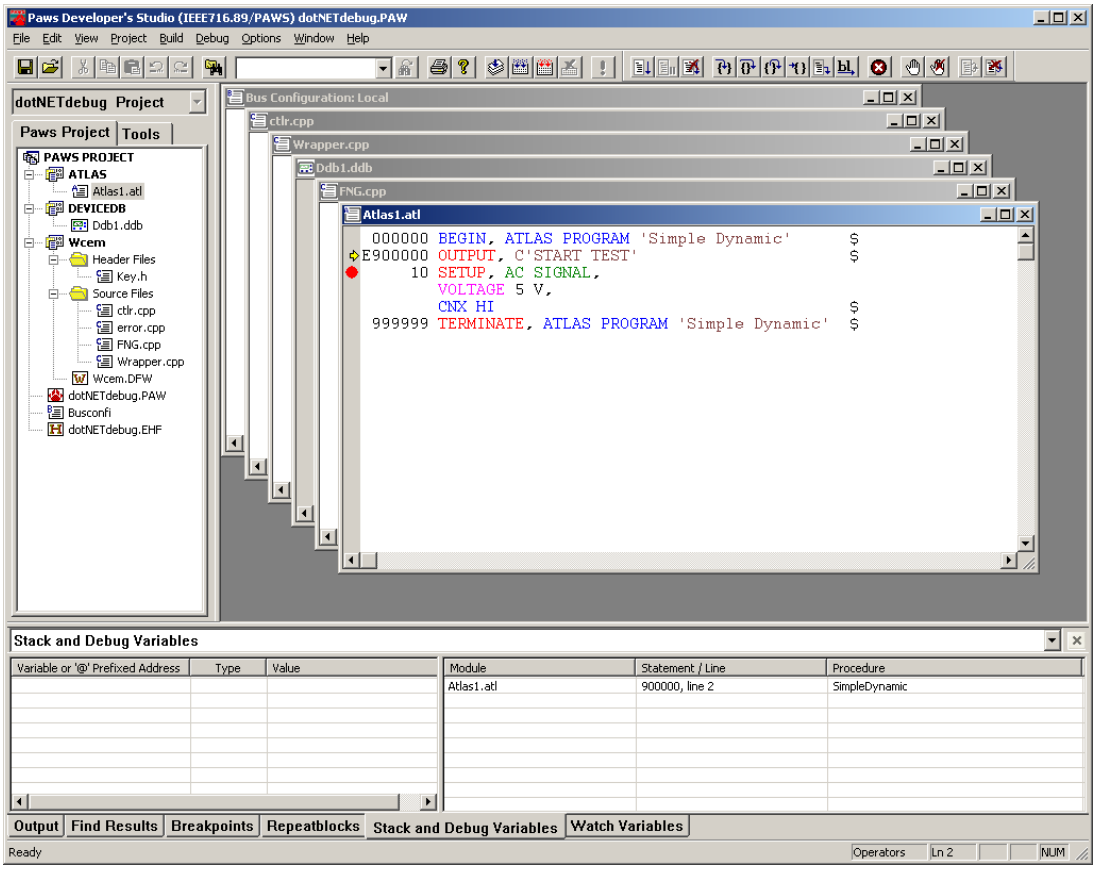
Note: Before you reach this point, the MSVS may have stopped at a breakpoint that you placed in the Wcem.dll code. This is not the case in this example.



5. Now, from the Paws Studio, go into **Debug/Start Debug/Attach Local**



6. This will lead to the following behavior in Paws Studio. In order for this to work you have to have the Wrts running. In this case, Wrts will have been launched from the MSVS .NET.



7. You may now debug both the Wcem.dll and the TPS at the same time: The Wcem.dll from the MSVS .NET and the TPS from the Paws Studio. When running the TPS, the execution of the TPS will be stopped at either breakpoint in the TPS or in the C++ code.