TYX Corporation

Productivity Enhancement Systems

Reference	TYX_0051_13
Revision	1.0
Document	COMNamExe.doc
Date	Apr 05, 2004

How to create a COM-NAM exe?

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

TYX PAWS Studio version used: **1.26.8**

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 an exe file as opposed to dll, which is 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.**

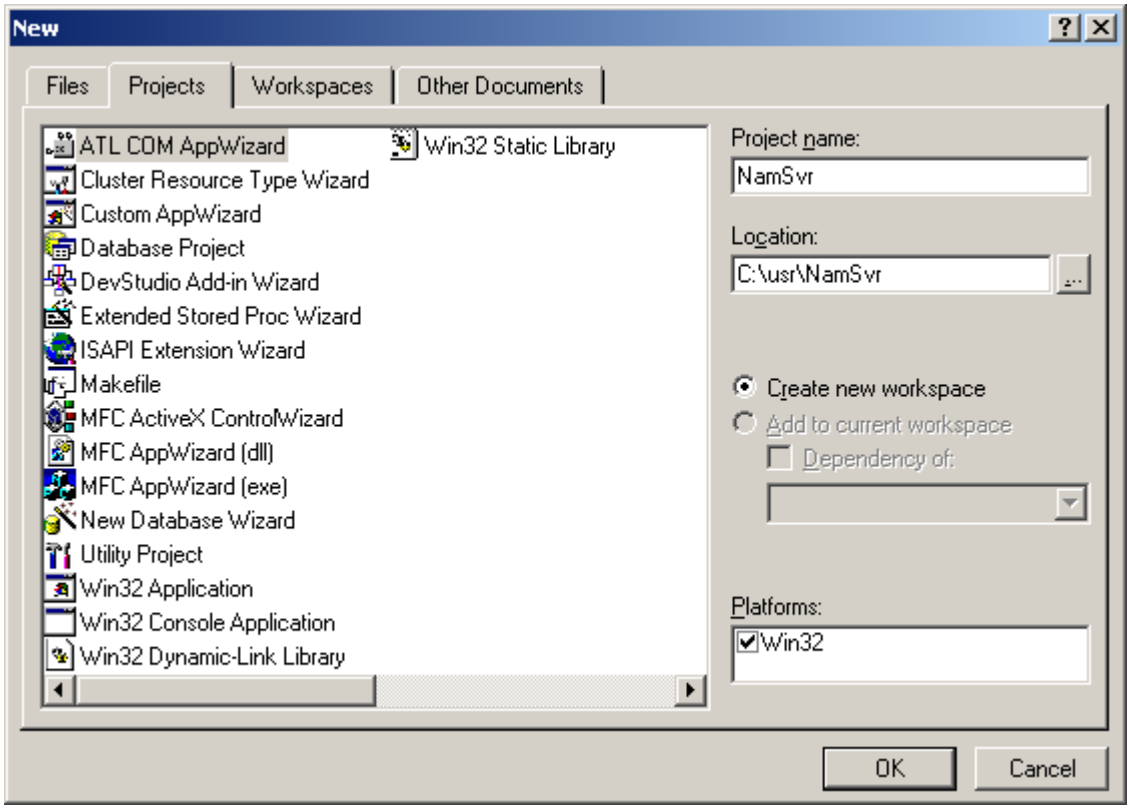
    - **The Exe can be moved in front or behind the Wrts at will.**

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

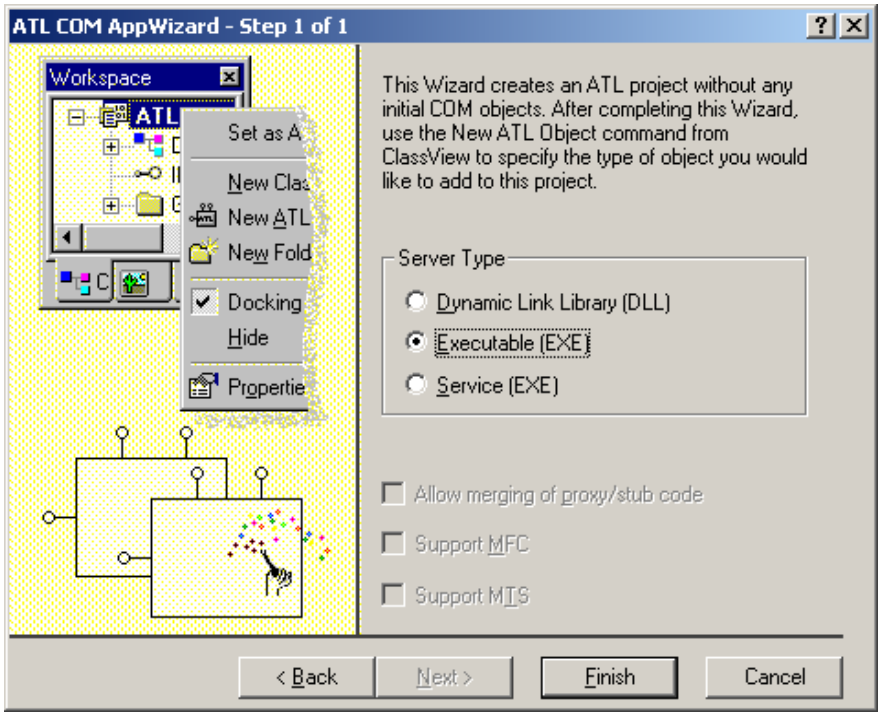
1 Generating an exe with MSVC++ 6.0

1.1 Starting a simple ATL COM Application:

- Open the Visual Studio 6.0.
- From **File | New**…, select an **ATL COM AppWizard**, then select:
  - A project name **NamSvr** as seen below.
  - The location for the project. In this case **C:\usr\NamSvr**.
- Click on **OK**

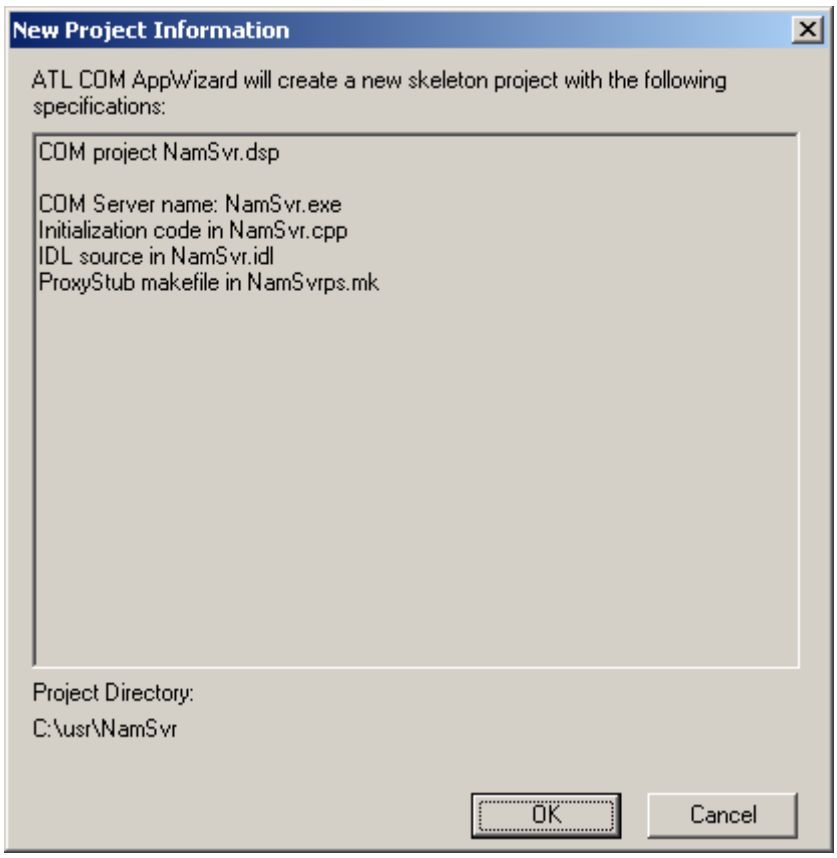


- Select **Executable (EXE)** in the following window.

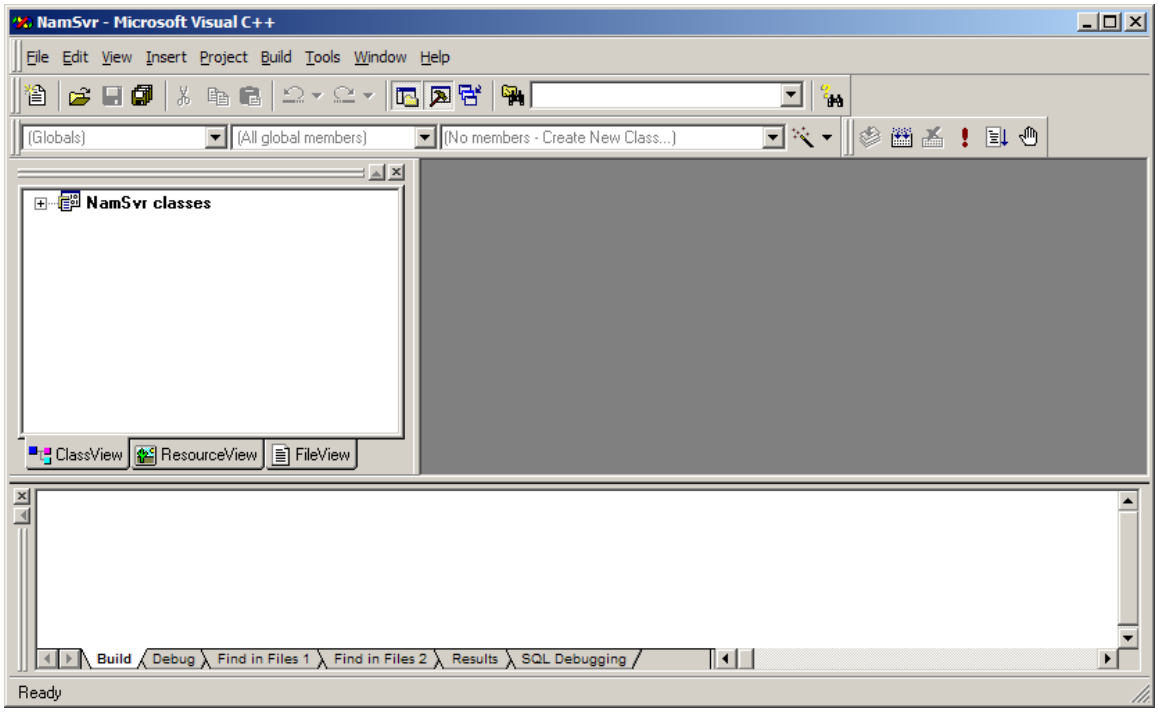


- Click on **Finish**.
- Click **OK** on the following window:

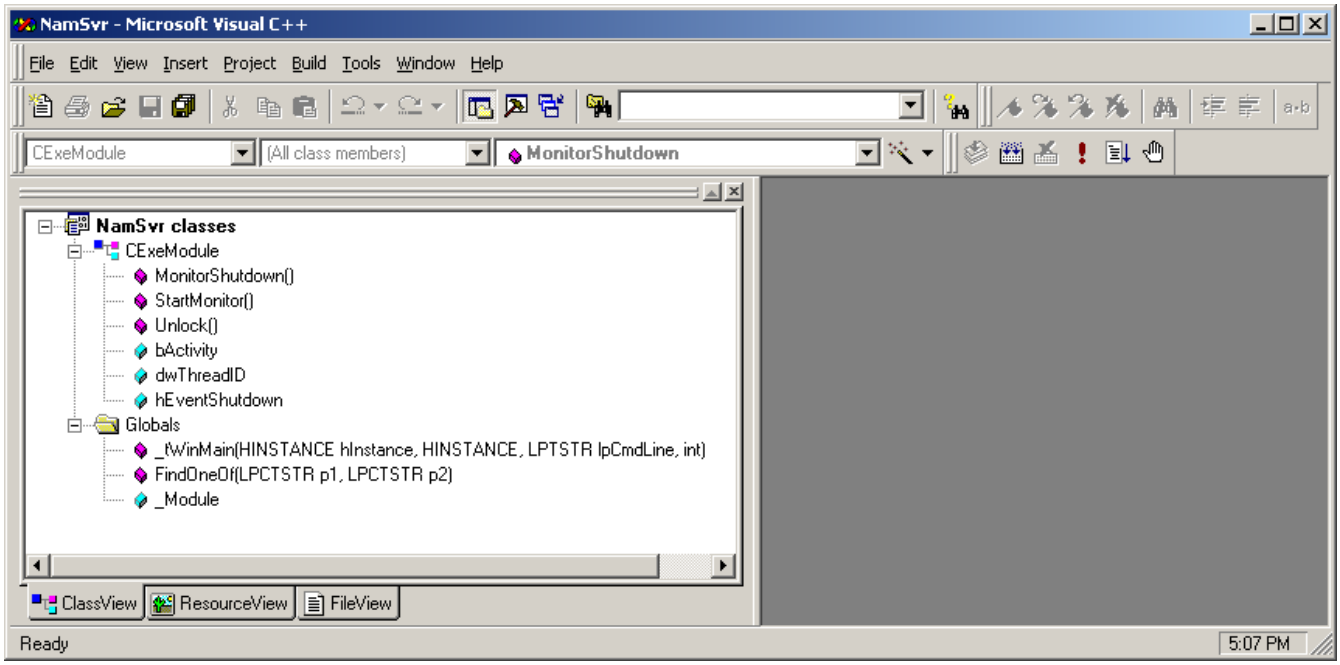
:



- The image below shows how the project should look like.

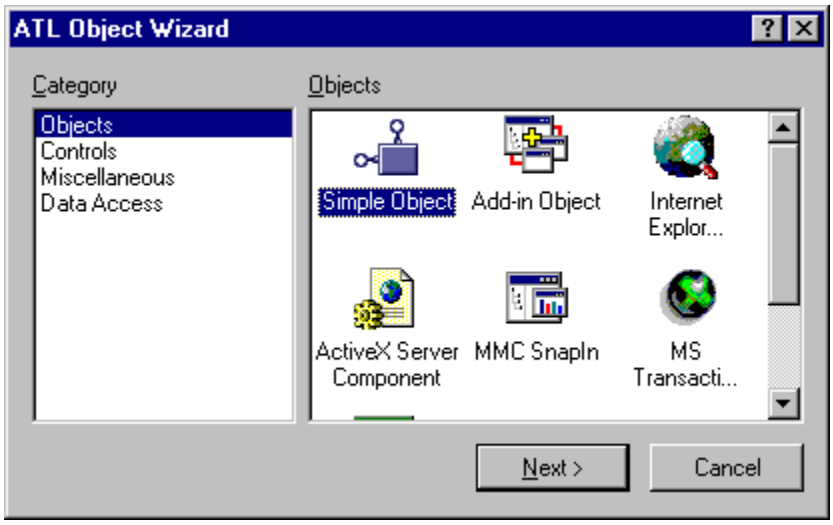


- Select the '+' sign next to the **NamSvr classes** folders to see all the methods generated automatically by the Wizard.



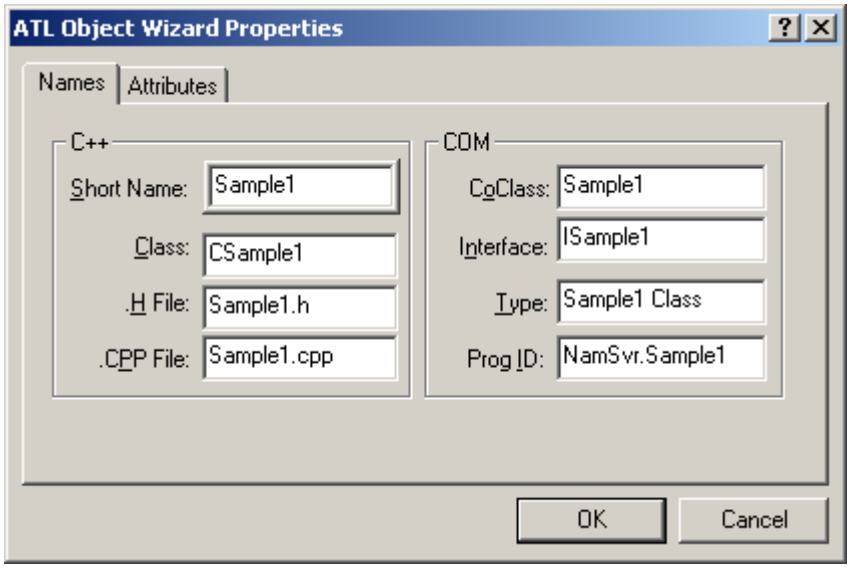
1.2 Adding the ATL object

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



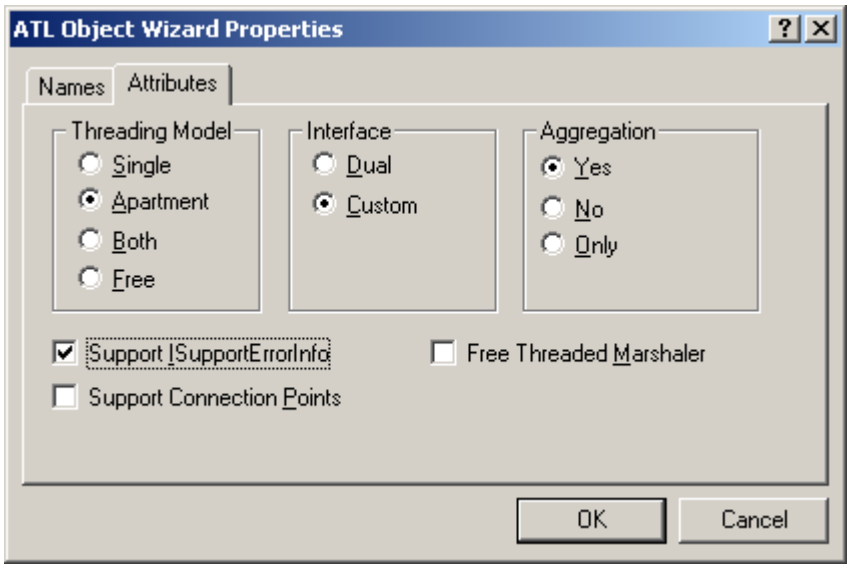
- Select **Simple Object** and click on **Next**.
- You will then see the following window. In the **Short name:** box, enter the name of the ATL object that you wish to create. The other boxes will fill in by themselves. In this case, we chose the name **Sample1**.

  Also, take note of the name in **Prog ID**: In this case **NamSvr.Sample1** (exe name followed by the short name). This will be used as an entry in the Wrts options.



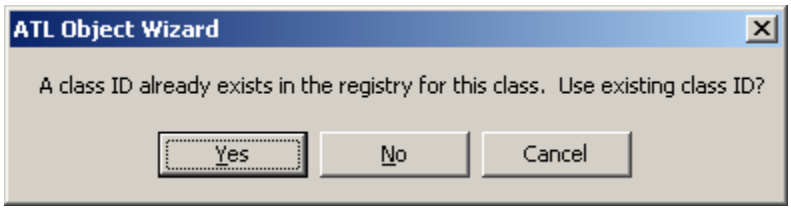
- In the list of attributes, as shown below, all defaults can be acceptable. You may want to add:

  - The **Support IsupportErrorInfo** option.
  - The **Custom** for **Interface** as opposed to **Dual**. The **Custom** option checked causes the COM interface to support only the **IUnknown** capability, which is sufficient for this project.



- Click on **OK**.

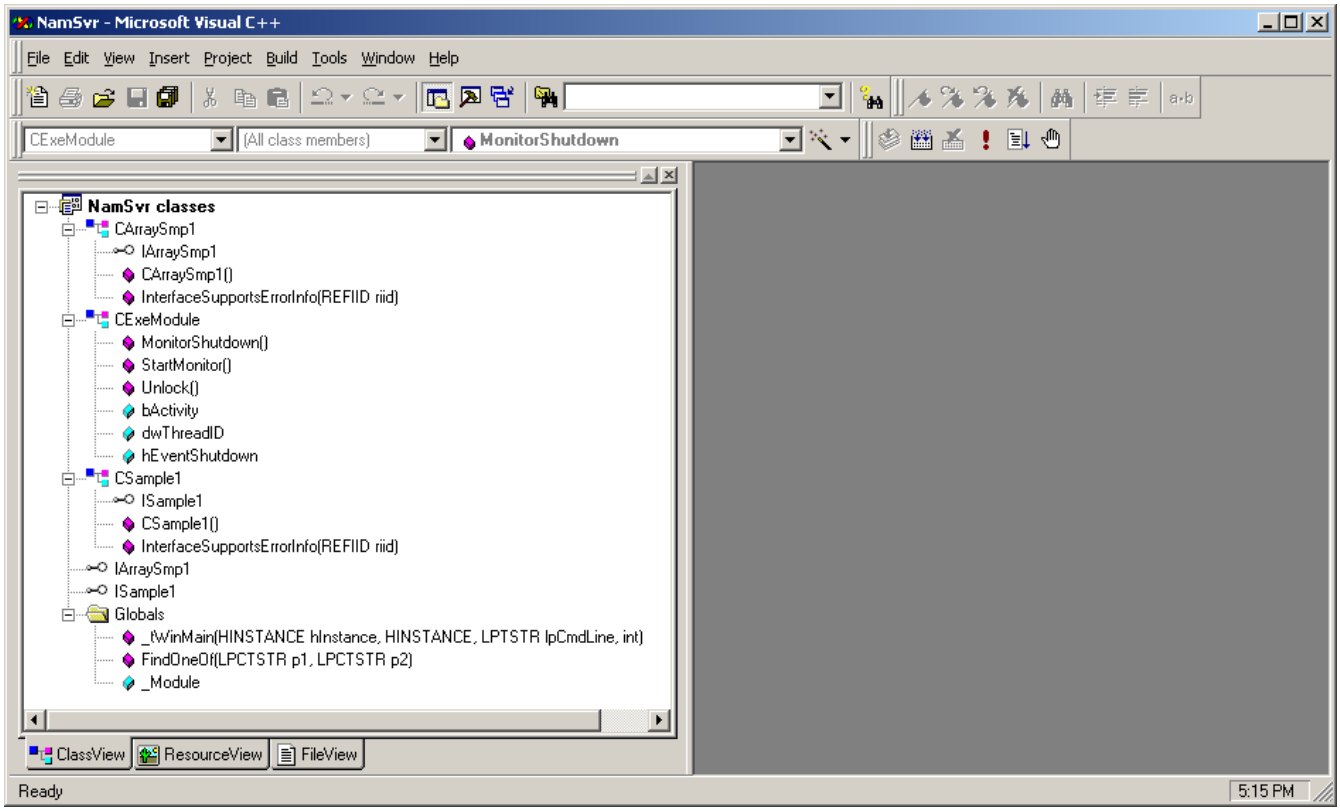
- If you get the following window:



- 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 **ArraySmp1** (Smp *one* and not the letter *L*).

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



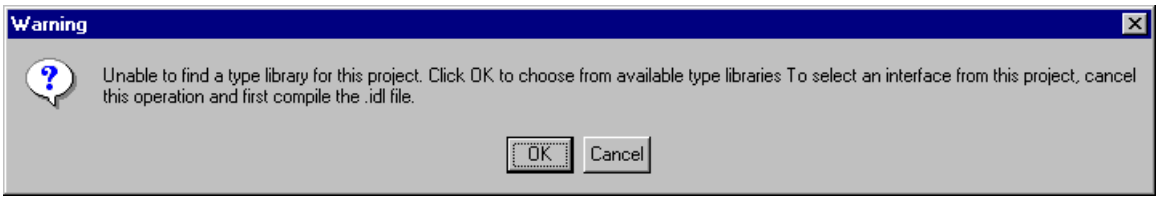
1.3 Implementing COM-NAM interface

- For the WRTS to instantiate the COM-NAM object successfully, 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.

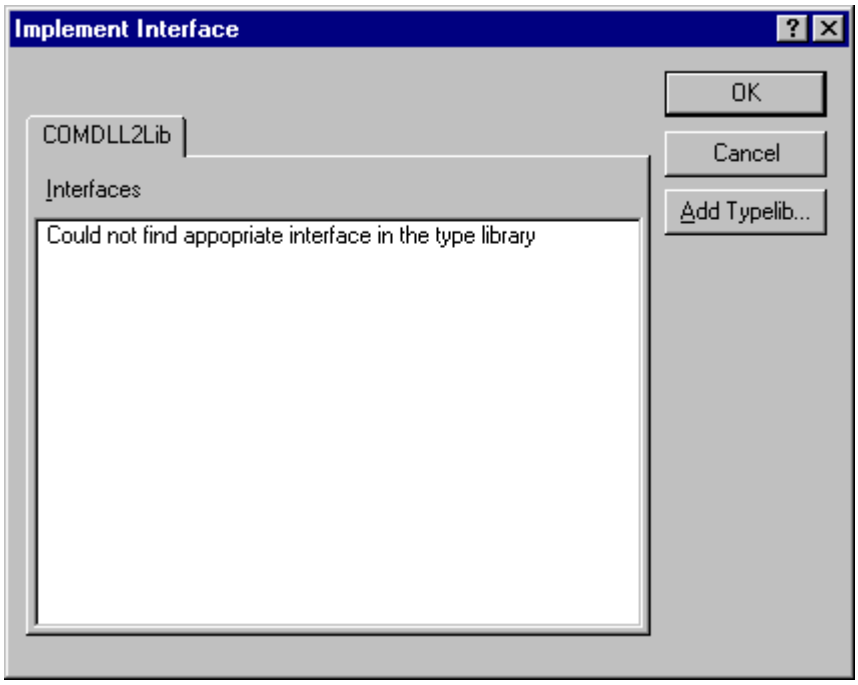
- In the workspace, you should right-click on the **CArraySmp1** Class

- After right clicking on **CArraySmp1**, Select **Implement Interface…** from the drop-down list.

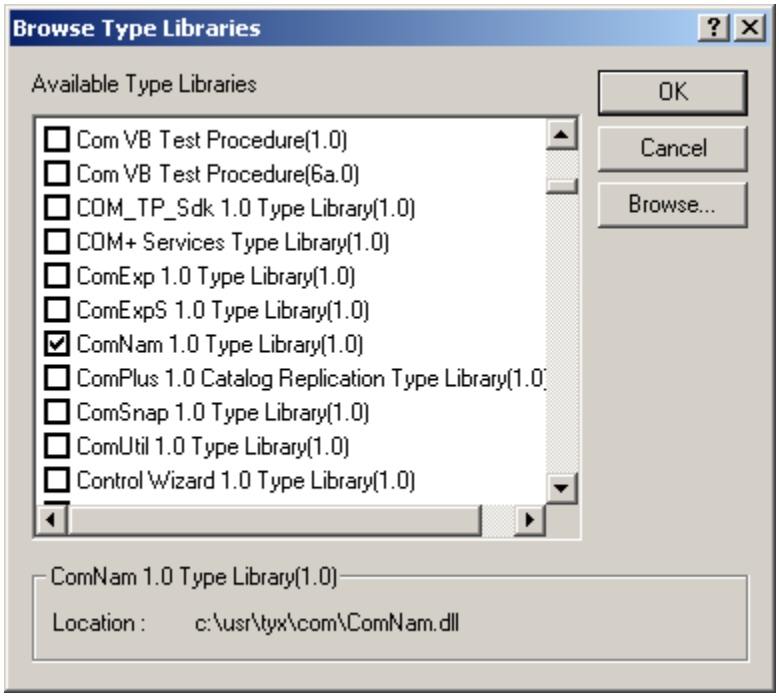
- You will get the following message and then click on **OK**.



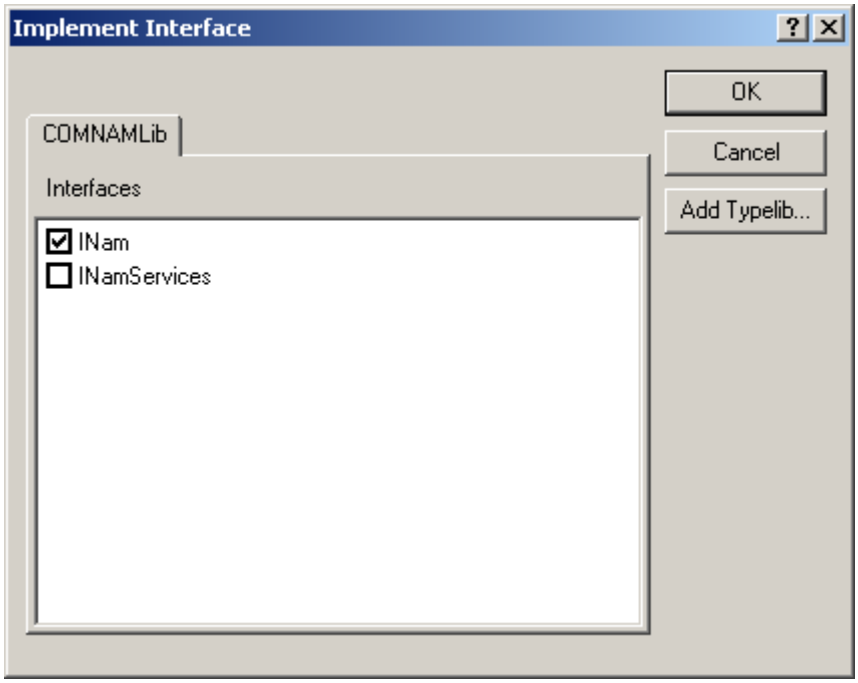
- In the event that you see the following image instead of the previous one, press **AddTypelib…**



- 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.

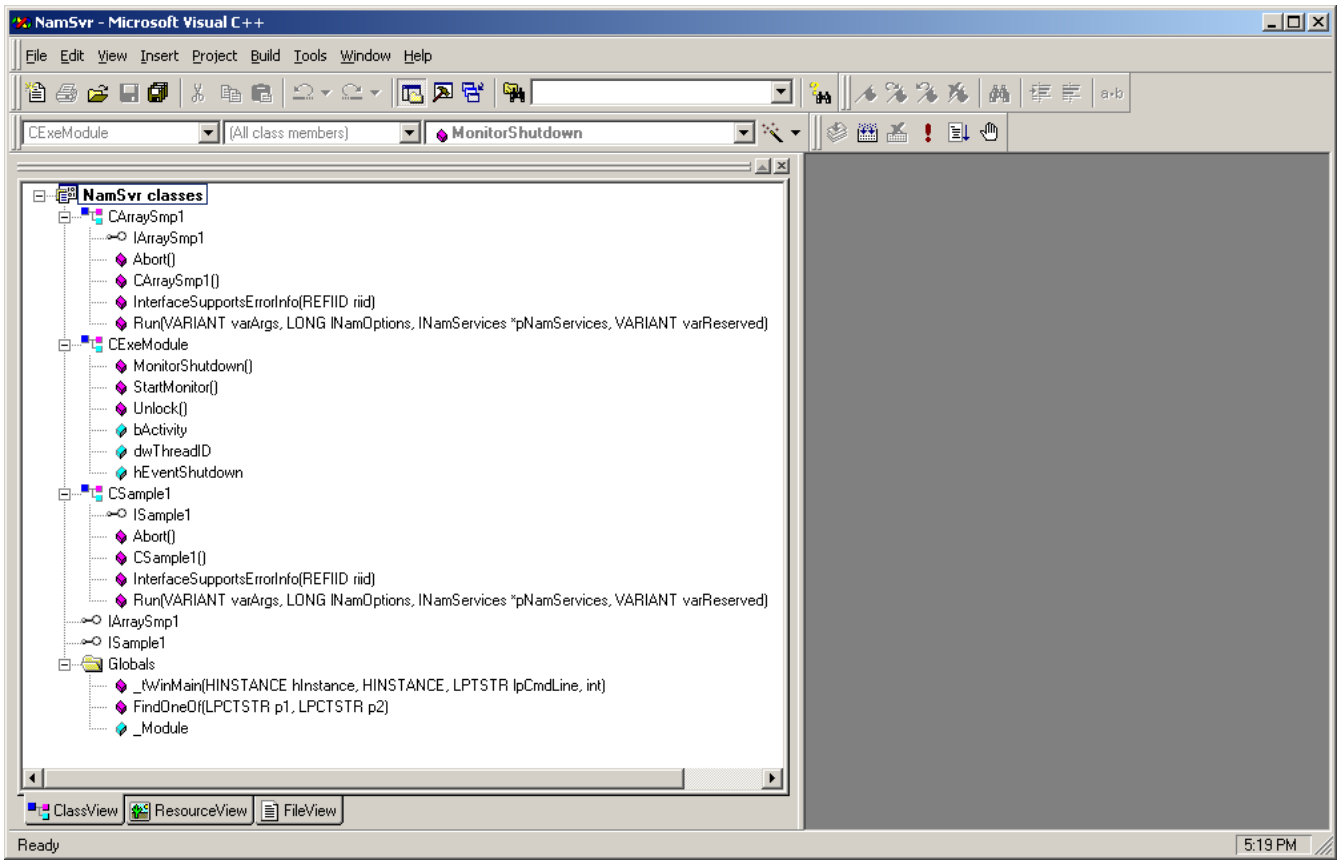


- In the following window, you need to check **INam** interface
- and then click on **OK**.



- Repeat the previous steps for implementing the same **INam** interface for the **CSample1** Class.

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



- As shown on the above capture both classes **CArraySmp1** and **CSample1** 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. The new lines of code are in bold:

- 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_SAMPLE1                     101
#define IDR_ARRAYSMP1                   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 **CSample1** class by adding the following declaration in **Sample1.h** file as presented in the code below:

```
// Sample1.h : Declaration of the CSample1
#ifndef __SAMPLE1_H_
#define __SAMPLE1_H_
```

```
#include "resource.h"       // main symbols

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


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

// CSample1

class ATL_NO_VTABLE CSample1 :

        public CComObjectRootEx<CComSingleThreadModel>,

        public CComCoClass<CSample1, &CLSID_Sample1>,

        public ISupportErrorInfo,

        public ISample1,

        public INam

{

public:

        CSample1()

        {

        }


DECLARE_REGISTRY_RESOURCEID(IDR_SAMPLE1)


DECLARE_PROTECT_FINAL_CONSTRUCT()


BEGIN_COM_MAP(CSample1)

        COM_INTERFACE_ENTRY(ISample1)

        COM_INTERFACE_ENTRY(ISupportErrorInfo)

        COM_INTERFACE_ENTRY(INam)

END_COM_MAP()
```

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


// ISample1
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 //__SAMPLE1_H_
```

- In the **Sample1.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:

```
// Sample1.cpp : Implementation of CSample1

#include "stdafx.h"

#include "NamSvr.h"

#include "Sample1.h"


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

// CSample1


STDMETHODIMP CSample1::InterfaceSupportsErrorInfo(REFIID riid)

{

	static const IID* arr[] =

	{

		&IID_ISample1,

		&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 CSample1::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 CSample1::Abort()
{
        return E_NOTIMPL;
}
```

```
HRESULT CSample1::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 **Sample1.h** file remove the body for **Run()** and **Abort()** methods in order to avoid compiling errors. Leave clean methods declarations. Change the following lines:

```
// ISample1
public:
// INam
    STDMETHOD(Run)(VARIANT varArgs, LONG lNamOptions, INamServices * pNamServices, VARIANT varReserved)
    {
        return E_NOTIMPL;
    }
    STDMETHOD(Abort)()
    {
        return E_NOTIMPL;
    }
```

into:

```
// ISample1
public:
// INam
STDMETHOD(Run)( VARIANT varArgs, LONG lNamOptions, INamServices * pNamServices, VARIANT varReserved);
STDMETHOD(Abort)();
```

- In **ArraySmp1.cpp** file do the following changes:
  - add an ID **&IID_INam**
  - implement the code for **Run()** method
  - implement the code for **Abort()** method

```
// ArraySmp1.cpp : Implementation of CArraySmp1

#include "stdafx.h"

#include "NamSvr.h"

#include "ArraySmp1.h"


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

// CArraySmp1


STDMETHODIMP CArraySmp1::InterfaceSupportsErrorInfo(REFIID riid)

{

	static const IID* arr[] =

	{

		&IID_IArraySmp1,

		&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 CArraySmp1::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 CArraySmp1::Abort()
    {
        return E_NOTIMPL;
    }
```

- In the **ArraySmp1.h** file, remove the body for **Run()**, **Abort()** methods in order to avoid compiling errors. Leave clean methods declarations. Change the following lines:

```
    // IArraySmp1
    public:
    // INam
        STDMETHOD(Run)(VARIANT varArgs, LONG lNamOptions, INamServices * pNamServices, VARIANT
    varReserved)
        {
            return E_NOTIMPL;
        }
```

```
STDMETHOD(Abort)()

{

        return E_NOTIMPL;

}
```

into:

```
// IArraySmp1

public:

// INam

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

    STDMETHOD(Abort)();
```

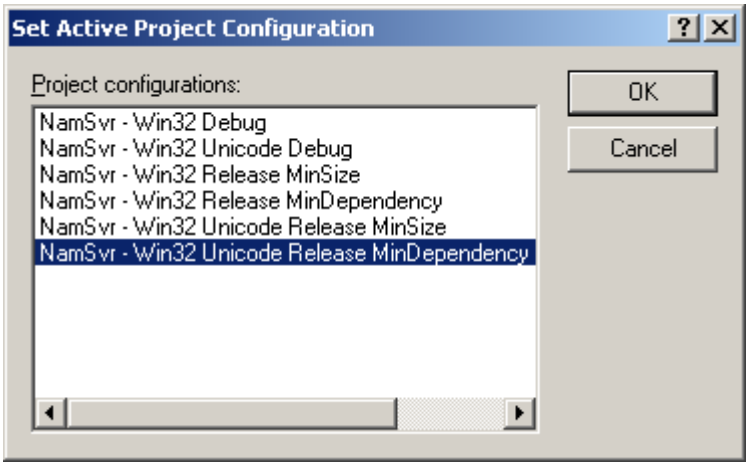
- Now you should be able to build the project without any errors. Press **F7** button or use the Build icon from the Toolbar.

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:

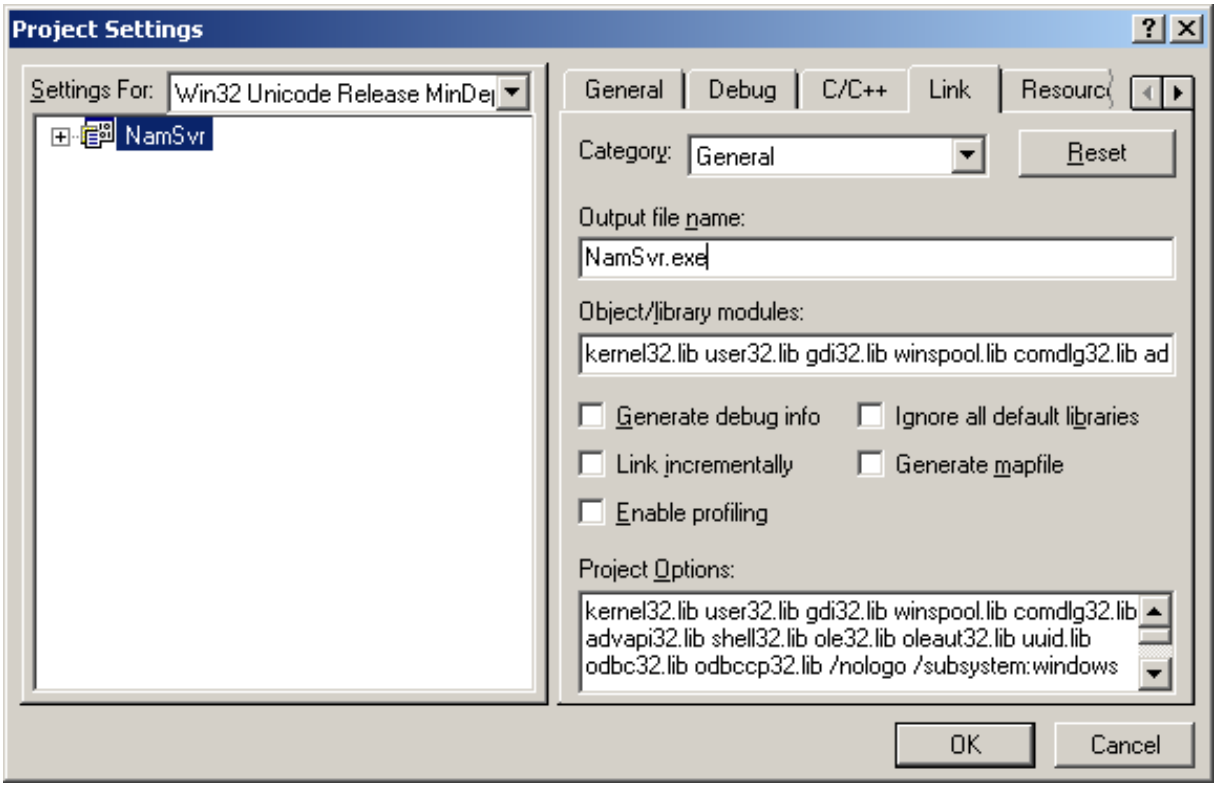
    NamSvr – Win32 Unicode Release MiniDependency

    as shown below:



- Press **OK**.
- For a debug session, you will want to select **NamSvr – Win32 Debug**.

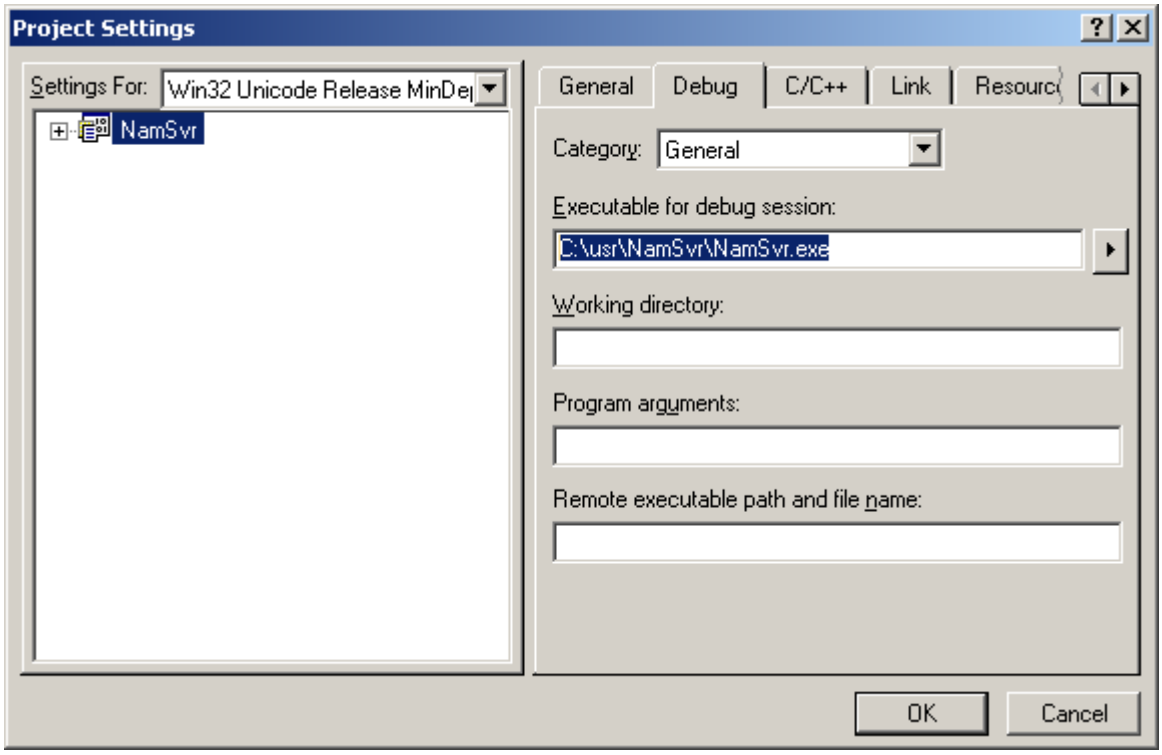
- From the menu bar, under **Project | Settings | Link | Category: General** set the **Output file name:** to **NamSvr.exe** without any path:



This will allow the .exe file to be generated in the main project directory. Eventually, you will be able to make use of this COM-NAM exe 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 exe. The command line to register the exe is the following: **<COM component name>.exe / Regserver**.

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

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



Note: for a debug session, you will need to make sure that you are generating a debug version of the exe by setting the right configuration.

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_SAMPLE1'                          $
C                                                                       $
     10 INCLUDE, NON-ATLAS MODULE 'NAMSVR.SAMPLE1'                      $
     15 INCLUDE, NON-ATLAS MODULE 'NAMSVR.ARRAYSMP1'                    $
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.SAMPLE1' ('B', 'I', 'R', 'T', 'D')             $
     65 PERFORM, 'NAMSVR.ARRAYSMP1' ('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_SAMPLE1'                      $
```

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.

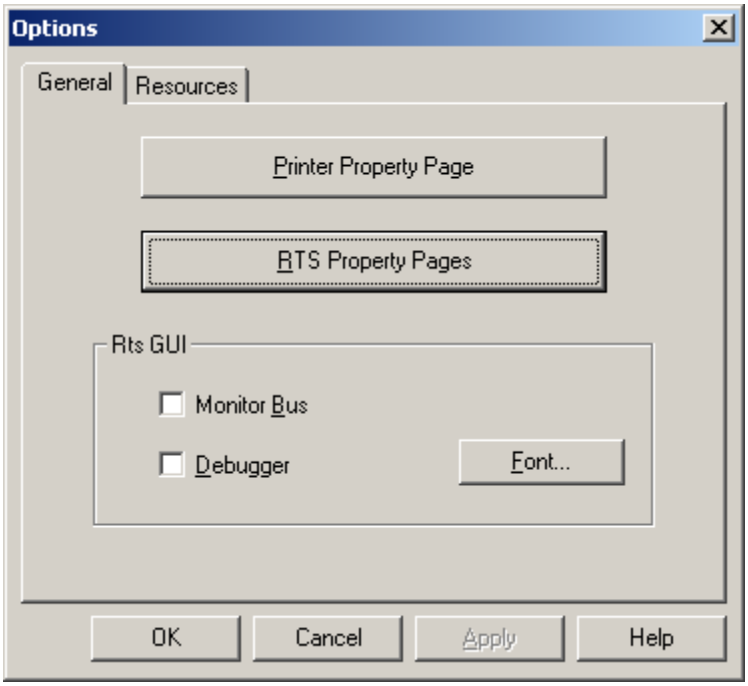
In the example above, the Atlas code includes two COM-NAM **ProgID's**: **NAMSVR.SAMPLE1** and **NAMSVR.ARRAYSMP1**. The first one takes all the parameter values from the Atlas (excluding the array type), processes them, and sends the modified values back to Atlas. The second one handles the parameters of the array type.

COM-NAM ProgID name cannot be longer than 16 characters to be recognized by WRTS!

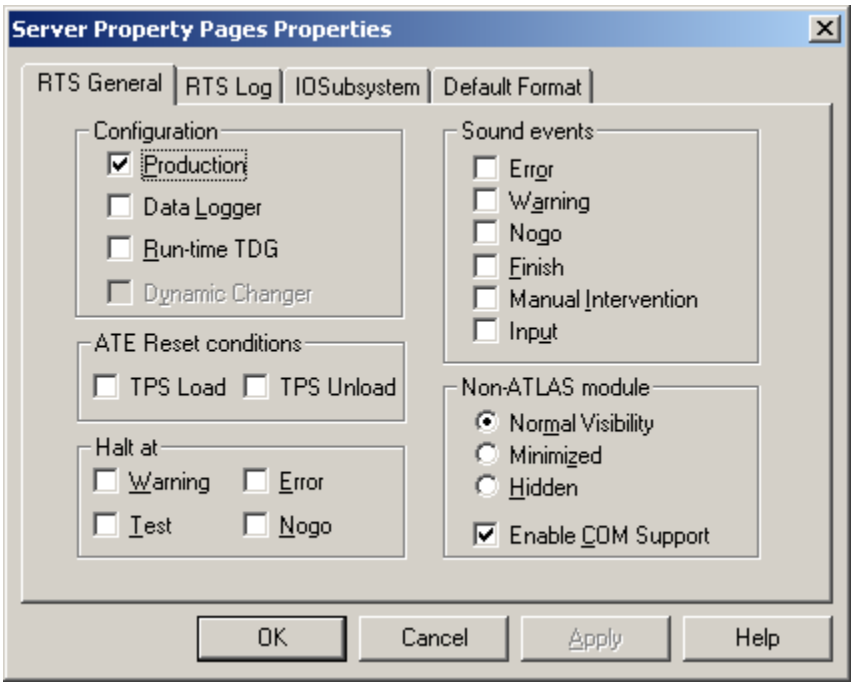
2.2 Wrts Settings:

- 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 exe will be invoked regardless of its location after it has been registered, either by the building process of the COM-NAM exe, or manually from a command line using **<COM-NAM name>.exe / Regserver**.

2.3 Having trouble?

2.3.1 WRTS warning:

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

WARNING: COM Non-ATLAS Module 'NAMSVR.SAMPLE1', COM Error 0x80070002, 'The system cannot find the file specified.

WARNING: COM Non-ATLAS Module 'NAMSVR.ARRAYSMP1', COM Error 0x80070002, 'The system cannot find the file specified.'

2.3.1.1 Solution:

This message can be symptomatic of an improperly registered exe.

In order to correct the problem, you will need to register the exe. Re-registering will be sufficient to solve that problem.

When registering the exe from a command line use the following command **<COM-NAM name>.exe / Regsvr**,

2.3.2 WRTS warning:

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

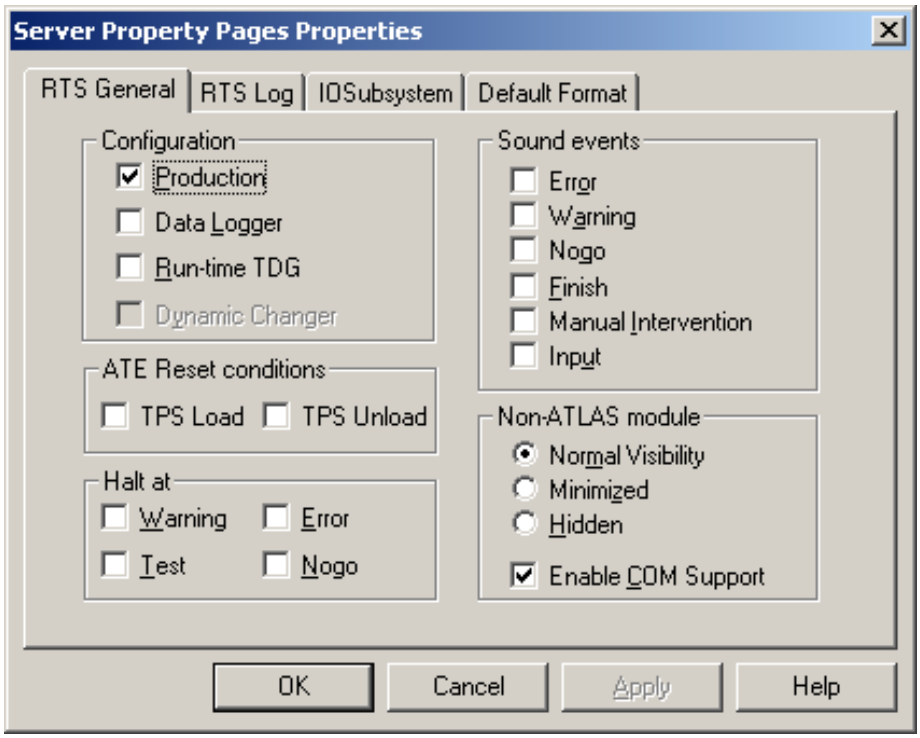
WARNING: Failed to start Non-ATLAS Module: NAMSVR.SAMPLE1 "C:\DOCUME~1\bartek\LOCALS~1\Temp\ComNam" 6952 6956 6960 6968 7012

WARNING: Failed to start Non-ATLAS Module: NAMSVR.ARRAYSMP1 "C:\DOCUME~1\bartek\LOCALS~1\Temp\ComNam" 7020 7168

2.3.2.1 Solution:

These messages can 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-Nam exe?

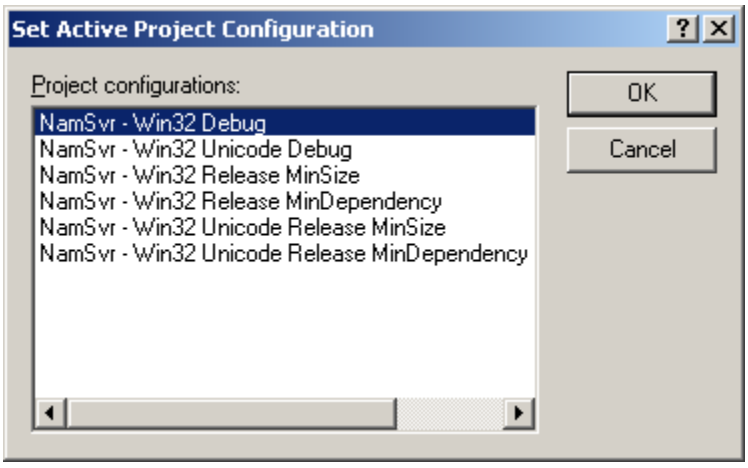
- The problem of trying to debug a COM-NAM exe is the same as for the NAM exe. You only have the time it is being executed to attach to the process. The only way to do this is to slow it down, or to stop it until you have a chance to attach to the process and put a breakpoint it the source code before you proceed.
- At the beginning of the source file you want to debug (in your c/cpp file), add for instance an infinite loop that will exit only if the status of the variable is changed at debug time.
  For example: (Debug code added into **Sample1.cpp**)

```
// Sample1.cpp : Implementation of CSample1

#include "stdafx.h"

#include "NamSvr.h"

#include "Sample1.h"


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

// CSample1


STDMETHODIMP CSample1::InterfaceSupportsErrorInfo(REFIID riid)

{

    static const IID* arr[] =

    {

            &IID_ISample1,

     &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 CSample1::Run(VARIANT varArgs, LONG lNamOptions,
INamServices * pNamServices, VARIANT varReserved)
{
    // Code within "if defined" statement is for debugging purposes
    // and will be executed only when the exe is created in a debug mode.
    // Once the NamSvr process is attached the "while" condition should be
evaluated to false to exit the loop.
    #if defined _DEBUG
    bool b_stop = false;
    while (!b_stop)
    {
        ;
    }
    #endif;
//… the rest of the code
```

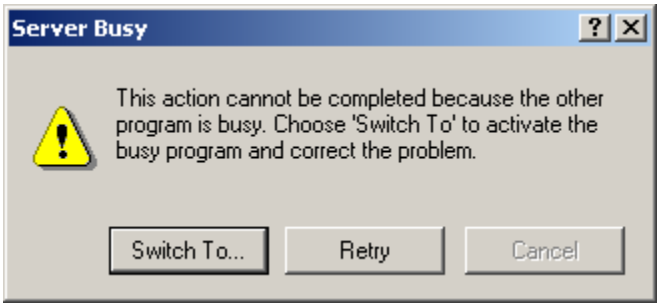
Note: The **#ifdefined** is there to make sure that this code is not taken in consideration for the release mode, i.e., it will only go into the infinite loop and “halt” the NAM for the debug version.

- Build the NAM EXE from MSVC in such a way that it is built in a debug mode.



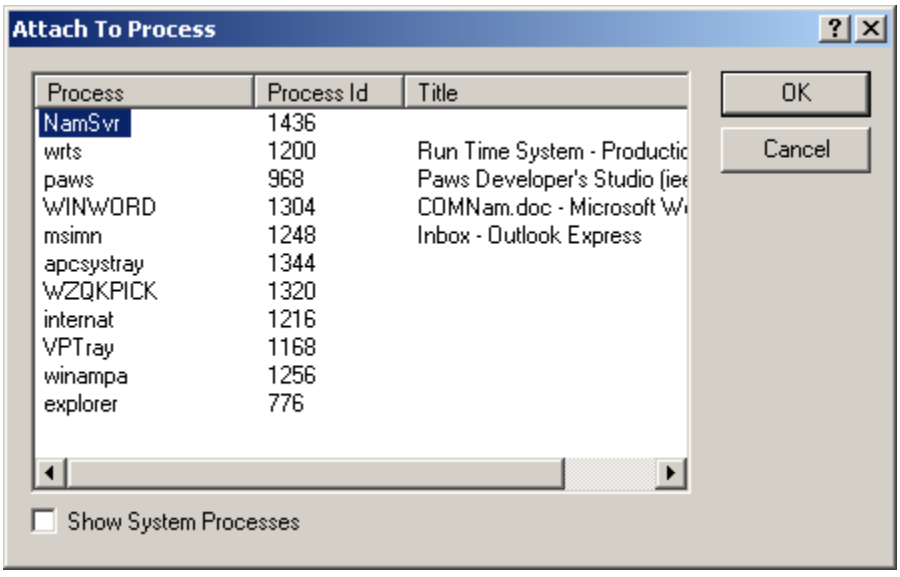
- Start the RTS and load the project that will call the NAM server (NamSvr.exe)

- Run the RTS. The NAM such entered should "hang" at this point - due to the infinite loop in it. At that time you may get the following Message, which you have to ignore:



- You are in the middle of the execution of the TPS

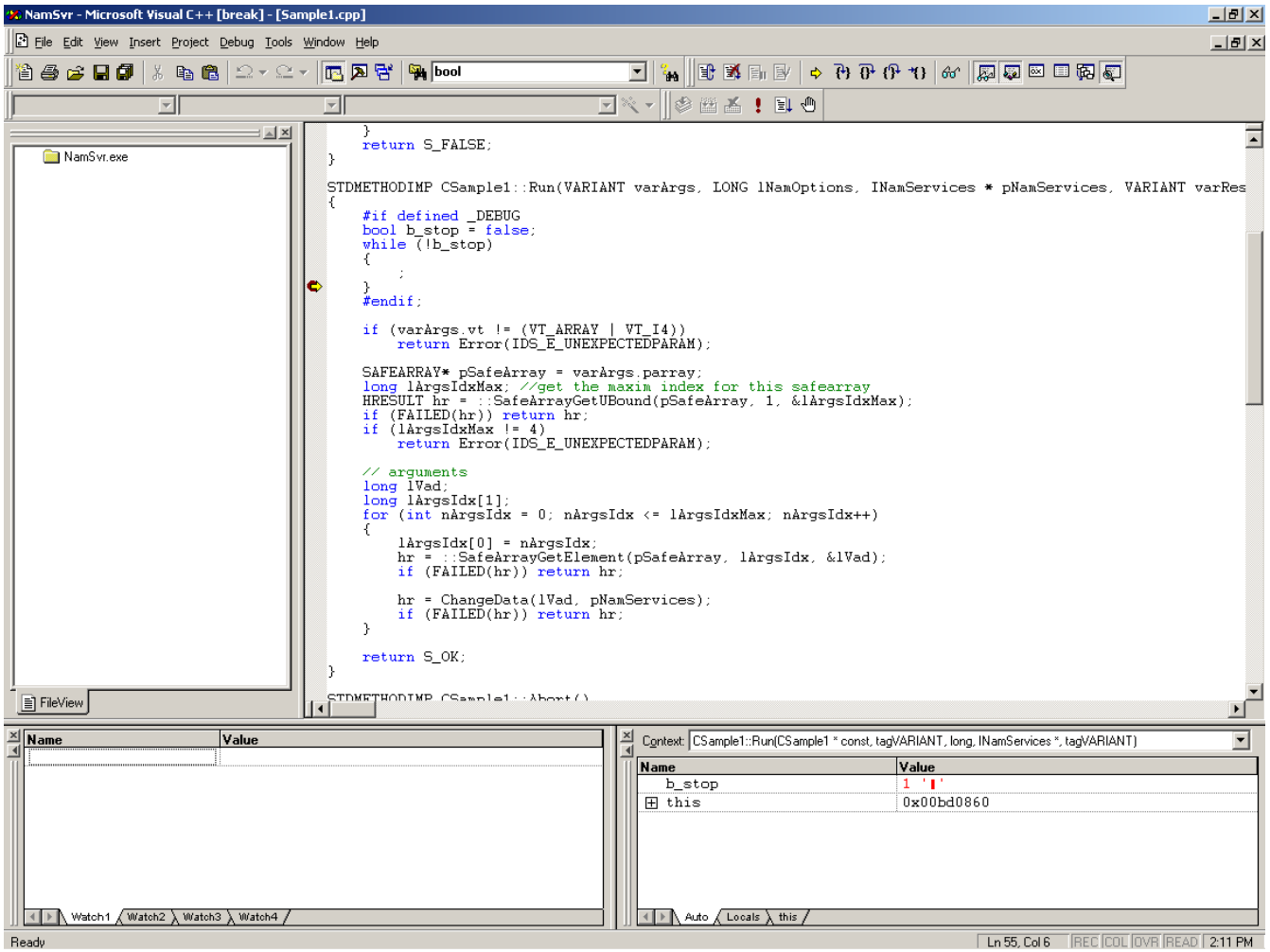
- In the MSVC, you will go to **Build/Start Debug/Attach to Process...**.
- Select **NamSvr**



- When or if asked if you want to save the changes to your project, say "**yes**".

- Now, if the source file vanishes from the MSVC edit window, re-open it in order to view it.
- Place the break point at the end of the while loop and the cursor should appear to indicate that you halted right there.
- Change the condition variable in a 'while' loop (in this example, **bStop**) to 1 in order to exit the loop.

```
#if defined _DEBUG

bool b_stop = false;

while (!1)

{

        ;

}

#endif;
```



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

References:

COM-NAM User's Guide