TYX Corporation

Productivity Enhancement Systems

Reference	TYX_0051_12
Revision	1.0
Document	COMNam.doc
Date	Jan 23, 2004

How to create a COM-NAM dll?

This document will help with creating the COM-NAM component (server) for Paws RTS (client).

TYX PAWS Studio version used: **1.26.0**

MSVC++ version used: **6.0**

Requirements: **Studio 1.10.x or above. Backward and forward compatibility between the COM build with one version of the Studio library and other Studio versions is not guarantied.**

Introduction: **This document describes how to create a Non-ATLAS Module (NAM) as a Common Object Component (COM-NAM). We use Active Template Library (ATL) to produce a COM component also called a server. The application that uses COM is Paws RTS – a client. In this scenario COM component is stored in a .dll file as opposed to .exe, which will be covered in another document. The last part of the document is dedicated to the TYX PAWS Studio environment and Atlas example in particular.**

Dll versus Exe:

1. **Advantage of Dlls versus Exe:**

    - **The Dll uses fewer resources than the Exe.**
    - **The Dll can easily remain on top of the Wrts.**
    - **The accelerators are directly passed on to the Wrts without additional code. When the Exe has the focus, it will need some code in order to pass on the accelerator destined for the Wrts.**

- **The message loop for painting the GUI is taken care of by the Wrts. For the Exe, the code needs to be placed into the Exe which may be done by the wizard to some extent.**

- **The Dll will be slightly faster than the Exe. This will however only make a difference for repeated transfer of a large amount of data.**

2. **Advantage of the Exe versus Dll:**

    - **The Exe is more isolated than the Dll and if it crashes, it will not affect the Wrts. This may be an issue if the Exe links to code that is not stable.**

    - **The Exe can be moved in front or behind the Wrts at will.**

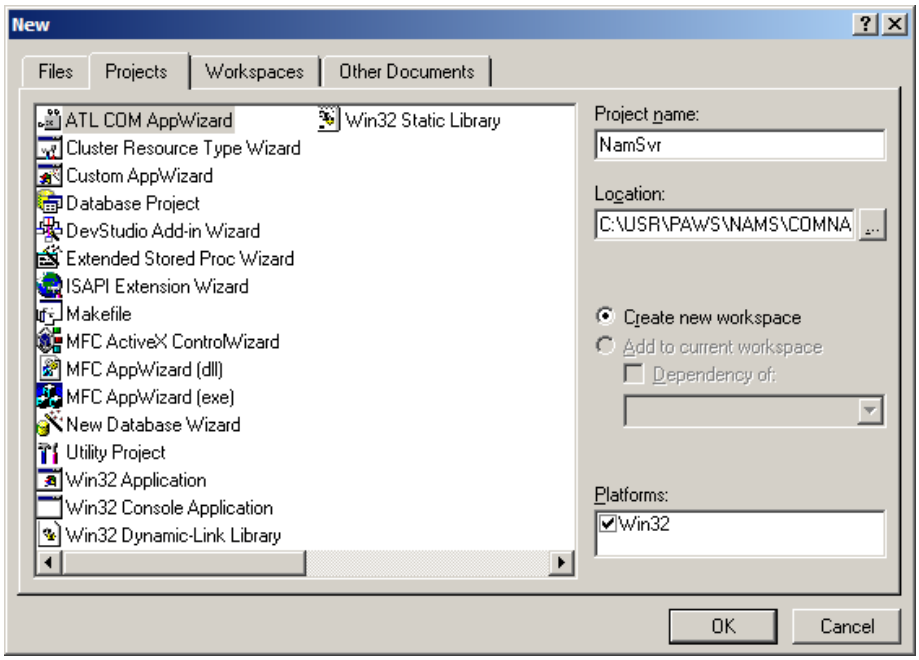
    - **The Exe can be run remotely.**

1 Generating a dll with MSVC++ 6.0

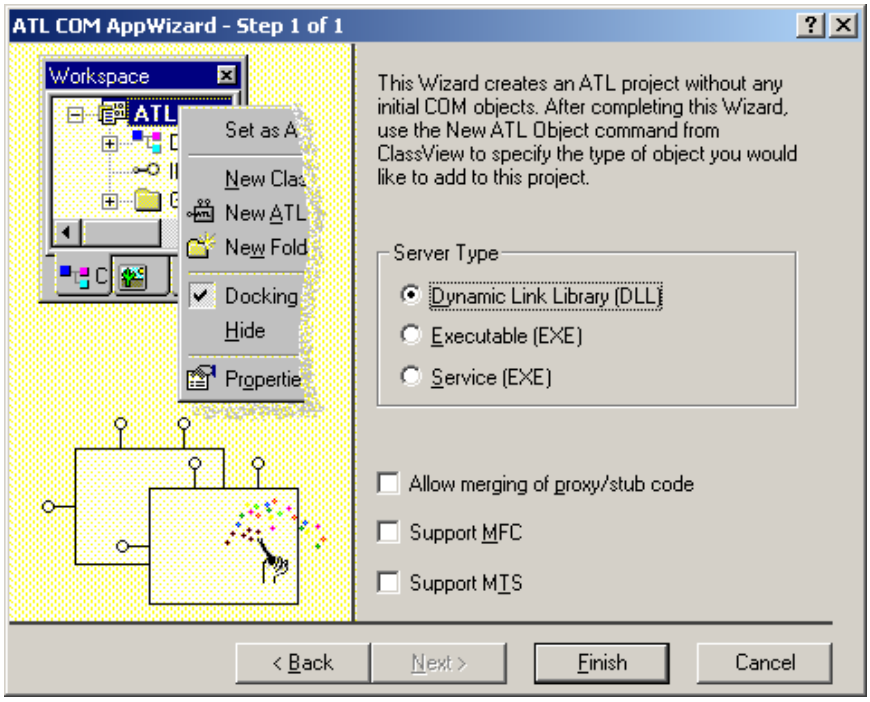
1.1 Starting a simple ATL COM Application:

- From **File | New**…, select an **ATL COM Application** and select a project name **NamSvr** as seen below:

- Click on **OK**

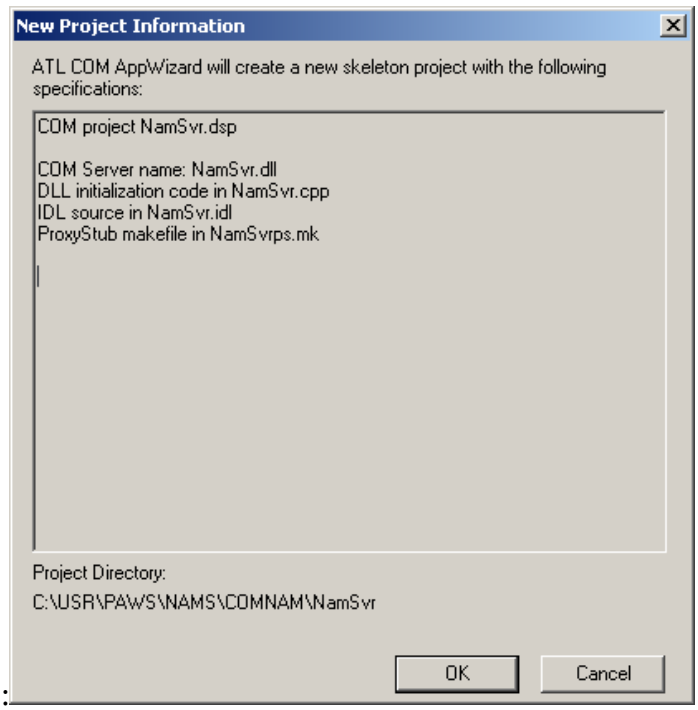


- Select **Dynamic Link Library (DLL)** in the following window.

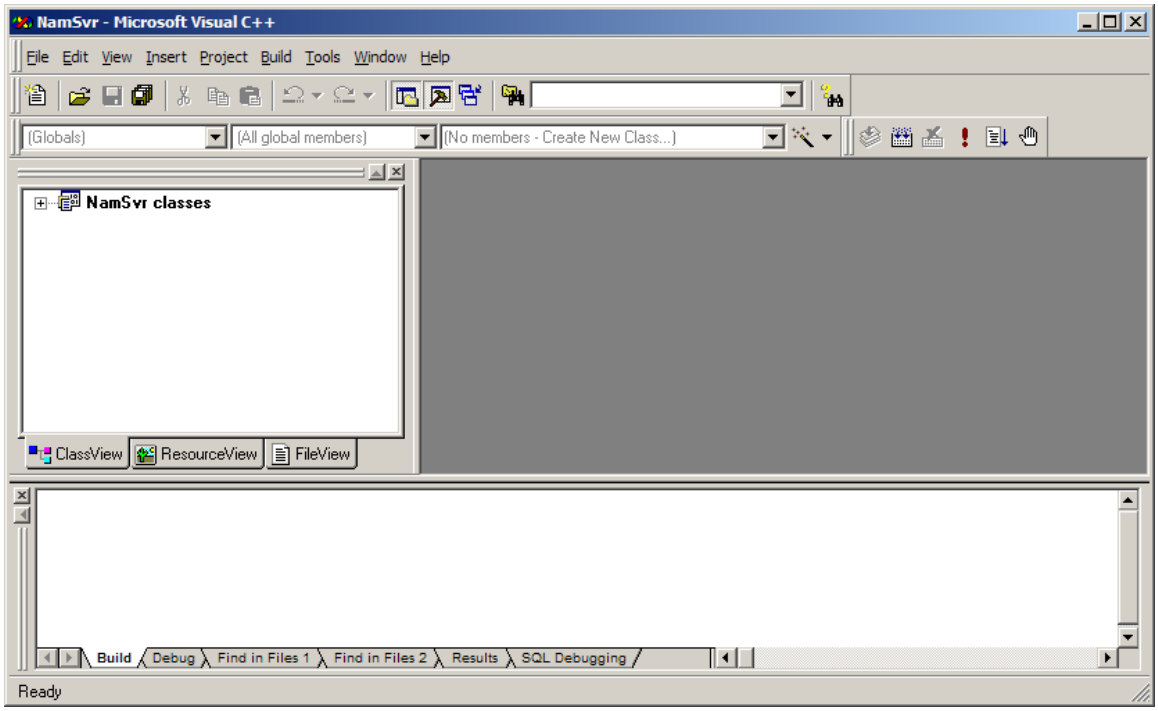


- Click on **Finish**.
- Click **OK** on the following window:

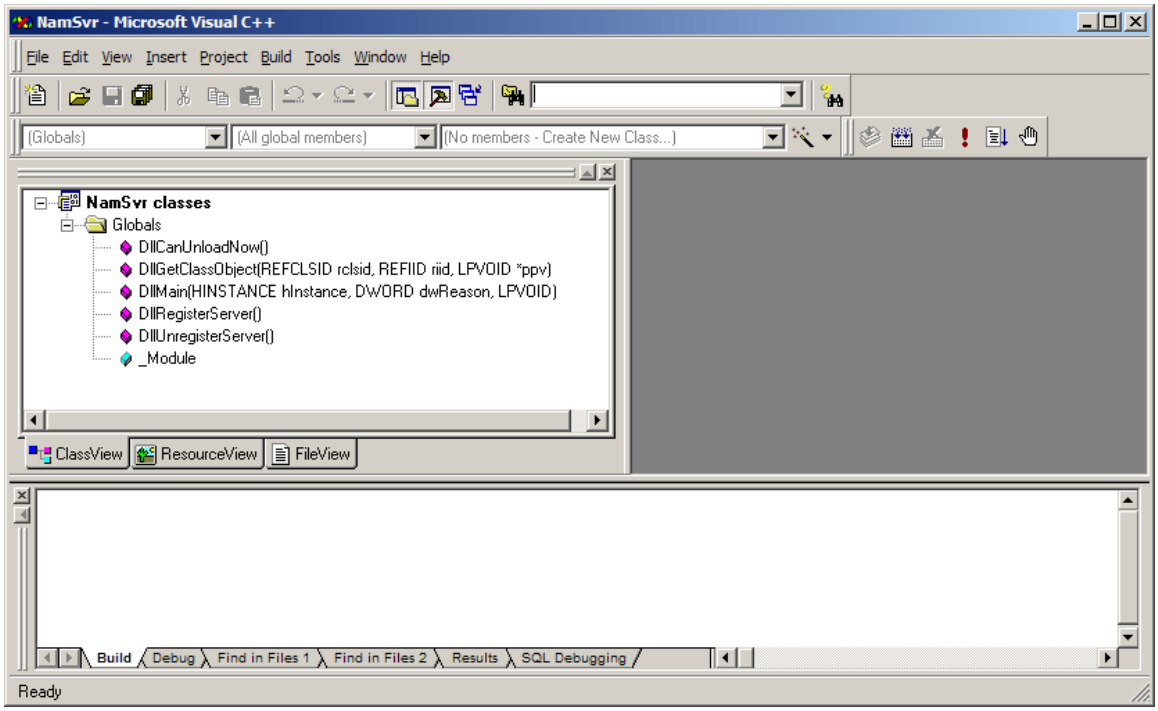
:



- Image below shows, how the project should look like.

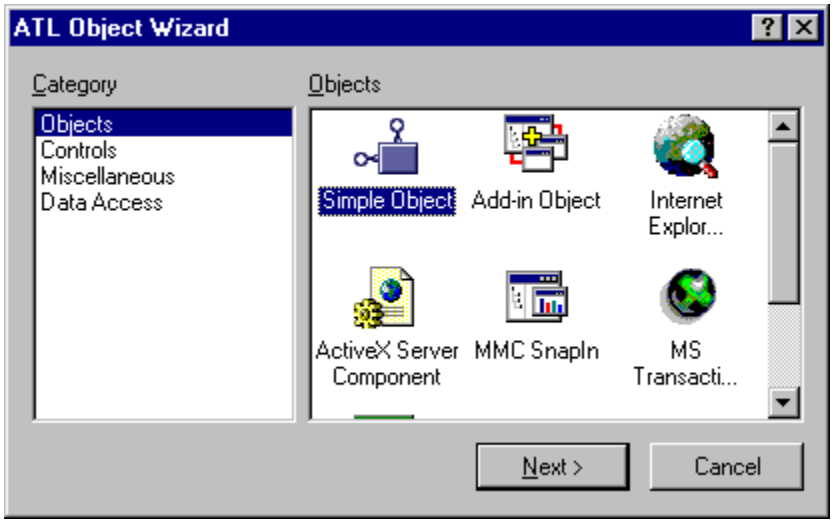


- Select the '+' sign next to the **NamSvr classes** and **Globals** folders to see all the methods generated automatically by the Wizard.



1.2 Adding the ATL object

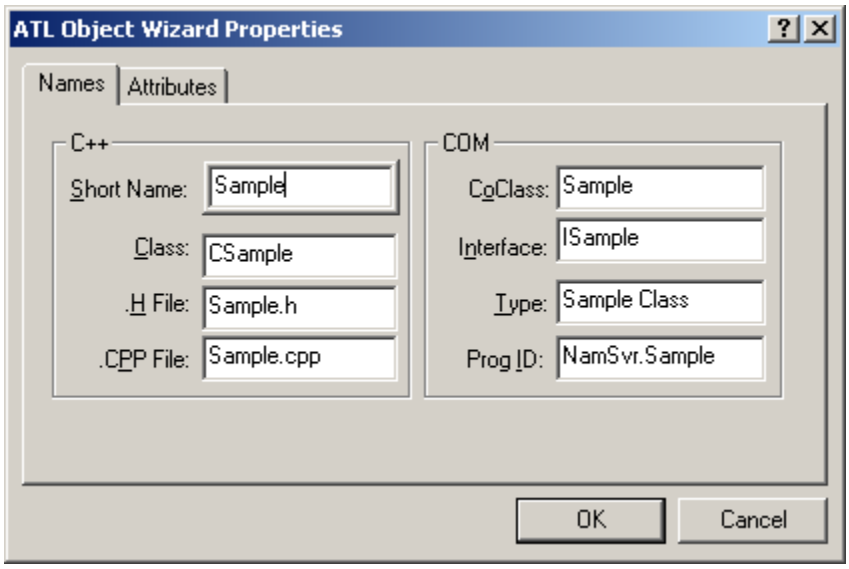
- Go to the **Insert** on the Menu Bar and select **New ATL Object**…
- You should see the following window:



- Select **Simple Object** and click on **Next**.

- You will then see the following window. In the **Short name:** box, enter the name of the ATL object that you wish to create. The other boxes will fill in by themselves. In this case, we chose the name **Sample**.

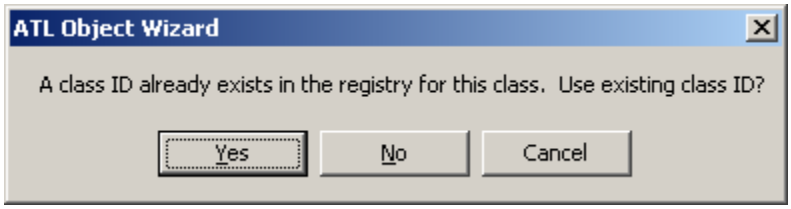
  Also, take note of the name in **Prog ID**: In this case **NamSvr.Sample** (dll name followed by the short name). This will be used as an entry in the Wrts options.



- In the list of attributes, as shown below, all defaults can be acceptable. You may want to add the option of **Support IsupportErrorInfo** and **Custom** for **Interface** as seen below. **Custom** option checked causes the COM interface to support only the **IUnknown** capability, which is sufficient for this project..

- Click on **OK**.

- If you get the following window:

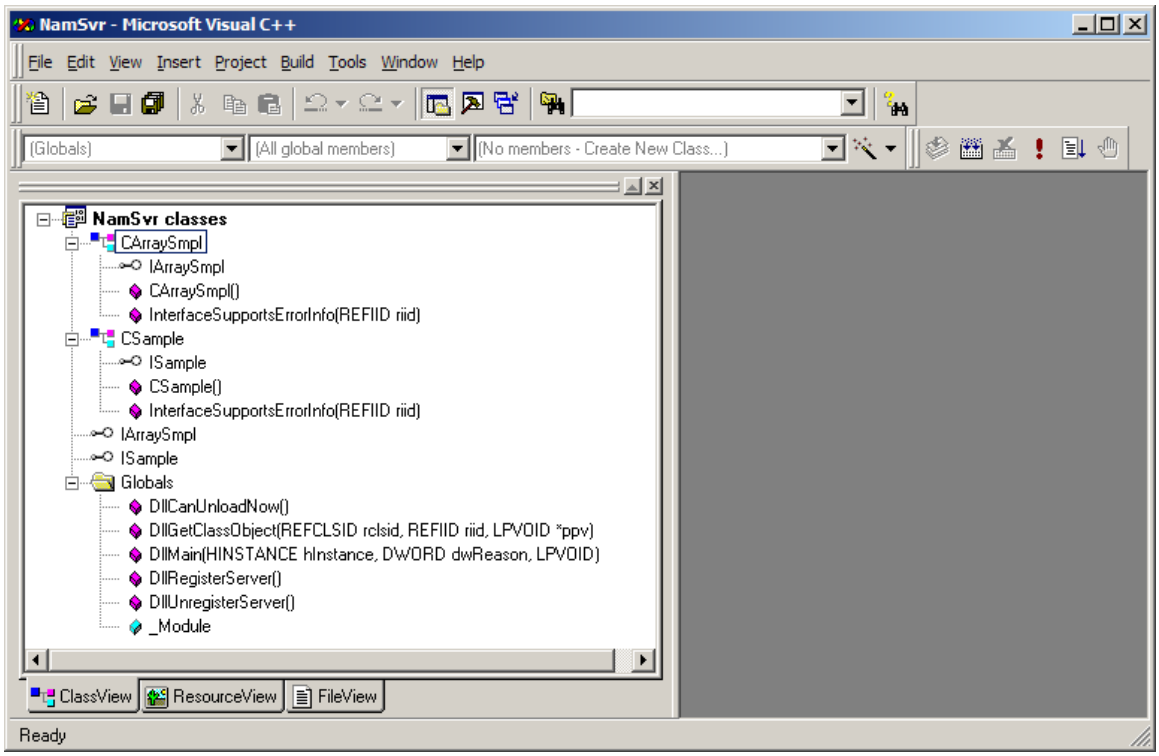


- Click on **Yes**. If not, just proceed.

- Now we are back to the MS Studio.

- You need to repeat the previous steps to add another ATL object. As a short name use **ArraySmpl**.

- At this point you should get the following classes and method under **ClassView**:



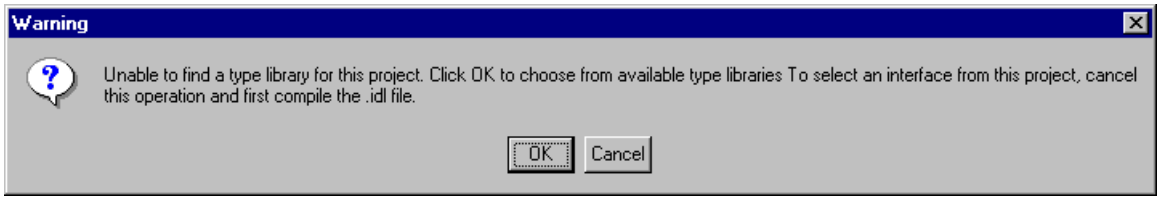
1.3 Implementing COM-NAM interface

- For Wrts to instantiate successfully the COM-NAM object the object must implement the proper interface. That is why the project needs to link to an interface, which is what will be described in the next steps.

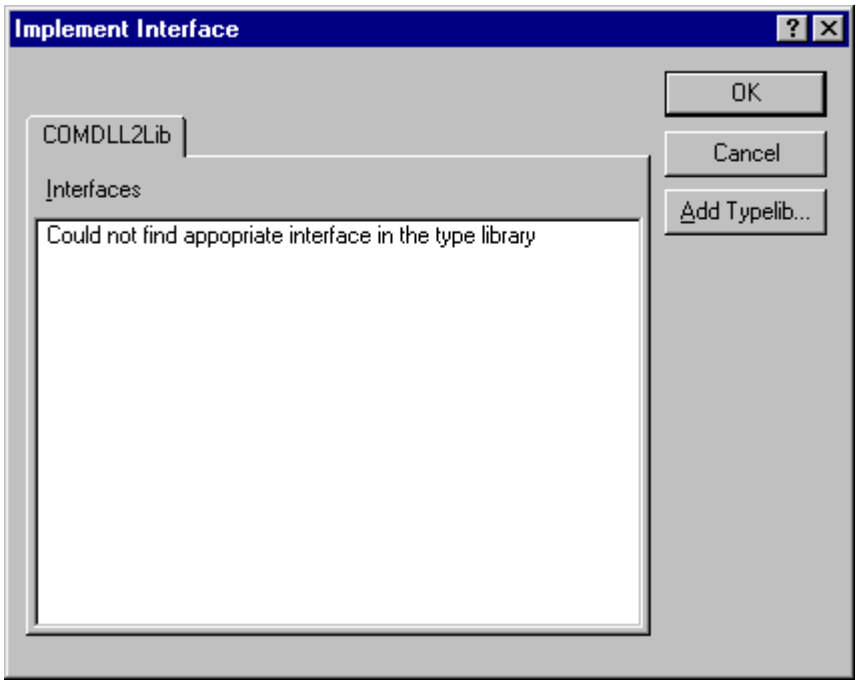
- In the workspace, you should right-click on the **CArraySmpl** Class

- After right clicking on **CArraySmpl**, Select **Implement Interface…** from the drop-down list.

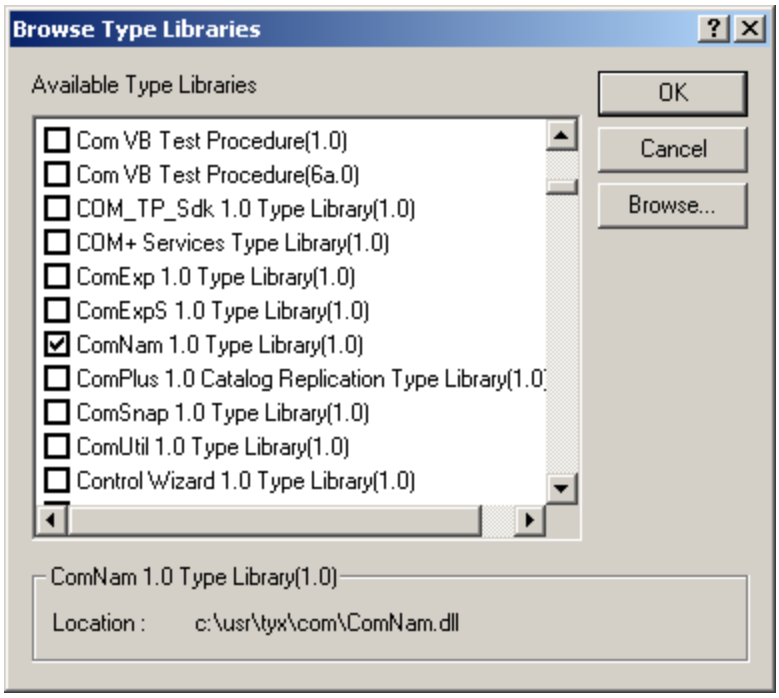
- You will get the following message and then click on **OK**.



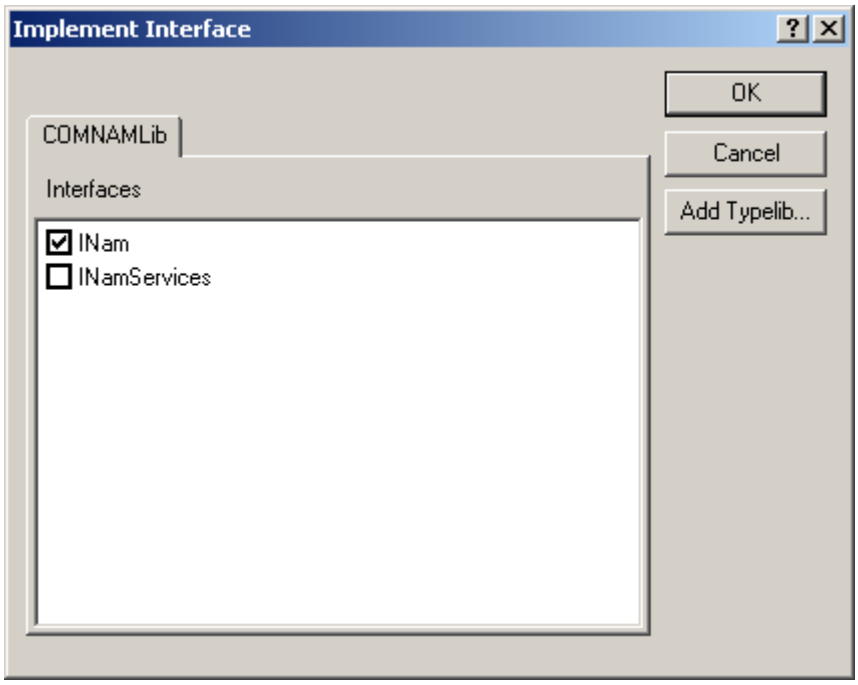
- In the event that you see the following image instead of the previous one, press **AddTypelib…**



- Select the **ComNam** type library and then click on **OK**. If you do not see **ComNam**, it is because you are using a version of the TYX Paws Studio that is older than 1.10.x.

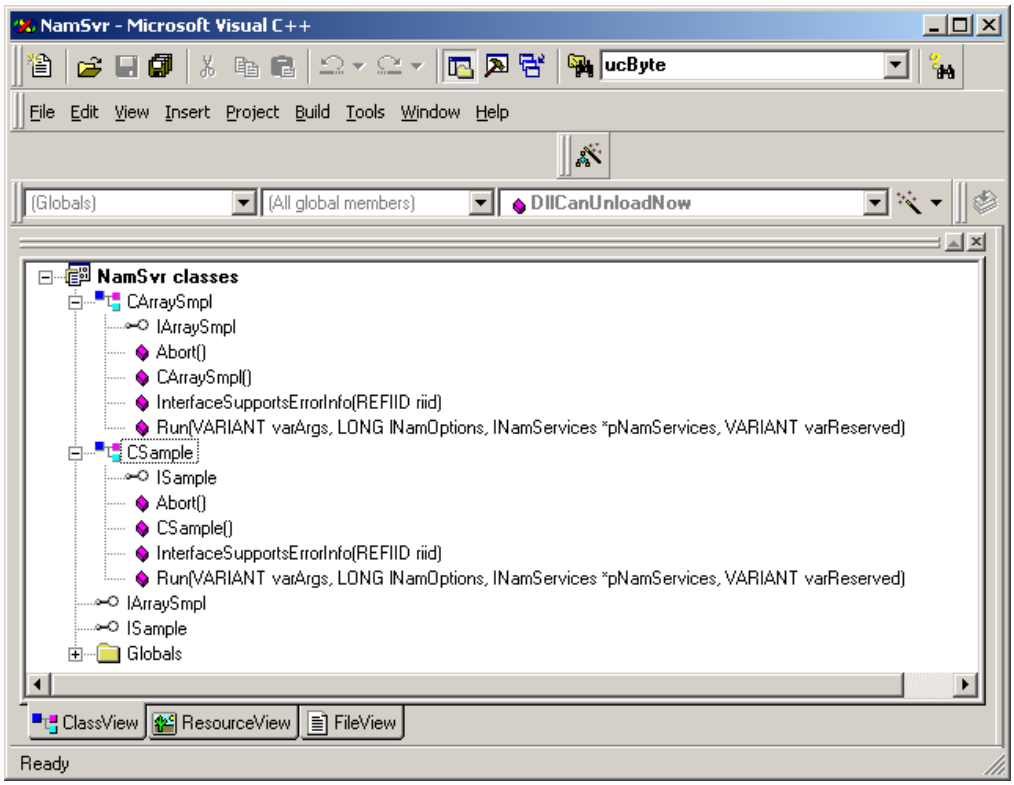


- In the following window, you need to check **INam** interface and then click on **OK**.



- Repeat the previous steps for implementing the same **INam** interface for the **CSample** Class.

- Once completed, you should get the following display under **ClassView**:



- As shown on the above capture both classes **CArraySmpl** and **CSample** received two additional methods**: Abort()** and **Run()** after implementing the **INam** interface.

1.4 Developing the project

- You reached the point when all the procedures related to the automatic code generation were accomplished. From now on you need to adjust and develop the code manually.

- In the **Resource.h** file add the additional directive as shown in the code **in bold** below:

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by NamSvr.rc
//
#define IDS_PROJNAME                    100
#define IDR_SAMPLE                      101
#define IDR_ARRAYSMPL                   102
#define IDS_E_UNEXPECTEDPARAM           0x201 //added directive

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        201
#define _APS_NEXT_COMMAND_VALUE         32768
#define _APS_NEXT_CONTROL_VALUE         201
#define _APS_NEXT_SYMED_VALUE           103
#endif
#endif
```

- Add an extra member function **ChangeData()** to the **CSample** class by adding the following declaration in **Sample.h** file as presented in the code below:

```
// Sample.h : Declaration of the CSample

#ifndef __SAMPLE_H_

#define __SAMPLE_H_


#include "resource.h"       // main symbols

#import "c:\usr\tyx\com\ComNam.dll" raw_interfaces_only, raw_native_types, no_namespace, named_guids


//////////////////////////////////////////////////////////////////////////

// CSample

class ATL_NO_VTABLE CSample :

        public CComObjectRootEx<CComSingleThreadModel>,

        public CComCoClass<CSample, &CLSID_Sample>,

        public ISupportErrorInfo,

        public ISample,

        public INam

{

public:

        CSample()

        {

        }


DECLARE_REGISTRY_RESOURCEID(IDR_SAMPLE)


DECLARE_PROTECT_FINAL_CONSTRUCT()


BEGIN_COM_MAP(CSample)

        COM_INTERFACE_ENTRY(ISample)

        COM_INTERFACE_ENTRY(ISupportErrorInfo)
```

```
        COM_INTERFACE_ENTRY(INam)
    END_COM_MAP()


    // ISupportsErrorInfo
        STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid);


    // ISample
    public:
    // INam
        STDMETHOD(Run)(VARIANT varArgs, LONG lNamOptions, INamServices *
pNamServices, VARIANT varReserved)
        {
            return E_NOTIMPL;
        }
        STDMETHOD(Abort)()
        {
            return E_NOTIMPL;
        }
    // utility functions
        HRESULT ChangeData(long lVad, INamServices* pNamServices); // added
    };


    #endif //__SAMPLE_H_
```

- In the **Sample.cpp** make the following changes:
  - Add an ID **&IID_INam** for proper error reporting (and add a comma to the end of the previous line).
  - Implement the code for **Run()** method
  - Implement the code for **Abort()** method

- Implement the code for **ChangeData()** method

Those changes are shown below in bold:

```
// Sample.cpp : Implementation of CSample

#include "stdafx.h"

#include "NamSvr.h"

#include "Sample.h"


/////////////////////////////////////////////////////////////////////////////

// CSample


STDMETHODIMP CSample::InterfaceSupportsErrorInfo(REFIID riid)

{

	static const IID* arr[] =

	{

		&IID_ISample,

		&IID_INam

	};

	for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)

	{

		if (InlineIsEqualGUID(*arr[i],riid))

			return S_OK;

	}

	return S_FALSE;

}
```

STDMETHODIMP CSample::Run(VARIANT varArgs, LONG lNamOptions, INamServices * pNamServices, VARIANT varReserved)

```
{
        if (varArgs.vt != (VT_ARRAY | VT_I4))
                return Error(IDS_E_UNEXPECTEDPARAM);

        SAFEARRAY* pSafeArray = varArgs.parray;
        long lArgsIdxMax; //get the maxim index for this safearray
        HRESULT hr = ::SafeArrayGetUBound(pSafeArray, 1, &lArgsIdxMax);
        if (FAILED(hr)) return hr;
        if (lArgsIdxMax != 4)
                return Error(IDS_E_UNEXPECTEDPARAM);

        // arguments
        long lVad;
        long lArgsIdx[1];
        for (int nArgsIdx = 0; nArgsIdx <= lArgsIdxMax; nArgsIdx++)
        {
                lArgsIdx[0] = nArgsIdx;
                hr = ::SafeArrayGetElement(pSafeArray, lArgsIdx, &lVad);
                if (FAILED(hr)) return hr;

                hr = ChangeData(lVad, pNamServices);
                if (FAILED(hr)) return hr;
        }

        return S_OK;
}

STDMETHODIMP CSample::Abort()
```

```
{
	return E_NOTIMPL;
}

HRESULT CSample::ChangeData(long lVad, INamServices* pNamServices)
{
	CComVariant varVal;
	HRESULT hr = pNamServices->GetData(lVad, &varVal);
	if (FAILED(hr)) return hr;

	long lTypeInfo;
	hr = pNamServices->GetType(lVad, &lTypeInfo);
	if (FAILED(hr)) return hr;

	long lType = lTypeInfo & 0xf;

	if (varVal.vt == VT_BOOL)
		// BOOL
		varVal.boolVal = !varVal.boolVal;
	else if (varVal.vt == VT_I4)
		// INTEGER
	varVal.lVal++;
	else if (varVal.vt == VT_R8)
		// DECIMAL(REAL)
		varVal.dblVal += 1.234;
	else if (varVal.vt == VT_BSTR && lType == NAM_TYPE_TEXT)
	{
		// TEXT (STRING OF CHAR)
```

```
		hr = varVal.Clear();
		if (FAILED(hr)) return hr;

		varVal = CComBSTR("Modified Text");
}
else if (varVal.vt == (VT_ARRAY | VT_UI2))
{
		// DIGITAL

		// check the size for digital
		long lDigIdxMax; //get the maxim index for this safearray
		HRESULT hr = ::SafeArrayGetUBound(varVal.parray, 1, &lDigIdxMax);
		if (FAILED(hr)) return hr;
		if (lDigIdxMax != 0)
			return Error(IDS_E_UNEXPECTEDPARAM);

		// check the excess for digital
		if (((lTypeInfo & NAM_EXCESSMASK) >> 4) != 0)
			return Error(IDS_E_UNEXPECTEDPARAM);

		// modify the only digital word
		long lWordIndex[1];
		lWordIndex[0] = 0;
		unsigned short* pWord;
		hr = ::SafeArrayPtrOfIndex(varVal.parray, lWordIndex, (void**)&pWord);
		if (FAILED(hr)) return hr;

		*pWord ^= 0xffff;
```

```
        }

        else

                return Error(IDS_E_UNEXPECTEDPARAM);


        return pNamServices->PutData(lVad, varVal);

}
```

- In the **Sample.h** file remove the body for **Run()** and **Abort()** methods in order to avoid compiling errors. Leave clean methods declarations. Change the following lines:

```
// ISample

public:

// INam

    STDMETHOD(Run)(VARIANT varArgs, LONG lNamOptions, INamServices * pNamServices, VARIANT varReserved)

    {

            return E_NOTIMPL;

    }

    STDMETHOD(Abort)()

    {

            return E_NOTIMPL;

    }
```

into:

```
// ISample

public:
```

```
// INam

STDMETHOD(Run)( VARIANT varArgs, LONG lNamOptions, INamServices * pNamServices, VARIANT varReserved);

STDMETHOD(Abort)();
```

- In **ArraySmpl.cpp** file do the following changes:
  - add an ID **&IID_INam**
  - implement the code for **Run()** method
  - implement the code for **Abort()** method

```
// ArraySmpl.cpp : Implementation of CArraySmpl

#include "stdafx.h"

#include "NamSvr.h"

#include "ArraySmpl.h"


/////////////////////////////////////////////////////////////////////////////

// CArraySmpl


STDMETHODIMP CArraySmpl::InterfaceSupportsErrorInfo(REFIID riid)

{

    static const IID* arr[] =

    {

        &IID_IArraySmpl,

        &IID_INam

    };

    for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)

    {

        if (InlineIsEqualGUID(*arr[i],riid))

            return S_OK;
```

```
    }

    return S_FALSE;

}
```

STDMETHODIMP CArraySmpl::Run(VARIANT varArgs, LONG lNamOptions, INamServices * pNamServices, VARIANT varReserved)

```
{

    if (varArgs.vt != (VT_ARRAY | VT_I4))

            return Error(IDS_E_UNEXPECTEDPARAM);



    SAFEARRAY* pSafeArray = varArgs.parray;

    long lArgsIdxMax; //get the maxim index for this safearray

    HRESULT hr = ::SafeArrayGetUBound(pSafeArray, 1, &lArgsIdxMax);

    if (FAILED(hr)) return hr;

    if (lArgsIdxMax != 1) // 1 is for two arguments(arg number 0 and arg number 1)

            return Error(IDS_E_UNEXPECTEDPARAM);



    // get the vad for the integer array(which is the vad of its first element)

    long lArgsIdx[] = {0};

    long vadInts;

    hr = ::SafeArrayGetElement(pSafeArray, lArgsIdx, &vadInts);

    if (FAILED(hr)) return hr;



    // get the vad for the array size

    lArgsIdx[0] = 1;

    long vadArraySize;

    hr = ::SafeArrayGetElement(pSafeArray, lArgsIdx, &vadArraySize);

    if (FAILED(hr)) return hr;
```

```
// get the size of the size array
CComVariant varArraySize;
hr = pNamServices->GetData(vadArraySize, &varArraySize);
if (FAILED(hr)) return hr;
if (varArraySize.vt != VT_I4)
        return Error(IDS_E_UNEXPECTEDPARAM);

// increment all elements of the integer array
for (int i = 0; i < varArraySize.lVal; i++)
{
        // get the array element whose vad is vadInts
        CComVariant varInt;
        hr = pNamServices->GetData(vadInts, &varInt);
        if (FAILED(hr))
                return hr;
        if (varInt.vt != VT_I4)
                return Error(IDS_E_UNEXPECTEDPARAM);

        // increment its value
        varInt.lVal++;

        // put the incremented value back in the array
        hr = pNamServices->PutData(vadInts, varInt);
        if (FAILED(hr))
                return hr;

        // Steps for passing to the next element of the array:
        // Step one: get the type of the current element
```

```
            long nTypeInfo;
            hr = pNamServices->GetType(vadInts, &nTypeInfo);
            if (FAILED(hr))
                  return(E_FAIL);

            // Step two: calculate the size of the current element
            long lElementSize = (nTypeInfo & NAM_SIZEMASK) >> 8;

            // Step three: calculate the vad of the next element
            vadInts += lElementSize;
      }

      return S_OK;
}

STDMETHODIMP CArraySmpl::Abort()
{
      return E_NOTIMPL;
}
```

- In the **ArraySmpl.h** file, remove the body for **Run()**, **Abort()** methods in order to avoid compiling errors. Leave clean methods declarations. Change the following lines:

```
// IArraySmpl
public:
// INam
      STDMETHOD(Run)(VARIANT varArgs, LONG lNamOptions, INamServices * pNamServices, VARIANT
varReserved)
```

```
{
        return E_NOTIMPL;
}
STDMETHOD(Abort)()
{
        return E_NOTIMPL;
}
```

into:

```
// IArraySmpl
public:
// INam
    STDMETHOD(Run)(VARIANT varArgs, LONG lNamOptions, INamServices * pNamServices, VARIANT varReserved);
    STDMETHOD(Abort)();
```

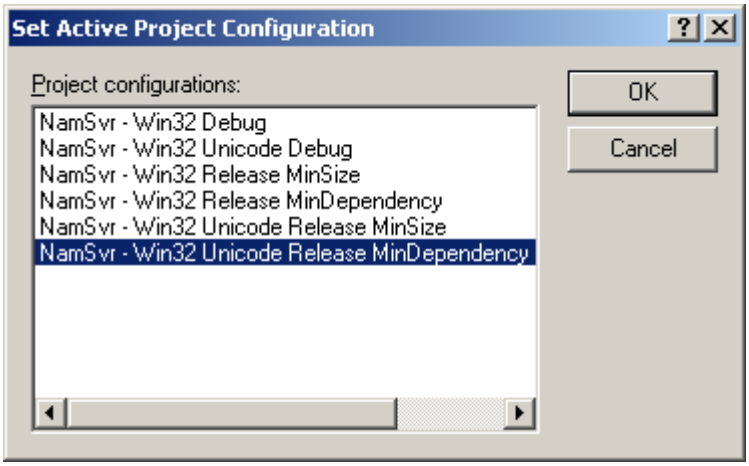
- Now you should be able to build the project without any errors. Press **F7** button or use the Build icon from the Toolbar.

1.4.1 Setting the MSVC++ project

- In addition to the previous changes adjust the following project settings:
  - From the menu bar, under **Build | Set Active Configuration…** select:

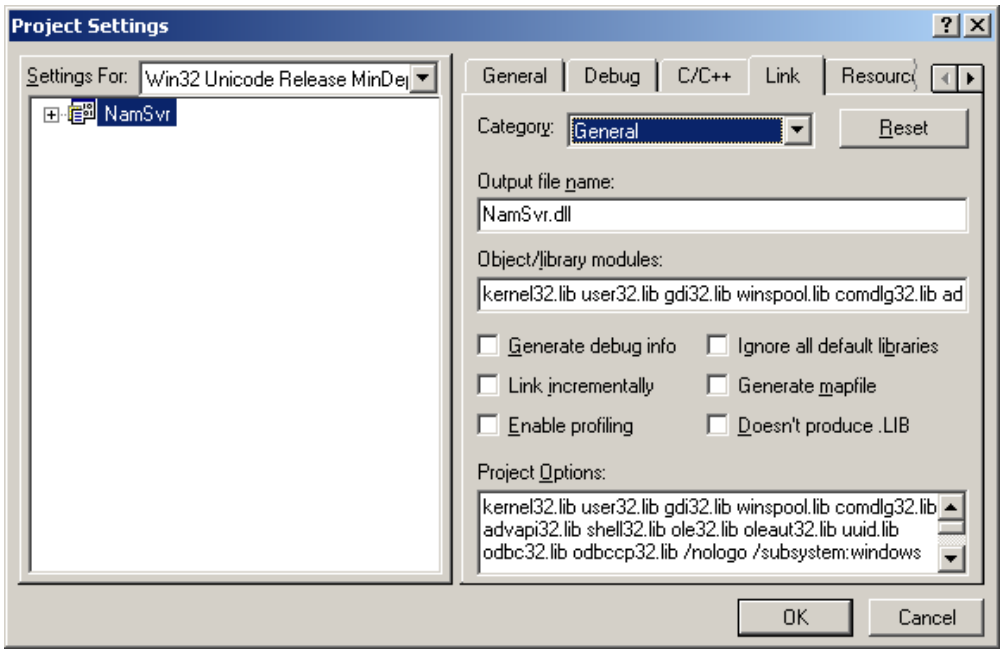
    NamSvr – Win32 Unicode Release MiniDependency

    as shown below:



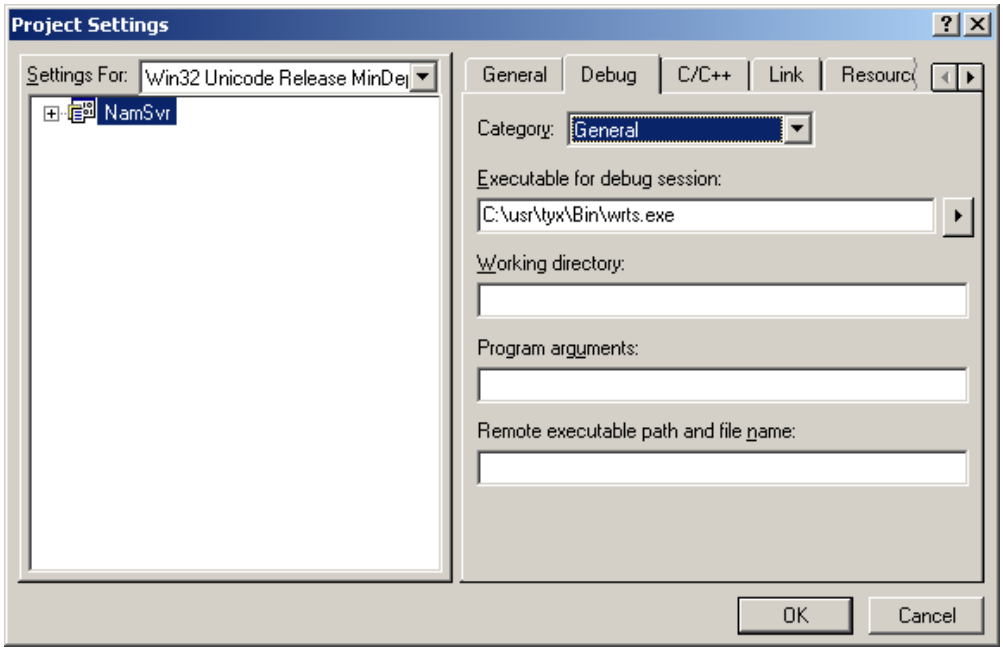
Press **OK.**

- From the menu bar, under **Project | Settings | Link | Category: General** set the **Output file name:** to **NamSvr.dll** without any path:



This will allow the .dll file to be generated in the main project directory. Eventually, you will be able to make use of this COM-NAM dll from any folder because the path information will be taken care of by the registration mechanism. You will however have to register the COM component from the location you want to use it from. This can be done easily from a command line after you moved to the directory containing the dll. The command line to register the dll is the following: **regsvr32 <COM component name>.dll**. You will get a confirmation that it registered if all goes well.

- From the menu bar, under **Project | Settings | Debug | Category: General** set the following file as an **Executable for debug session: C:\usr\tyx\bin\Wrts.exe** (include the proper path for Wrts.exe).



This is only necessary if you want to debug the dll. If you omit this setting and you want to debug the dll, the MSVC studio will ask for it when you start your debug session.

2 Setting TYX PAWS Studio

- At this point, you will need to have an Atlas program that makes use of that COM-NAM object and to setup the Wrts in order to properly use COM-NAM technique.

2.1 Sample code for IEEE716.89 and IEEE716.95 ATLAS:

```
 000100 BEGIN, ATLAS PROGRAM 'COM_NAM_SAMPLE'                          $
C                                                                      $
     10 INCLUDE, NON-ATLAS MODULE 'NAMSVR.SAMPLE'                      $
     15 INCLUDE, NON-ATLAS MODULE 'NAMSVR.ARRAYSMPL'                   $
C                                                                      $
     20 DECLARE, VARIABLE, 'B' IS BOOLEAN INITIAL = FALSE              $
     30 DECLARE, VARIABLE, 'I' IS INTEGER INITIAL = 0                  $
     40 DECLARE, VARIABLE, 'R' IS DECIMAL INITIAL = 1.234              $
     50 DECLARE, VARIABLE,
            'T' IS STRING (80) OF CHAR INITIAL = C'Input text'         $
     60 DECLARE, VARIABLE, 'D' IS STRING (16) OF BIT INITIAL = X'01FE' $
     70 DECLARE, VARIABLE, 'IDX' IS INTEGER                            $
     75 DECLARE, CONSTANT, 'ARRAY_SIZE' IS 5                           $
     80 DECLARE, VARIABLE, 'INT_ARRAY' IS ARRAY(1 THRU 'ARRAY_SIZE')
              OF INTEGER INITIAL = 2, 4, 6, 8, 10                      $
C                                                                      $
C**********************************************************************$
C                                                                      $
E100000 OUTPUT, TO 'DISPLAY', C'INPUT NAM:\LF\',
           C'\HT\Boolean', 'B', C'\LF\',
```

```
            C'\HT\Integer', 'I', C'\LF\',
            C'\HT\Decimal ', 'R':5:3, C'\LF\',
            C'\HT\Text   ', 'T', C'\LF\',
            C'\HT\Digital', 'D', C'\LF\',
            C'\HT\Integer Array'$
     10 FOR, 'IDX' = 1 THRU 'ARRAY_SIZE', THEN                          $
     20     OUTPUT, TO 'DISPLAY', 'INT_ARRAY'('IDX')$
     30 END, FOR                                                        $
     40 OUTPUT, TO 'DISPLAY', C'\LF\'                                   $
C                                                                       $
     50 OUTPUT, TO 'DISPLAY',
            C'\ESC\[31;1m<---Perform COM Non-Atlas-Modules--->\ESC\[m' $
     60 PERFORM, 'NAMSVR.SAMPLE' ('B', 'I', 'R', 'T', 'D')              $
     65 PERFORM, 'NAMSVR.ARRAYSMPL' ('INT_ARRAY', 'ARRAY_SIZE')         $
C                                                                       $
     70 OUTPUT, C'OUTPUT NAM:\LF\',
            C'\HT\Boolean', 'B', C'\LF\',
            C'\HT\Integer', 'I', C'\LF\',
            C'\HT\Decimal ', 'R':5:3, C'\LF\',
            C'\HT\Text   ', 'T', C'\LF\',
            C'\HT\Digital', 'D', C'\LF\',
            C'\HT\Integer Array'$
     80 FOR, 'IDX' = 1 THRU 5, THEN                                     $
     90     OUTPUT, TO 'DISPLAY', 'INT_ARRAY'('IDX')$
 100100 END, FOR                                                        $
     10 OUTPUT, TO 'DISPLAY', C'\LF\'                                   $
C                                                                       $
     20 TERMINATE, ATLAS PROGRAM 'COM_NAM_SAMPLE'                       $
```

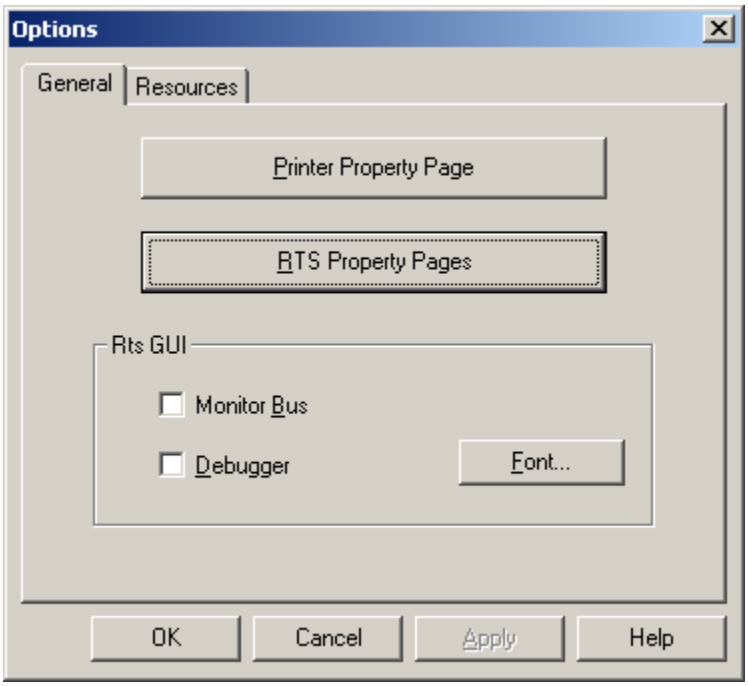
NOTE:

Regardless what you want the Atlas to accomplish you need to include **COM-NAM** (Common Object Module Non-ATLAS Module) using **INCLUDE** statement in the Atlas preamble. The name of the module needs to be defined as a **ProgID** so that the WRTS can identify it. The next step would be to write a **PERFORM** statement. This statement will call the COM-NAM and pass the arguments. At that time RTS identifies the module name as a **ProgID** and it instantiates the COM object through **INam** interface.

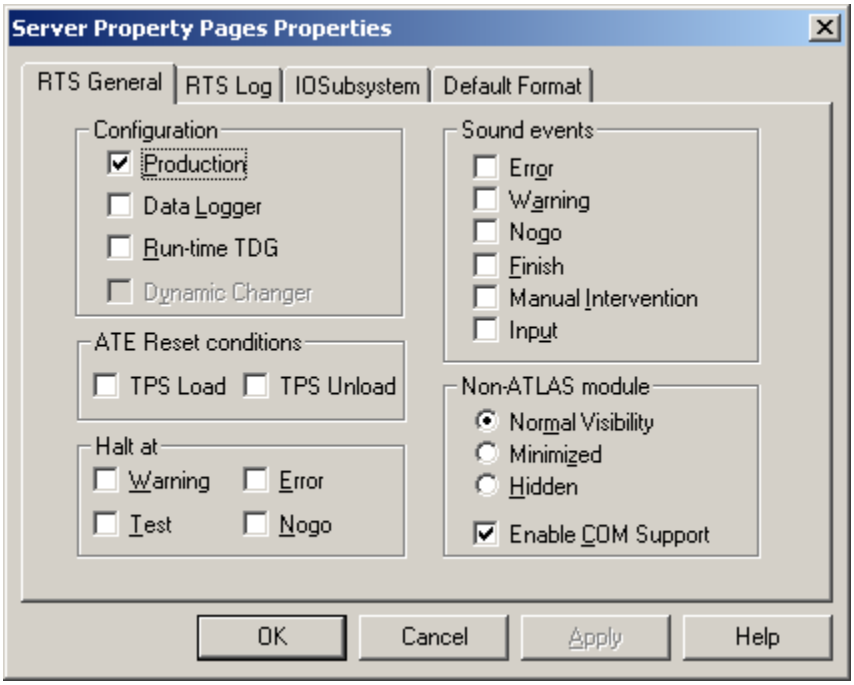
In the example above, the Atlas code includes two COM-NAM **ProgID**: **NAMSVR.SAMPL** and **NAMSVR.ARRAYSMPL**. The first one takes all the parameter values from the Atlas (excluding the array type), processes them, and sends the modified values back to Atlas. The second one handles the parameters of the array type.

2.2 Wrts Settings:

- Prior to building the Atlas project you need to properly set the Wrts..
- Go into **Control/Options…**



- Now Click on **RTS Property Pages** and select the **Enable COM Support** checkbox.



NOTE:

Checking this box causes that Wrts identifies the name of each non-ATLAS module (NAM) as a **ProgID** and then instantiates the COM object whose **ProgID** is the name of the NAM in the ATLAS code.

- Now you can build TYX PAWS project, lunch Wrts and run the project. The COM .dll will be invoked regardless of its location after it has been registered, either by the building process of the COM-NAM dll, or manually from a command line using **regsvr32 <COM-NAM name>.dll**.

2.3 Having trouble?

2.3.1 WRTS warning:

If you get a warning in the WRTS ouput that looks like this:

WRN: Failed to start Non-ATLAS Module: NAMSVR.SAMPLE "C:\DOCUME~1\clgrard\LOCALS~1\Temp\ComNam" 6948 6952 6956 6964 7008

2.3.1.1 Solution 1:

This message can be symptomatic of an improperly registered dll.

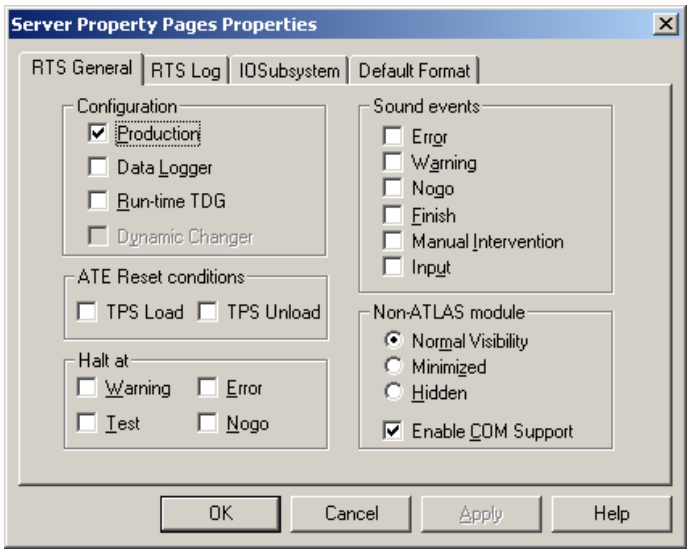
In order to correct the problem, you will need to register the dll. Re-registering and getting a positive confirmation message will be sufficient.

After registering the dll from a command line with **regsvr32 <COM-NAM name>.dll**, the confirmation message will look like this:

2.3.1.2 Solution 2:

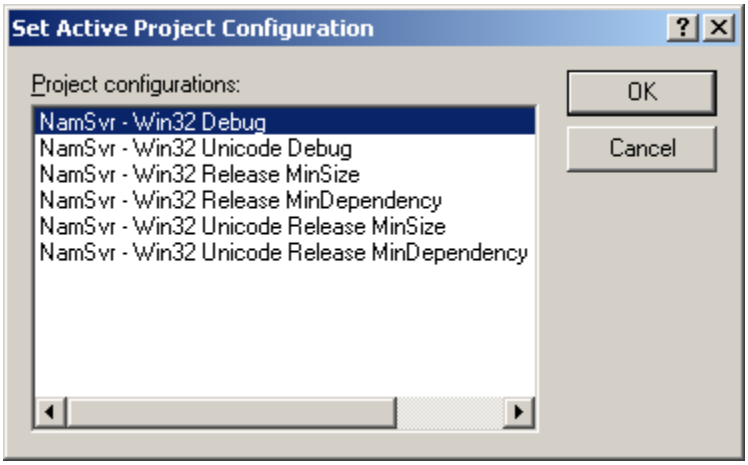
This message can also be symptomatic of an incorrect setting in the WRTS.

To correct this problem, make sure that you select the **Enable COM Support** from the server property page described above and displayed below:



3 How to debug the Com dll?

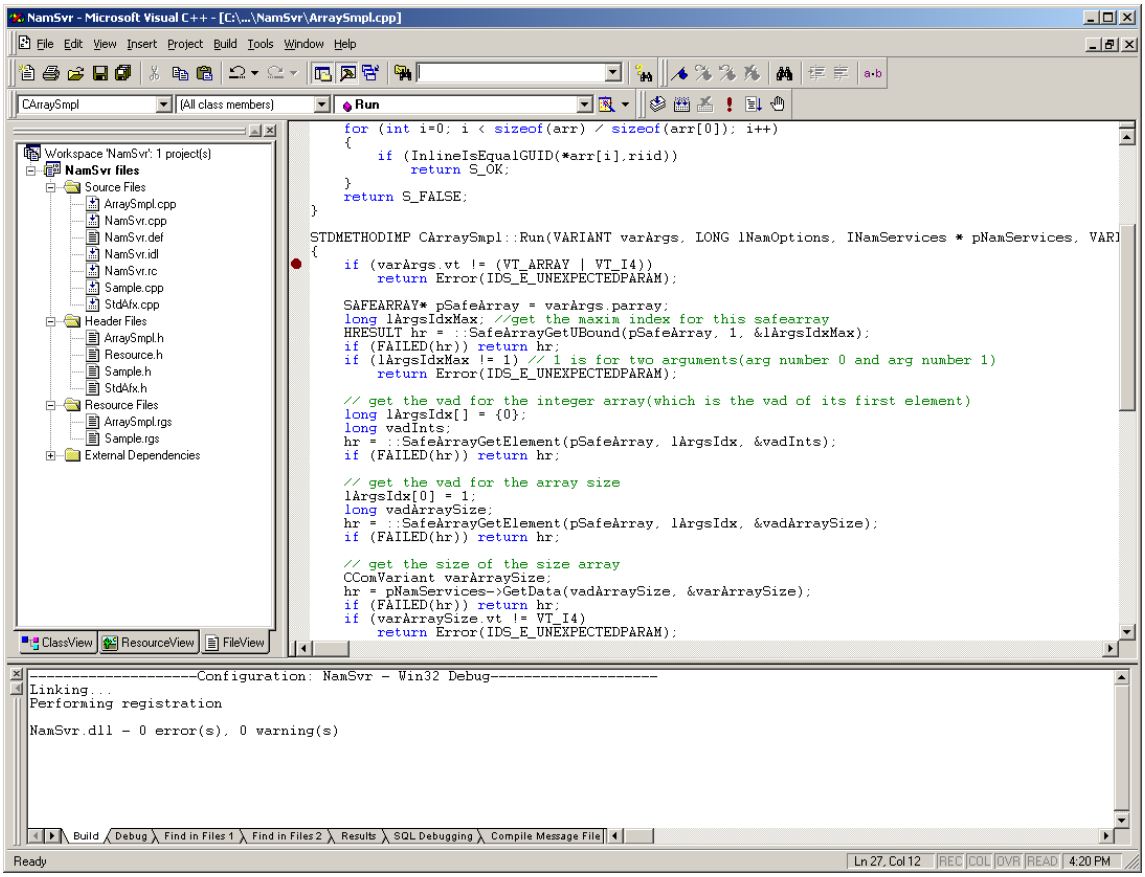
- In the MSVC++, You will need to change the configuration, or make sure that the configuration is set to **NamSvr – Win32Debug**:



If not, select the **NamSvr – Win32Debug** configuration and press **OK**.

- Build the project and this should register the COM-NAM dll. If not, you will need to do it manually.

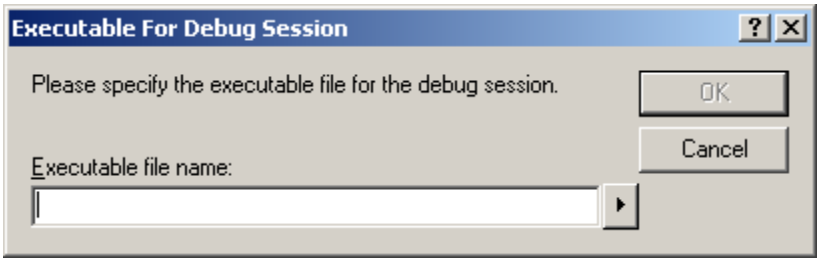
  - In the MSVC++, set some breakpoints in your COM-NAM source code. Your project could look like this:



  - You are now ready to run the COM-NAM in debug mode. You can do this in two ways:

1. From the menu, by going into **Build/Start debug/Go**
2. By pressing the shortcut key which is **F5** and yields to the same result.

- If you did not set the **Executable for debug session:** to **C:\usr\tyx\bin\Wrts.exe**, you will see the following window:

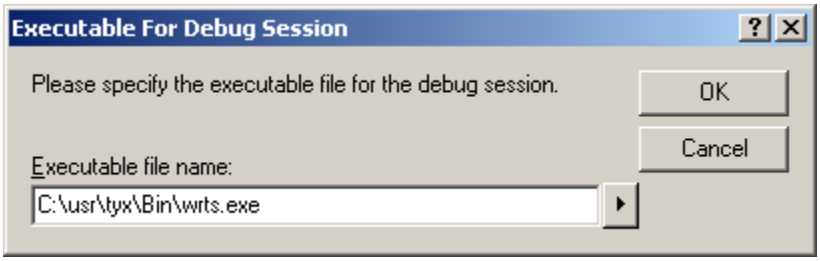


You can enter the **Executable file name:** two ways:

1. Manually, by typing in the path to the WRTS followed by wrts.exe.

2. by clicking on the  button and browing to the location of the wrts.exe on your machine.

The default location for the wrts.exe is under **c:\usr\tyx\bin**:



Now, you should press **OK**.

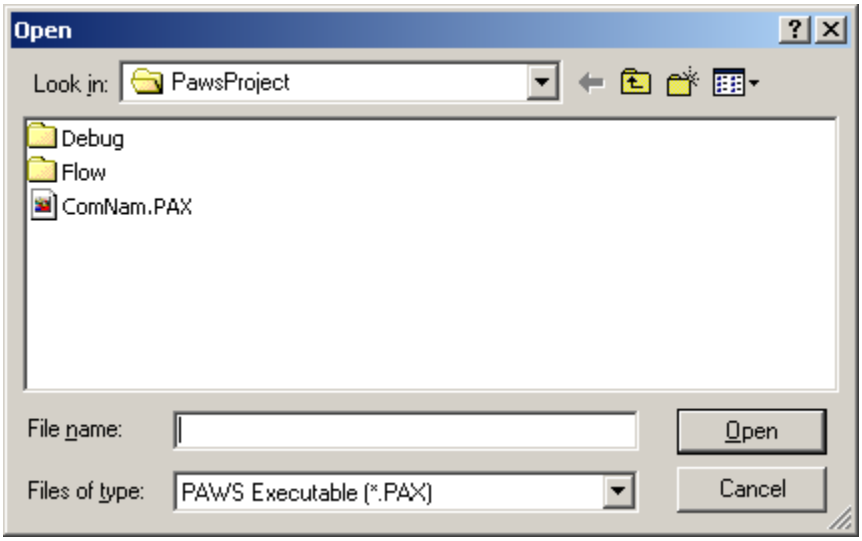
If you did set the **Executable for debug session:** to **C:\usr\tyx\bin\Wrts.exe**, you will not see the widow above and go straight to the following step.

- The first time you will run the COM-NAM in debug mode, you will have this following message:

All that says is that you can't debug the RTS, which is normal and to be expected.

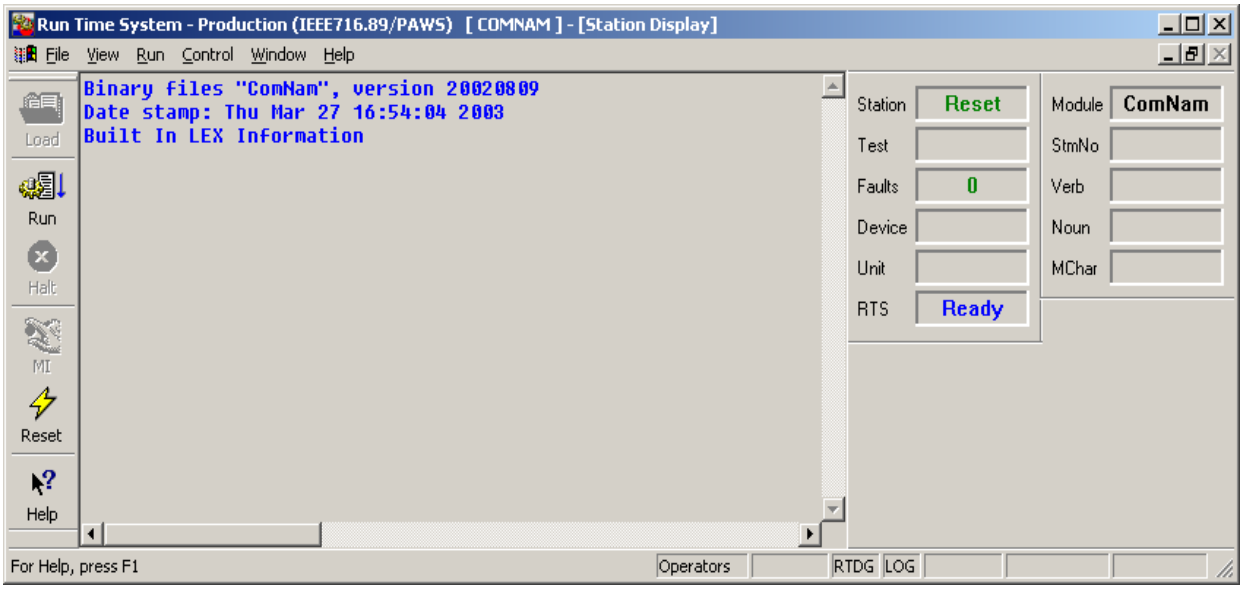
You should check the **Do not prompt in the future.** box and click on **OK**.

- The WRTS will start and ask for you to load a project:

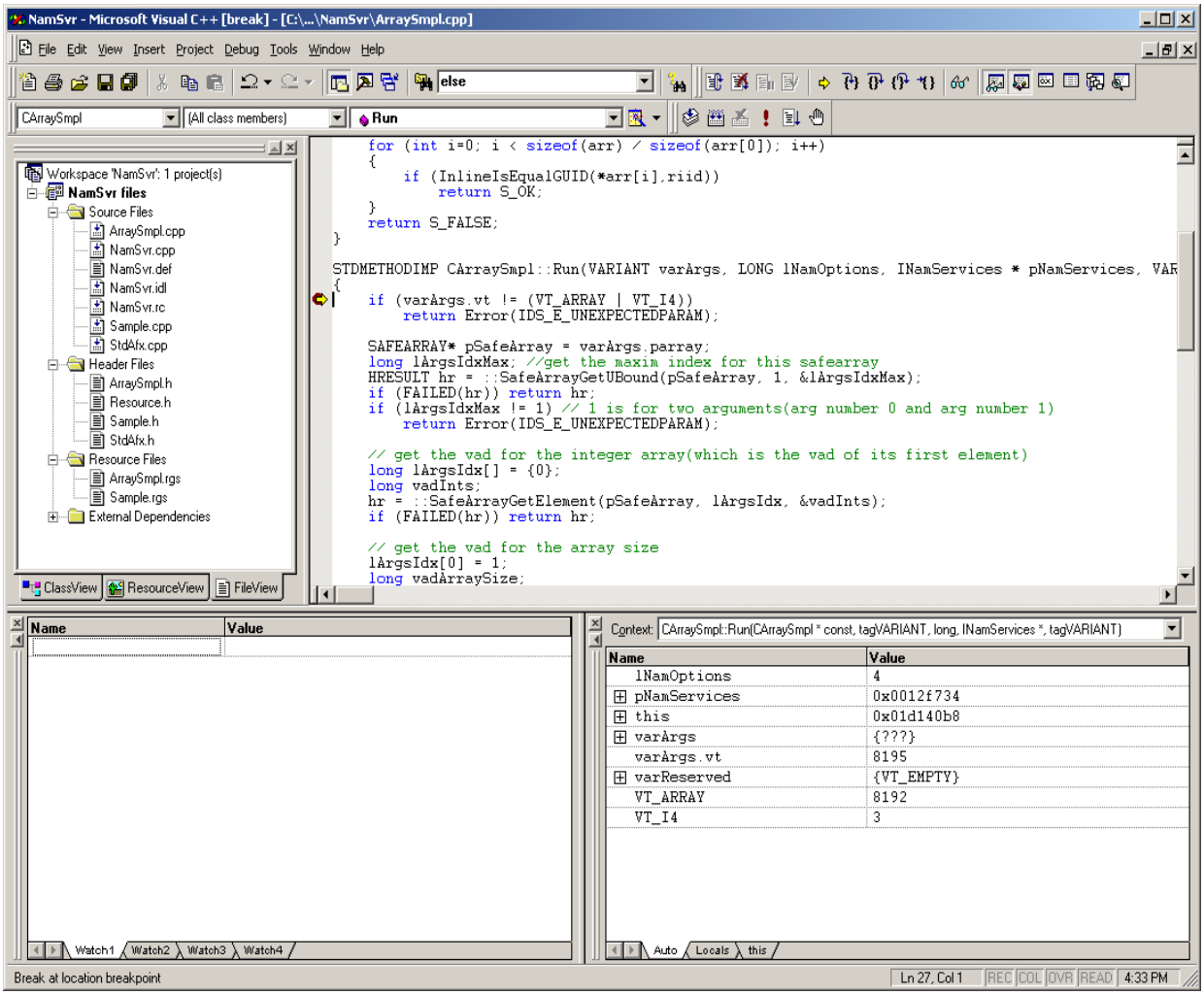


Select the project that uses the COM-NAM that you have the source code for and click on **Open**.

- The WRTS is now ready do be executed



- Press on **Run** and the MSVC will become active and stop at the breakpoint that you set in your code, if one has been set.



- You are now free to proceed from here and debug your COM-NAM source code.

References:

COM-NAM User's Guide