

TYX Corporation

Productivity Enhancement Systems

Reference	TYX_0051_4
Revision	1.1
Document	MSDebug2.doc
Date	January 14, 2003



How to debug with MSVC++6.0

1. Studio Version:

This document is for Studio **1.17.0** and later. The pictures below have been made using version **1.21.2**.

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

This document assumes that the user has some knowledge of Studio and **MSVC++6.0**.

2. How to debug the Wcem.dll from the MS compiler:

We can utilize Microsoft Visual C++ 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 and device database 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 -----
```

The busconfi file should be as follows:

```
----- busconfi -----
; IEEE-488 Bus Configuration File -

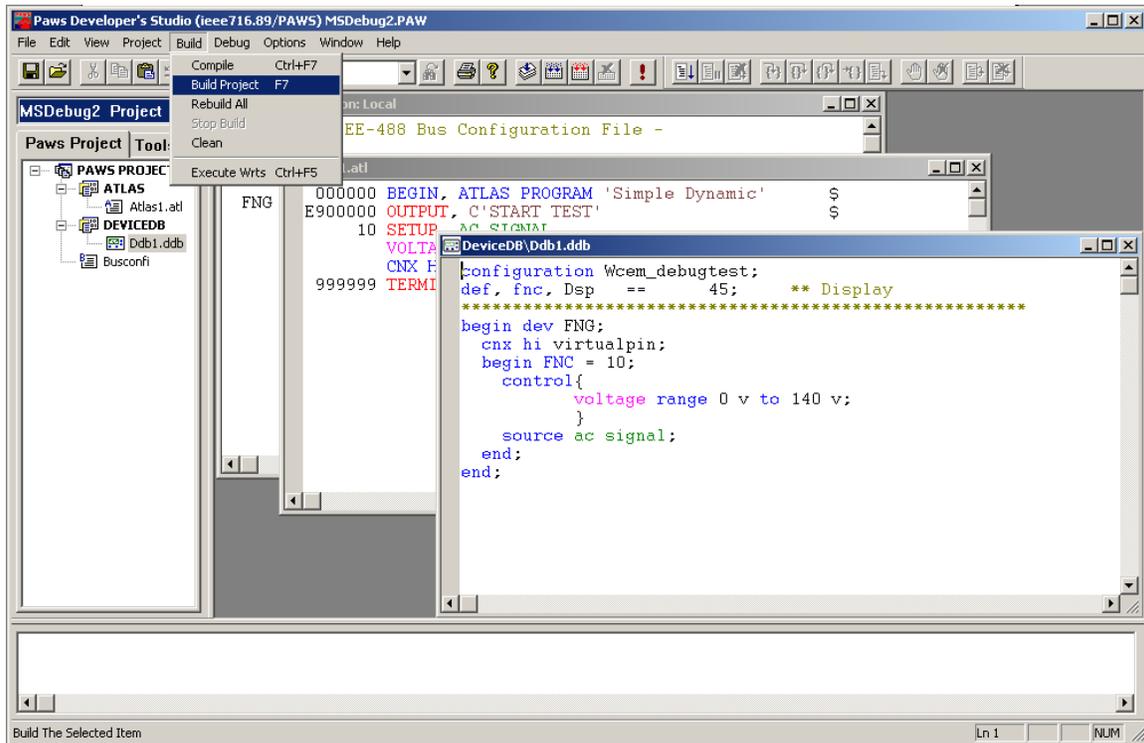
"Channel" 2

FNG BUS 2 MLA 11 MTA 11
----- end of busconfi-----
```

Put those files in a Paws project called **MSDebug2**. In our case, we will put the project under **C:\usr\paws\MSDebug**.

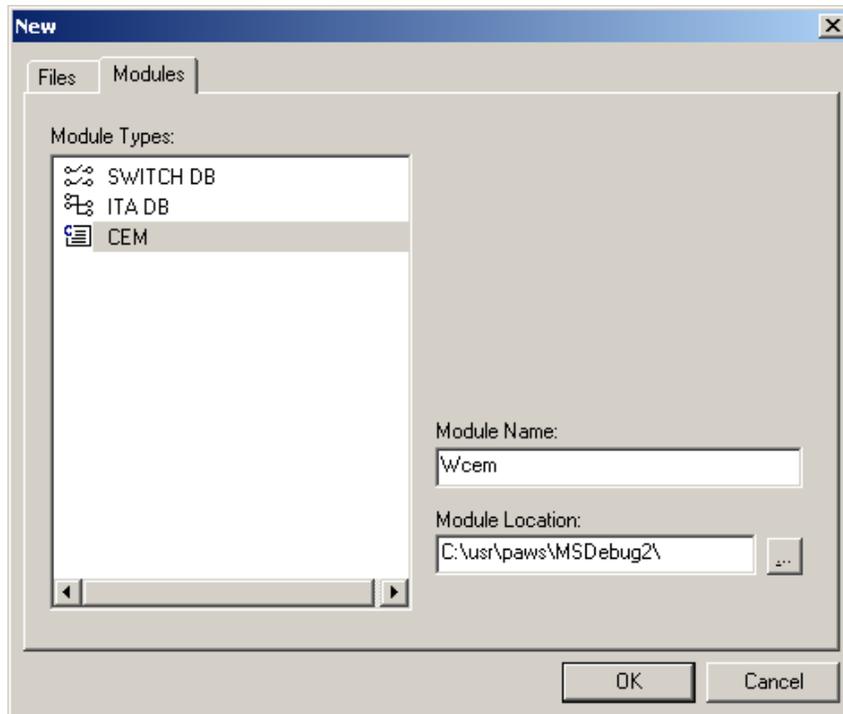
You may place the **Busconfi** either in the local directory or you may place it in the station subdirectory in your Station.

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

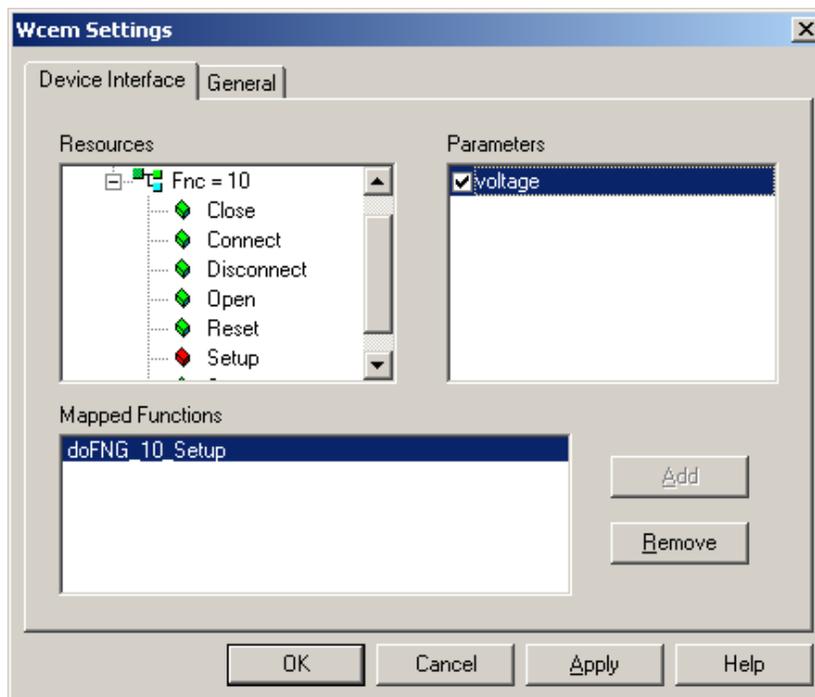


- Now that the device database is built, you can create a **Wcem** module as shown below. You can call the module **Wcem** and redefine the path to locate the module in the **.paw** subfolder rather than in **.\Wcem**.

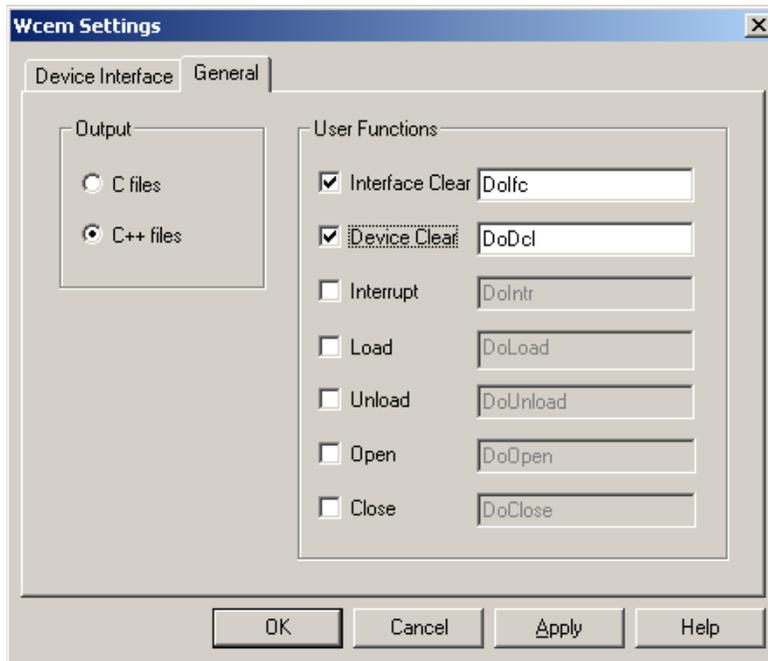
Note: By default, when you add the name for the **Module Name**, the files will be put into Module name subfolder into the **.paw** folder, so you will need to delete the module name folder extension that gets added automatically in the **Module Location** edit box.



- 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. You will want to check the **voltage** parameter:



- You may wish to add some features under **General** such as shown below. This will be responsible for an additional C++ file generated by the **Wcem Wizard** called **ctrl.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. In this case, the list of files is the following: **Wrapper.cpp, key.h, error.cpp, ctrlr.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"
#include <limits.h>
#include <float.h>
//BEGIN{DFW}:FNG:10:0
int doFNG_10_Setup (double VOLT)
//END{DFW}
{
    Display("Entering doFNG_10_Setup \n");
    return (int) 0;
}
-----

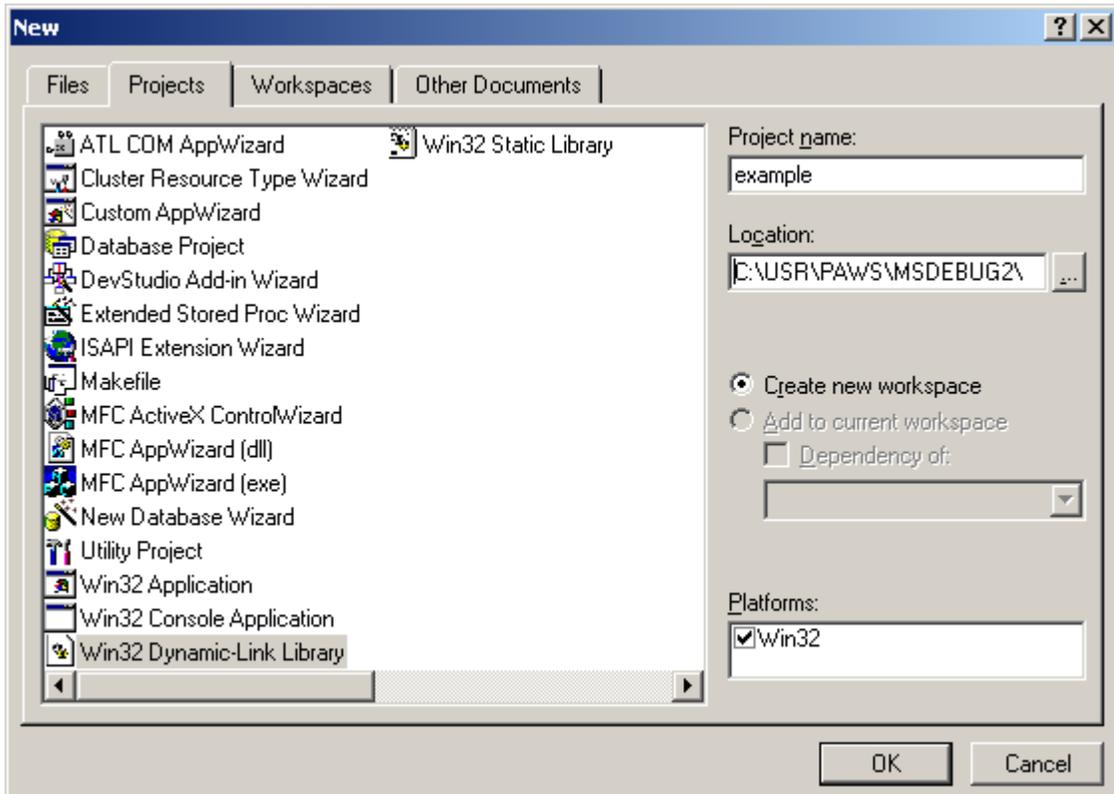
```

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 MS Studio, or it might cause some problems when debugging the **dll** from the MSVC++6.0 generated by the MS Studio.

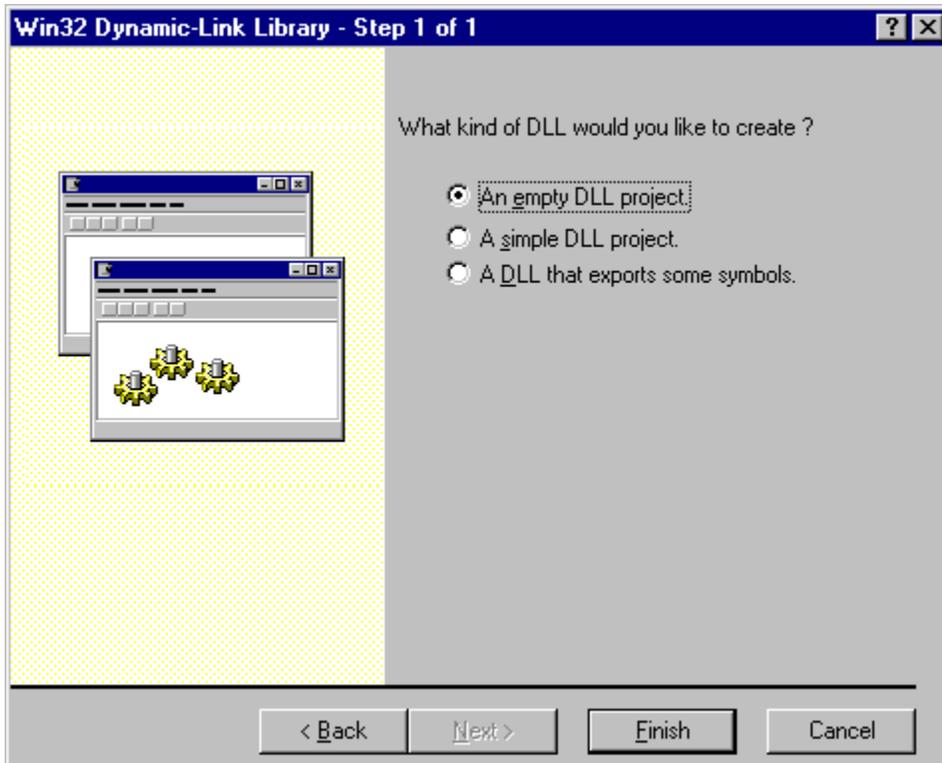
3. The MSVC environment:

Now, we are ready to move on to the MS environment. This example uses the support of **MSVC 6.0**.

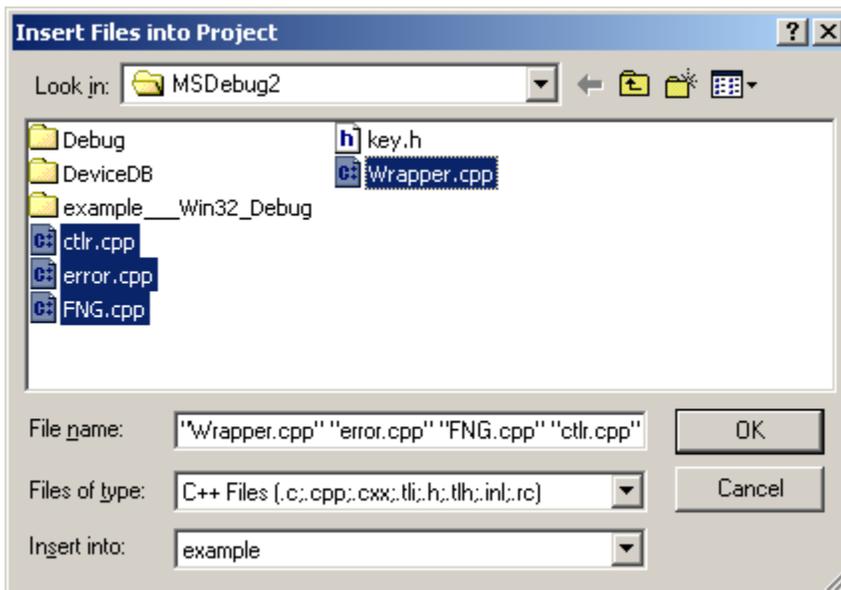
- From the **MSVC++6.0** under **File**, chose **New...** and select **Projects**. Fill in the **Project name**, Project type and **Location** as shown below.



- After pressing **OK**, select **An empty DLL project**. As shown below.

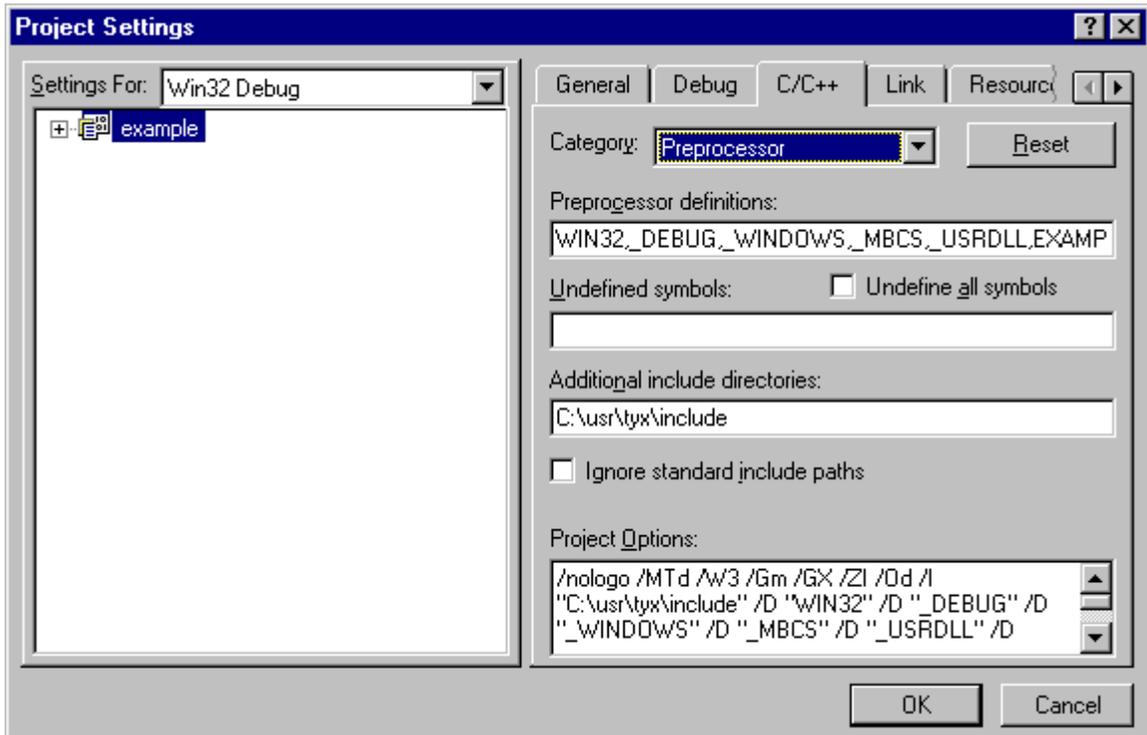


- Press **Finish** and then **OK**.
- Now, select **Project, Add to Project and Files....** This will open the window below. Select the TYX Cem C++ files one after another. Here, we will select **Wrapper.cpp, error.cpp, ctr.cpp** and the **FNG.cpp** files.

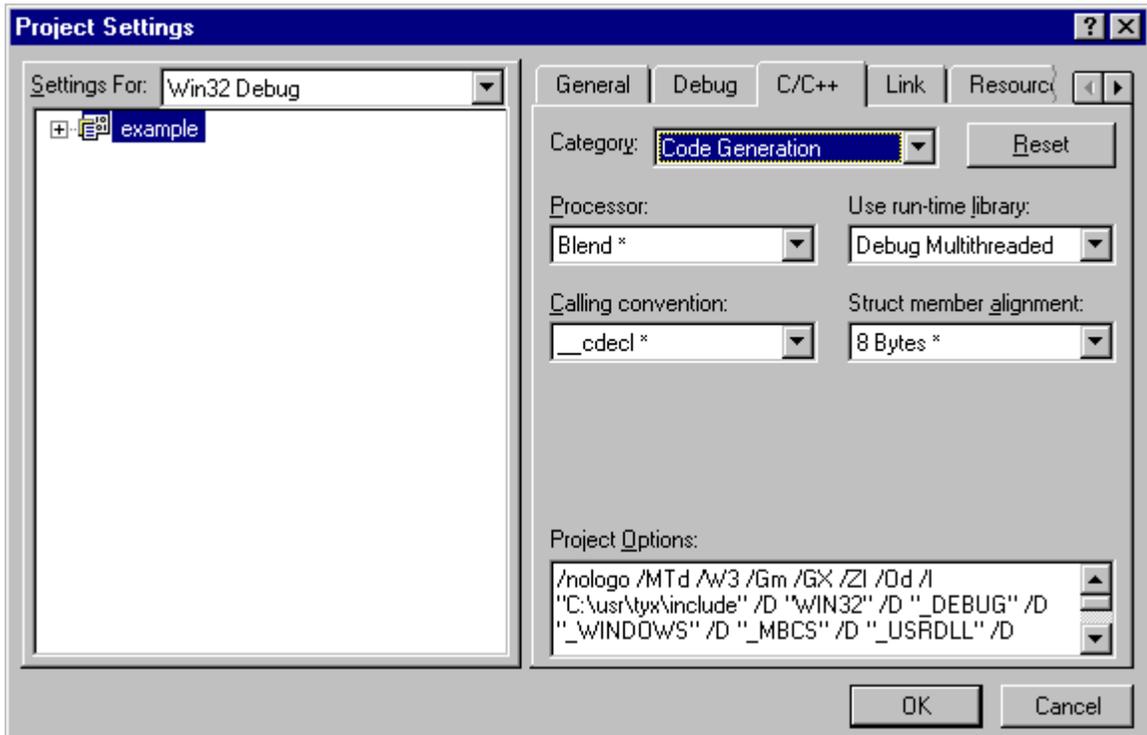


- Press **OK**.
- We now need to add basically all the settings that you would have in the TYX Wcem settings. The first thing will be to include the additional path of all the header files that you wish to include in your project. In this case, one that is unavoidable is the **cem.h** file from TYX. The path is usually

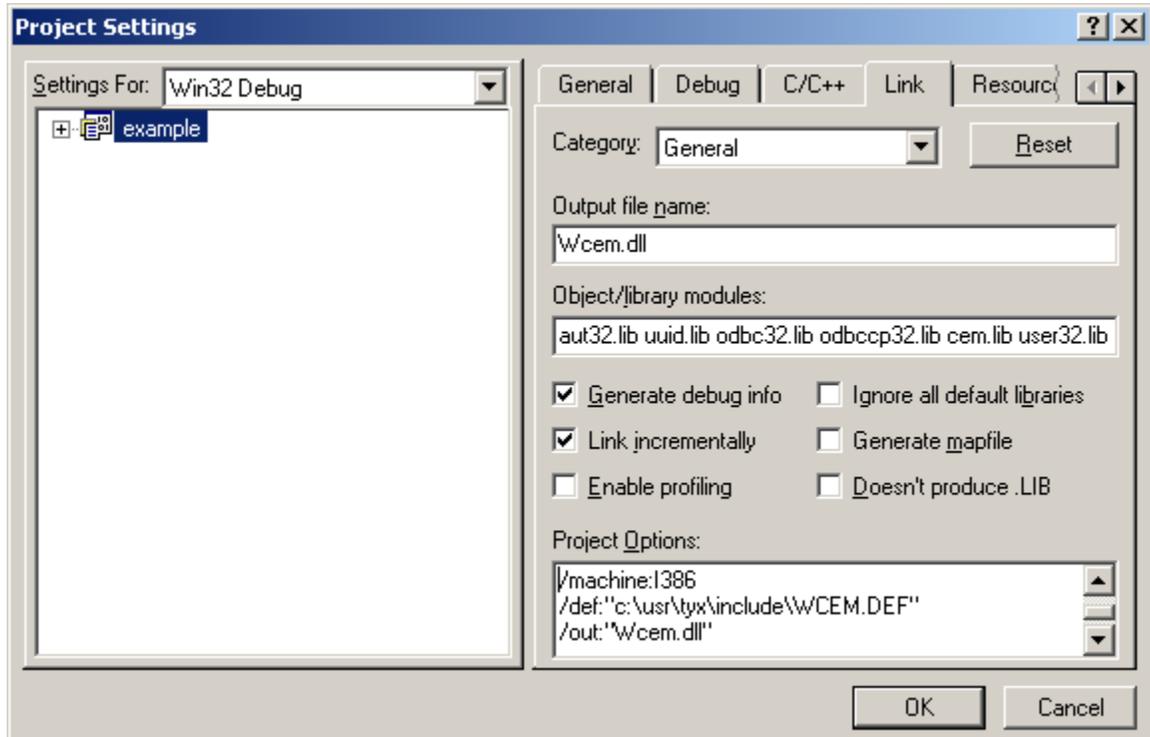
C:\usr\tyx\include. This can be done going into **Project->Settings->C/C++** in the **Preprocessor Category** as shown below:



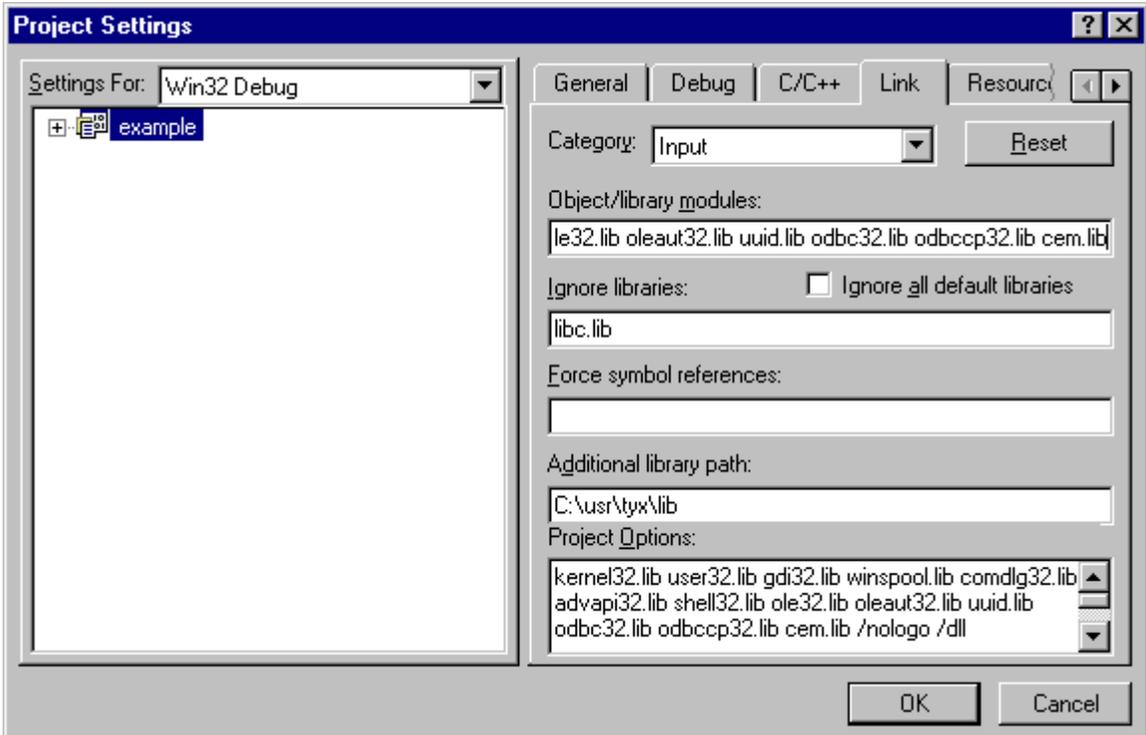
- **Note:** For greater safety, make sure that the **Use run-time library** option under **Code Generation** is set to **Debug Multithreaded** as shown below.



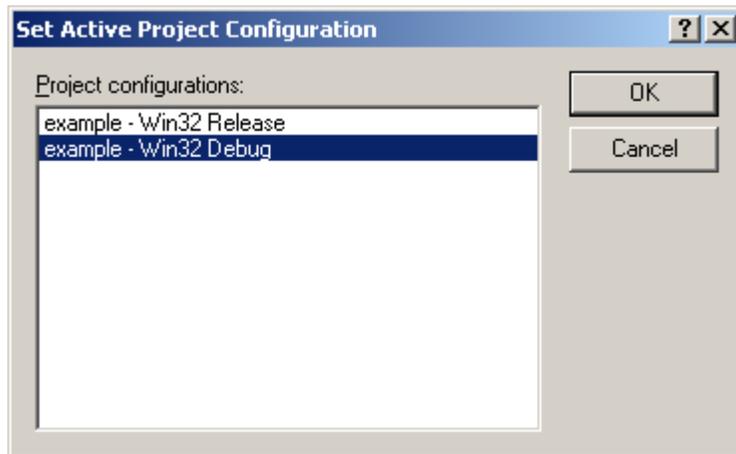
- Under the **General** Category in **Link**, we need to include all the libraries that will be addressed by our project. In this case, we need to add **cem.lib** and **user32.lib** at the end of the libraries already included as shown below. You may wish to delete any additional libraries that aren't used, but it will not affect your project if you leave them there.
- 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, hence locally.
- Under the **Project Options**, 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**.



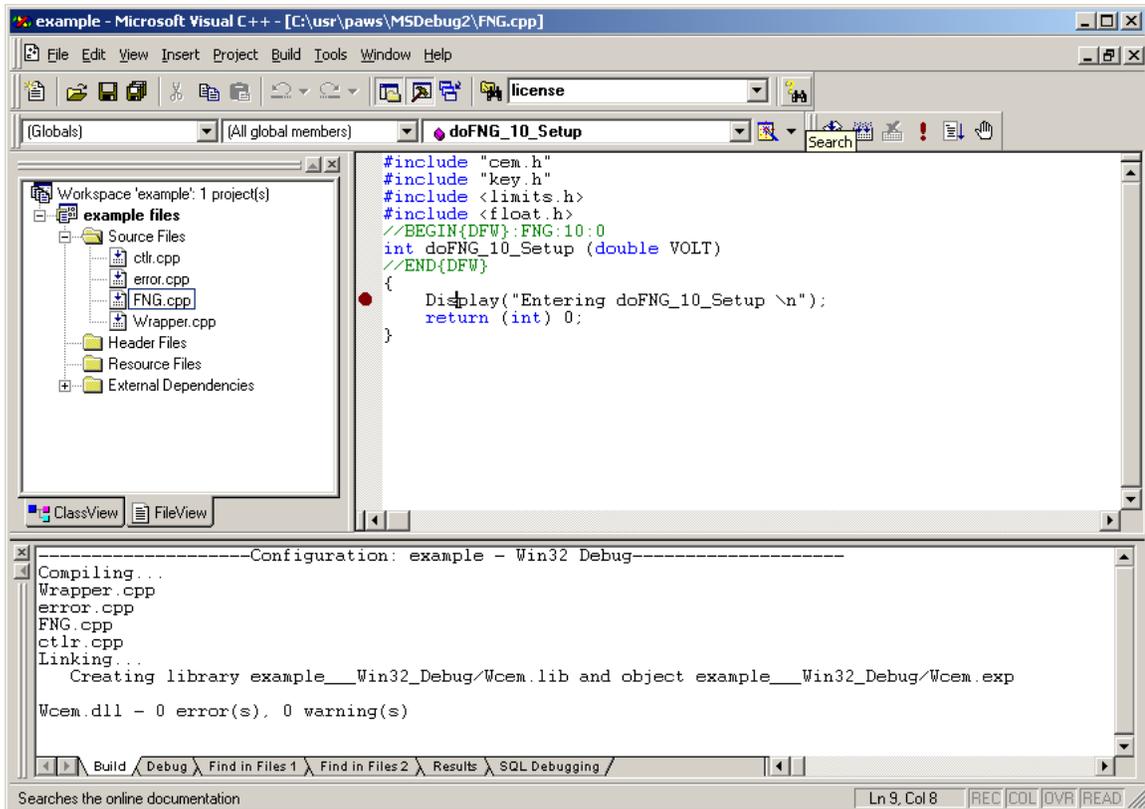
- Under the **Input** Category in **Link**, we need add the additional paths for the additional libraries as show below.
- We also need to ignore the **libc.lib** library to avoid redefinition warnings.



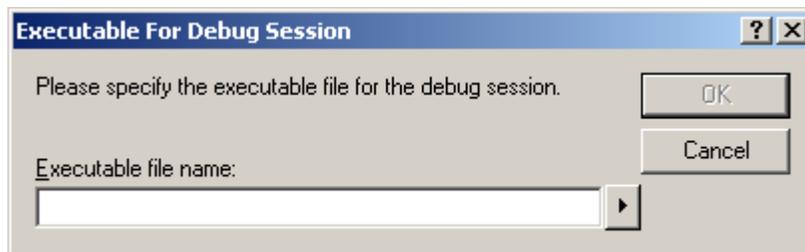
- The next step is to set up the environment so that the Wrts can be started from the MS studio. Under the **General** Category in **Debug**, you should look for the Wrts on your system for **Executable for debug session**. Typically, it's located under **C:\usr\tyx\bin**.
- Press **OK** on the **Project Settings** window.
- You are now ready to build the project and start it in debug mode. Make sure that the active configuration is the debug version, which is the default one (under **Build/Set Active configuration...**)



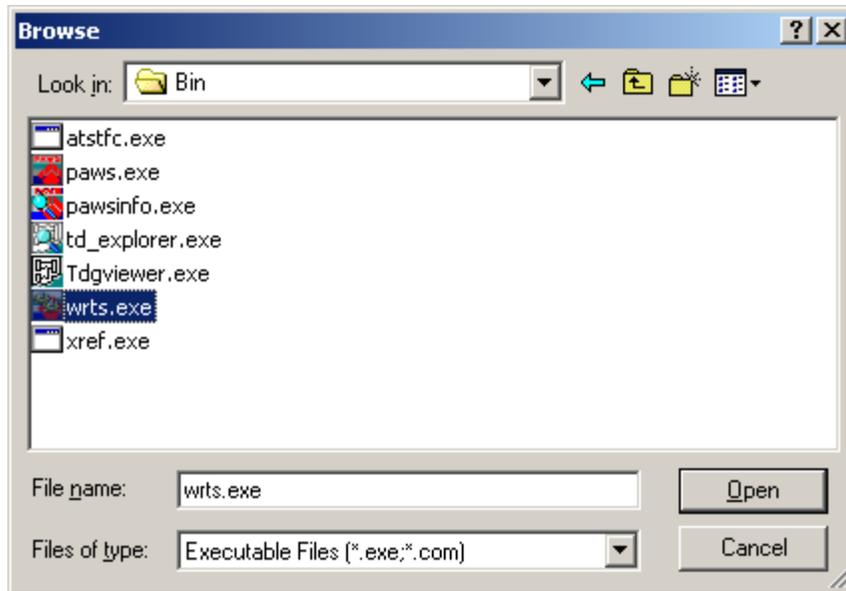
- Build the project by pressing **F7**.



- 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 in debug mode via **F5** or via **Build->Start Debug->Go**. You may see the following window if you missed the step regarding setting the **Executable for Debug Session**. The first time, you will see the following window



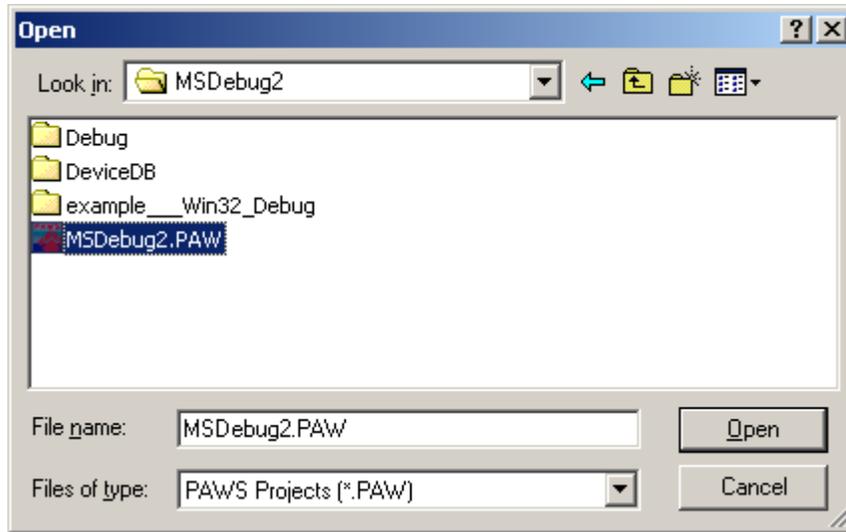
- That is the window that will determine which executable to launch that will load the **Wcem.dll**, which is the output of the MSVC project. Click on the arrow next to the edit box and select **Browse**. Select **Wrts.exe**. This exe is usually located under **c:\usr\tyx\bin**



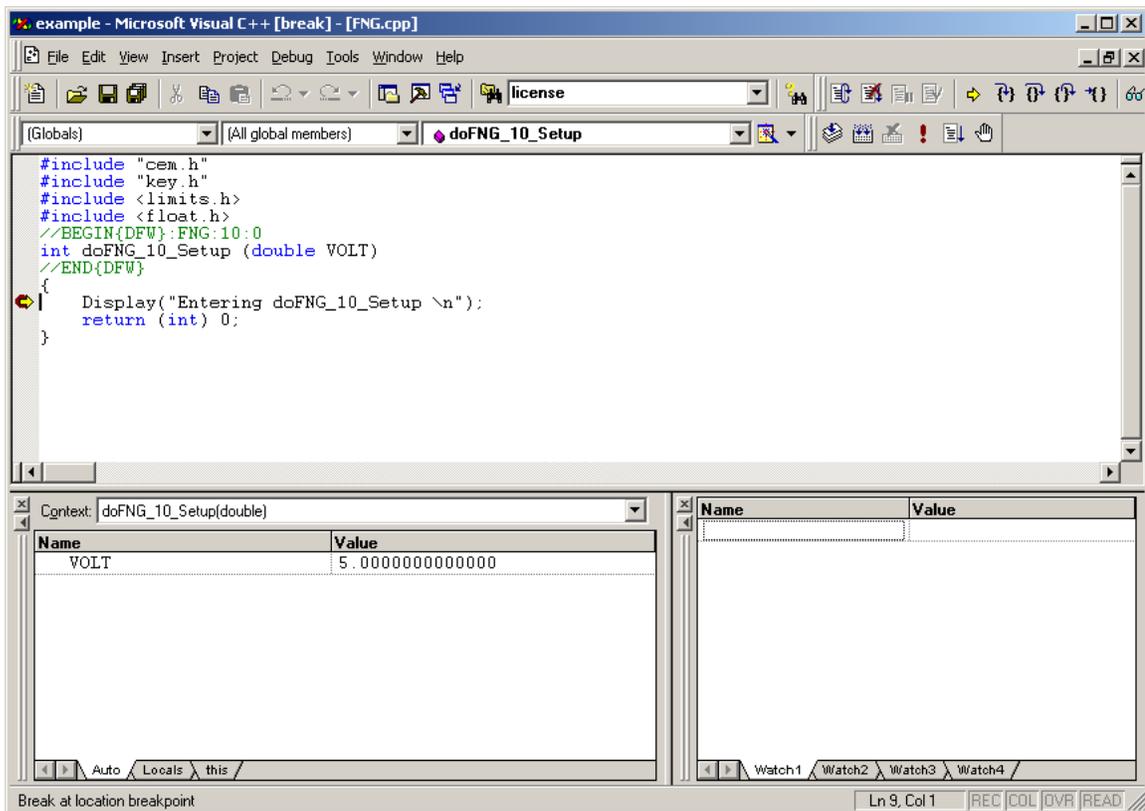
- Click on **Open**, and then **OK**.
- Click **OK** on the message box that tells you that the Wrts does not contain debug information after checking the checkbox in order to not see this window the following time. This is the window that you will see if you didn't miss the step regarding setting the **Executable for Debug Session**.



- Once the Wrts started, you should select the **MSDebug2.PAW** project as shown below



- Run the project from the Wrts and MSVC 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.

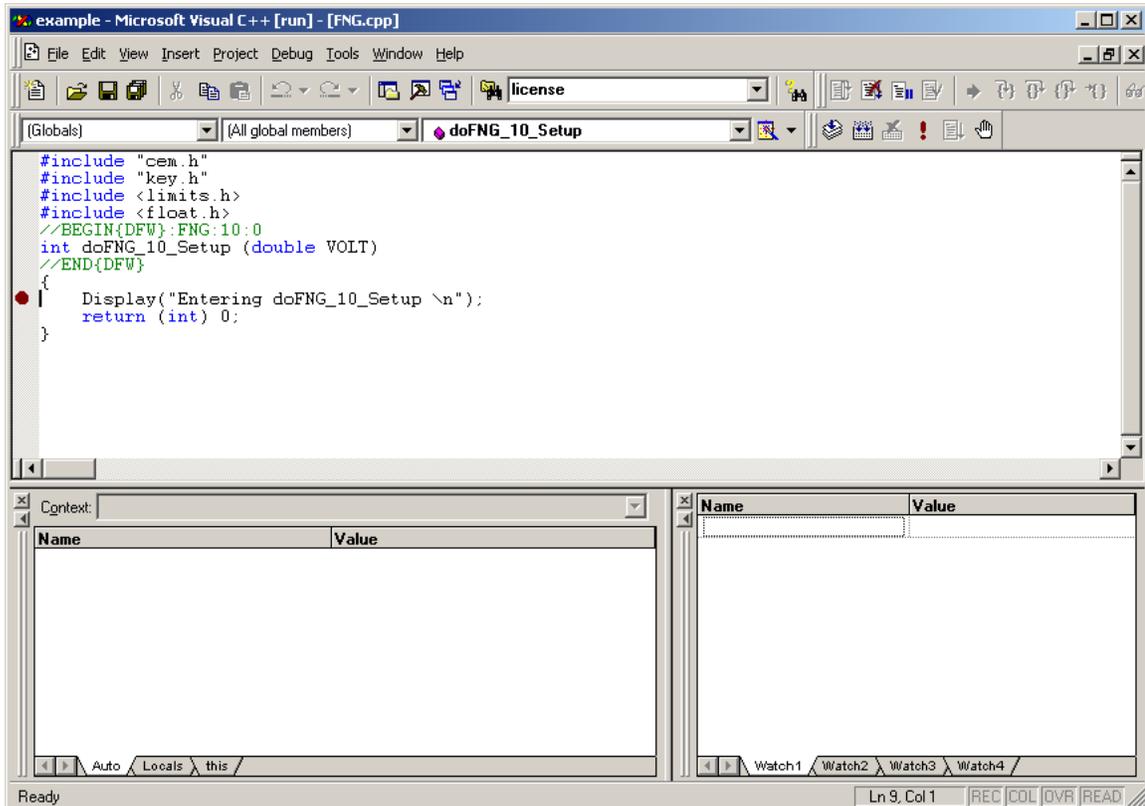


- As we can see, the value specified in the **Atlas** for the **Voltage** value is visible.
- **Note:** If you are having problem with the breakpoint, it will be because you will have built the Wcem.dll with the studio which has a release configuration and cannot be debugged. In order to overcome this problem, you need to do a rebuild all from the MSVC++6.0 compiler in order to overwrite the one generated by the Paws Studio.

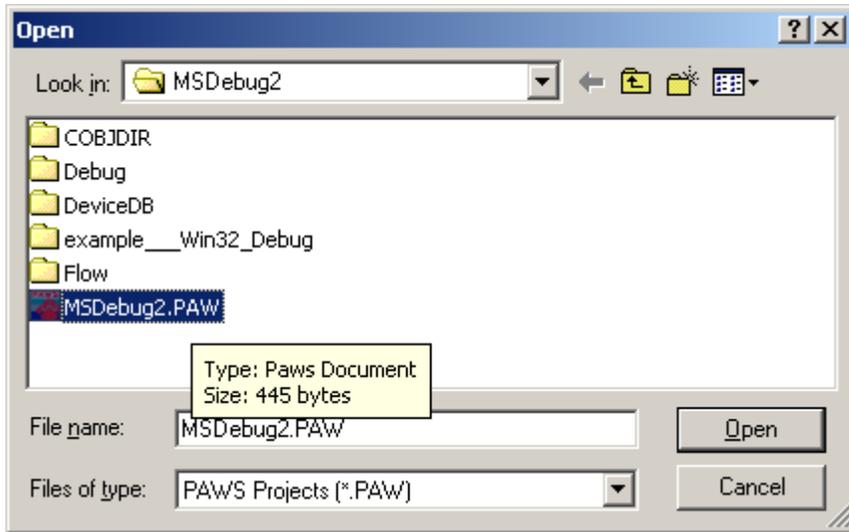
4. How to debug the Atlas while you are debugging the Wcem.dll driver?

This is a simple procedure.

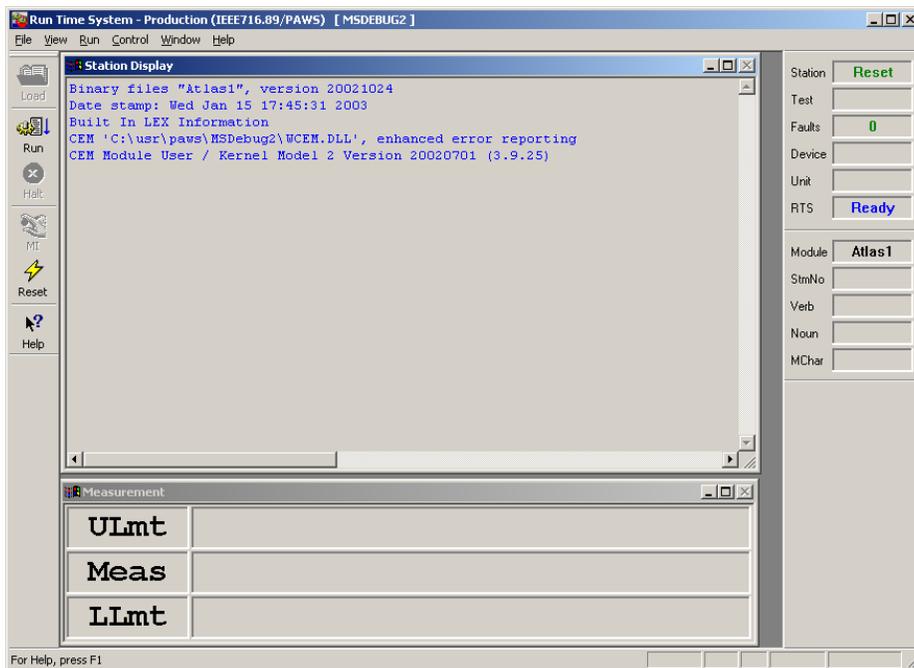
1. You need to build the **Wcem.dll** in debug mode.
2. You need to start the Wrts from the MSVC++6.0 with **F5**. This will allow to debug the **Wcem.dll**.



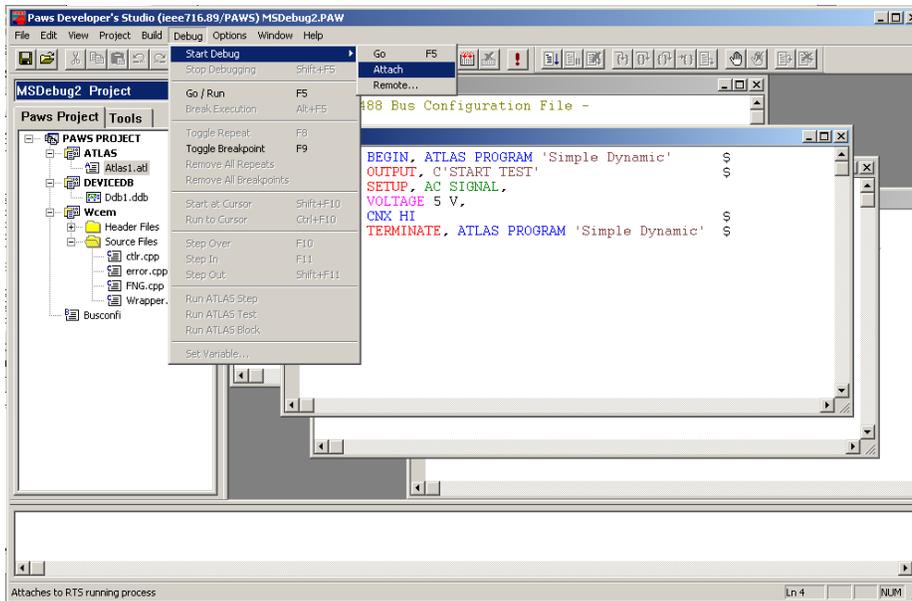
3. Load the project that you want to run in debug mode.



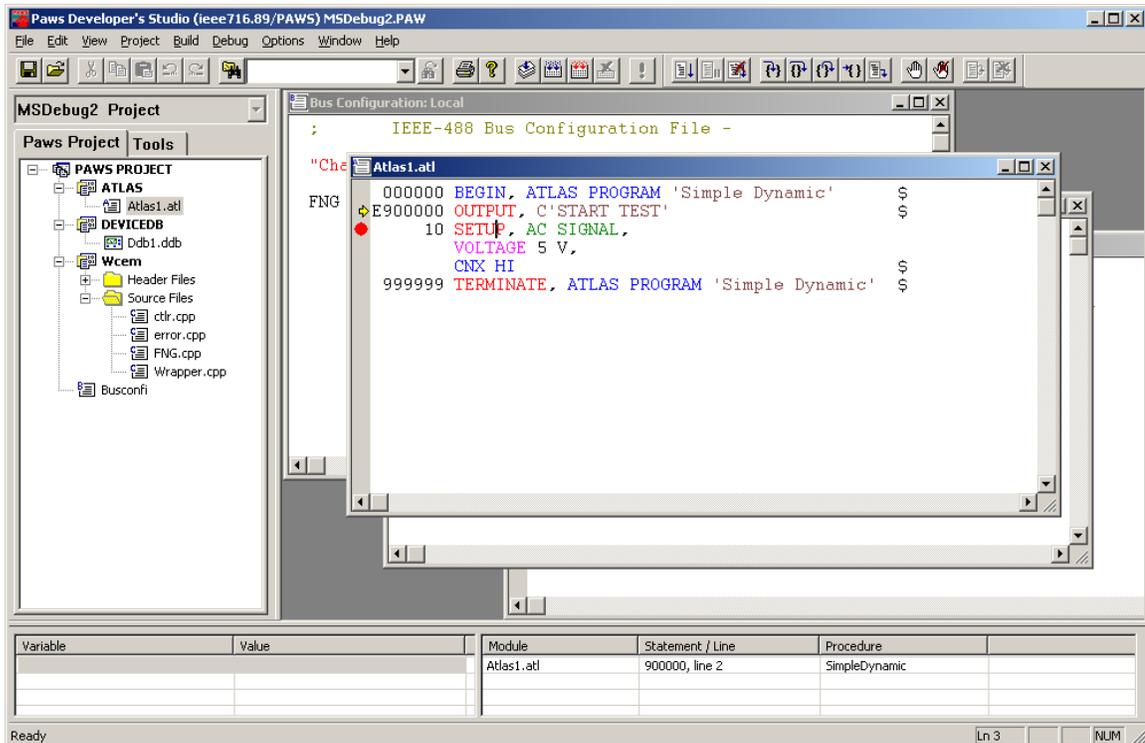
4. This will put the Wrts at the beginning of the TPS, ready to start.
Note: Before you reach this point, the MSVC may have stopped at a breakpoint that you placed in the Wcem.dll code. This is not the case in this example.



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



- 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 MSVC++6.0.



- You may now debug both the Wcem.dll and the TPS at the same time: The Wcem.dll from the MSVC++6.0 and the Tps from the Paws Studio. When running the Tps, the execution of the Tps will be stopped at either breakpoints in the Tps or in the C++ code.