TYX Corporation

Productivity Enhancement Systems

Reference	TYX_0051_18
Revision	1.0b
Document	Nam_Labwindows.doc
Date	Nov 21, 2005

How to create a NAM with LabWindows / CVI?

This document will help with creating the standard NAM component for Paws RTS using LabWindows / CVI.

TYX PAWS Studio version used: **1.33.0**

LabWindows / CVI (Measurement Studio): **6.0**

Requirements:

- **TYX's PAWS Developer's Studio V. 1.2.0 or later with Simulator or RTS.**
- **National Instrument's LabWindows / CVI**

Introduction: **This document describes how to create a Standard Non-ATLAS Module (NAM) as an exe using LabWindows / CVI. First part specifies the environment requirements and it includes a complete ATLAS program example. The remaining part of the document is dedicated to the details of LabWindows/ CVI project and it shows working example of 'c' code .**

1 Creating Non-ATLAS Modules with LabWindows / CVI

This application note will demonstrate the process of creating non-ATLAS modules for use with PAWS ATLAS programs using National Instruments LabWindows / CVI. In this example we use PAWS Developer's Studio version 1.33.0 and LabWindows / CVI version 6.0.

1.1 Non-ATLAS Module Overview

Throughout the history of the ATLAS programming language there have always been situations that have been outside the scope of ATLAS. The strengths of ATLAS lie in test signal description, but it doesn't have the built in constructs to provide easy access external programs, operating system functions, and other types of functions normally associated with application programming languages such as 'C', BASIC, FORTRAN, etc.

Non-ATLAS modules give the programmer an alternative when ATLAS constructs won't get the job done. A non-ATLAS module is a program written in a language other than ATLAS, typically an executable program that is included into the ATLAS project and is executed like a procedure. Non-ATLAS modules have been used for a variety of functions from performing digital testing to creating Integrated Electronic Technical Manuals (IETM's).

Non-ATLAS modules can be passed parameters just like an ATLAS procedure. Any ATLAS data type is legal. You will see that the only difference between executing a non-ATLAS module and an ATLAS procedure is that no RESULT parameters are used when a non-ATLAS module is executed.

When a non-ATLAS module is executed, the RTS waits until the NAM has terminated prior to resuming execution of the ATLAS program.

The PAWS system uses the ‘C’ language and Microsoft Visual C++ for non-ATLAS modules.

1.2 Non-ATLAS Module Requirements

To create non-ATLAS modules using LabWindows / CVI you will need the following elements:

- TYX’s PAWS Developer’s Studio V. 1.2.0 or later with Simulator or RTS.
- An ATLAS program that INCLUDE’s and PERFORM’s the non-ATLAS module.
- National Instrument’s LabWindows / CVI
- The TYX supplied header file ‘C:\USR\TYX\INCLUDE\NAM.H’
- The TYX supplied export library ‘C:\USR\TYX\LIB\NAM.LIB’
- The TYX supplied dynamic linked library ‘C:\WINDOWS\NAM.DLL’

The directories may vary depending on the version of windows you are running and where your PAWS system is installed.

1.3 Writing an ATLAS Program for a Non-ATLAS Module

The requirements in your ATLAS code to reference a non-ATLAS module are simple. You only need an INCLUDE statement to make the NAM part of your <complete atlas program structure>, and a PERFORM statement to execute it. The non-ATLAS module is not part of the ATLAS project in PAWS and should not be added to the project. The following code is a minimal ATLAS program that only includes a non-ATLAS module and executes it, passing several parameters.

ATLAS program example:

```
C                                                                    $
000100 BEGIN, ATLAS PROGRAM 'NAM_SAMPLE'                             $
C                                                                    $
000200 INCLUDE, NON-ATLAS MODULE 'cvi_nam'                           $
C                                                                    $
000400 DECLARE, VARIABLE, 'B1', 'B2', 'B3' IS BOOLEAN                $
       DECLARE, VARIABLE, 'I1', 'I2', 'I3' IS INTEGER                $
       DECLARE, VARIABLE, 'R1', 'R2', 'R3' IS DECIMAL                $
       DECLARE, VARIABLE, 'T1', 'T2', 'T3' IS STRING (80) OF CHAR    $
       DECLARE, VARIABLE, 'D1', 'D2', 'D3' IS STRING (16) OF BIT     $
C                                                                    $
E100000 OUTPUT, C'TEST ACTIVATION OF A NON-ATLAS MODULE'             $
C                                                                    $
C*********************************************************************$
```

```
C                                                                       $
      CALCULATE, 'B1'= FALSE, 'I1'= 123, 'R1'= 1.234,
            'T1'= C'Input text', 'D1'= X'FEDC'                          $
C                                                                       $
         OUTPUT, C'INPUT  :\LF\',
         C\HT\Boolean', 'B1', C\LF\',
         C\HT\Integer', 'I1', C\LF\',
         C\HT\Decimal', 'R1':7:3, C\LF\',
         C\HT\Text   ', 'T1', C\LF\',
         C\HT\Digital', 'D1', C\LF\'                                    $
C                                                                       $
         PERFORM, 'cvi_nam' ('B1', 'I1', 'R1', 'T1', 'D1')              $
C                                                                       $
         OUTPUT, C'OUTPUT :\LF\',
         C\HT\Boolean', 'B1', C\LF\',
         C\HT\Integer', 'I1', C\LF\',
         C\HT\Decimal', 'R1':7:3, C\LF\',
         C\HT\Text   ', 'T1', C\LF\',
         C\HT\Digital', 'D1', C\LF\'                                    $
C                                                                       $
C***********************************************************************$
C                                                                       $
      CALCULATE, 'B1'= TRUE, 'B2' = FALSE, 'B3' = TRUE                  $
```

```
C                                                                       $
      CALCULATE, 'I1'= 111, 'I2' = 222, 'I3' = 333                      $
C                                                                       $
      CALCULATE, 'R1'= 1.111, 'R2' = 2.222, 'R3' = 3.333                $
C                                                                       $
      CALCULATE, 'T1'= C'First', 'T2' = C'Second', 'T3' = C'Third'      $
C                                                                       $
      CALCULATE, 'D1'= X'1111', 'D2' = X'2222', 'D3' = X'3333'          $
C                                                                       $
         OUTPUT, C'INPUT  :\LF\',
         C\HT\Boolean', 'B1', C\LF\',
         C\HT\Integer', 'I1', C\LF\',
         C\HT\Decimal', 'R1':7:3, C\LF\',
         C\HT\Text   ', 'T1', C\LF\',
         C\HT\Digital', 'D1', C\LF\',
         C\HT\Boolean', 'B2', C\LF\',
         C\HT\Integer', 'I2', C\LF\',
         C\HT\Decimal', 'R2':7:3, C\LF\',
         C\HT\Text   ', 'T2', C\LF\',
         C\HT\Digital', 'D2', C\LF\',
         C\HT\Boolean', 'B3', C\LF\',
         C\HT\Integer', 'I3', C\LF\',
         C\HT\Decimal', 'R3':7:3, C\LF\',
```

```
      C\HT\Text   ', 'T3', C\LF\',
      C\HT\Digital', 'D3', C\LF\'                                    $
C                                                                    $
      PERFORM, 'cvi_nam' ('B1', 'I1', 'R1', 'T1', 'D1',
            'B2', 'I2', 'R2', 'T2', 'D2',
            'B3', 'I3', 'R3', 'T3', 'D3')                            $
C                                                                    $
      OUTPUT, C'OUTPUT  :\LF\',
      C\HT\Boolean', 'B1', C\LF\',
      C\HT\Integer', 'I1', C\LF\',
      C\HT\Decimal', 'R1':7:3, C\LF\',
      C\HT\Text   ', 'T1', C\LF\',
      C\HT\Digital', 'D1', C\LF\',
      C\HT\Boolean', 'B2', C\LF\',
      C\HT\Integer', 'I2', C\LF\',
      C\HT\Decimal', 'R2':7:3, C\LF\',
      C\HT\Text   ', 'T2', C\LF\',
      C\HT\Digital', 'D2', C\LF\',
      C\HT\Boolean', 'B3', C\LF\',
      C\HT\Integer', 'I3', C\LF\',
      C\HT\Decimal', 'R3':7:3, C\LF\',
      C\HT\Text   ', 'T3', C\LF\',
      C\HT\Digital', 'D3', C\LF\'                                    $
```

```
C                                                                    $
C*********************************************************************$
C                                                                    $
999999 TERMINATE, ATLAS PROGRAM 'NAM_SAMPLE'                         $
```

This program expects the non-ATLAS module to change the value of the variables being passed. Note that the RESULT parameters are not used. This is because the address of the variable, not its value, is passed to the non-ATLAS module.

1.4 ATLAS Program Execution / Non-ATLAS Module Theory

After you build your ATLAS project there are two files created that are required at run-time:

*.OBJ file

*.DAT file

The OBJ file is the executable code of the ATLAS program, and the DAT file contains all program data (variables, literals, connection data, etc.). When you load a project into the Run Time System the DAT is copied to a TMP file. This temp file is opened as virtual memory and is used to access all ATLAS program data. The DAT file remains unaltered so you always start with a fresh copy of the ATLAS data.

When you call a non-ATLAS module a series of arguments are passed. The first argument is always the name of the non-ATLAS module. The second argument is the name of the ATLAS program. Arguments one and two are not in the perform statement parameter list. Arguments three through n are the addresses of the parameters being passed to the NAM, in character form.

It is up to the non-ATLAS module to open virtual memory, retrieve each of the parameters passed to it, alter and return any required data, and close virtual memory (performed by WRTS only).

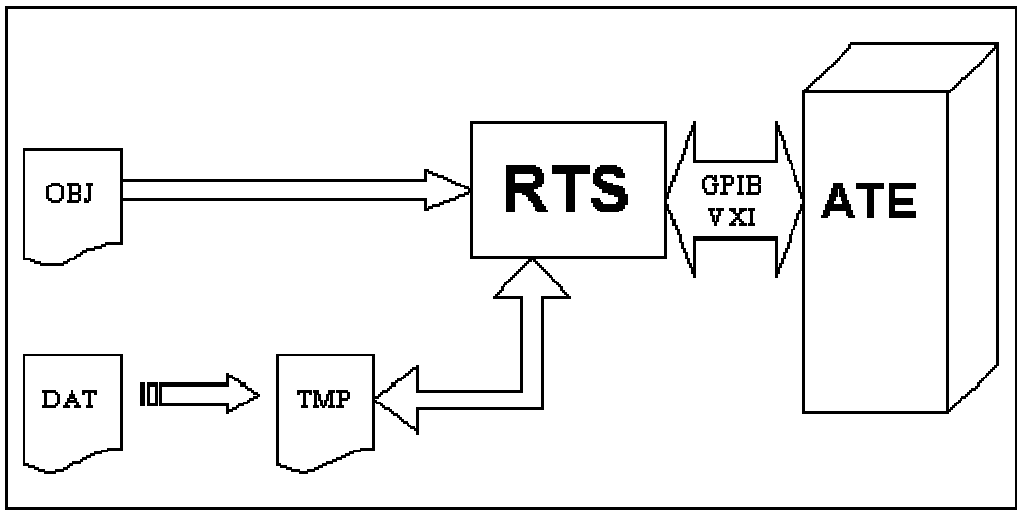

Figure 1

2 Creating the NAM in LabWindows / CVI environment

The first step in creating the non-ATLAS module in LabWindows / CVI is to create a new project (*.prj). Select to transfer all of the existing project options to the new project as performed on window below:

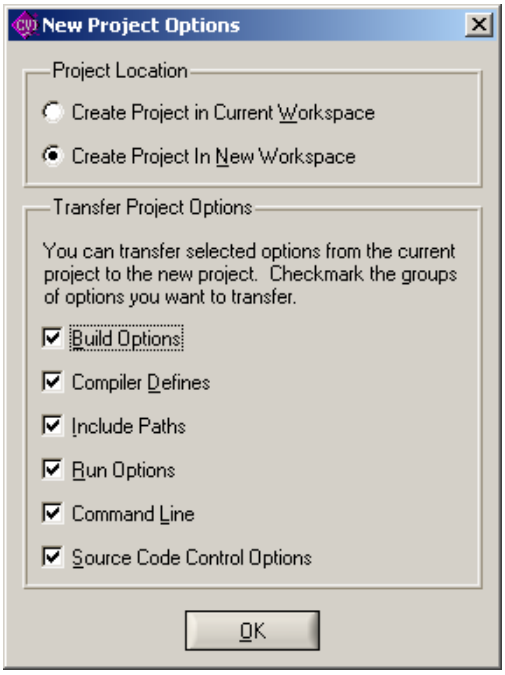

Figure 2.

When you create a new project after pressing OK the CVI main window should appear as in figure 3.

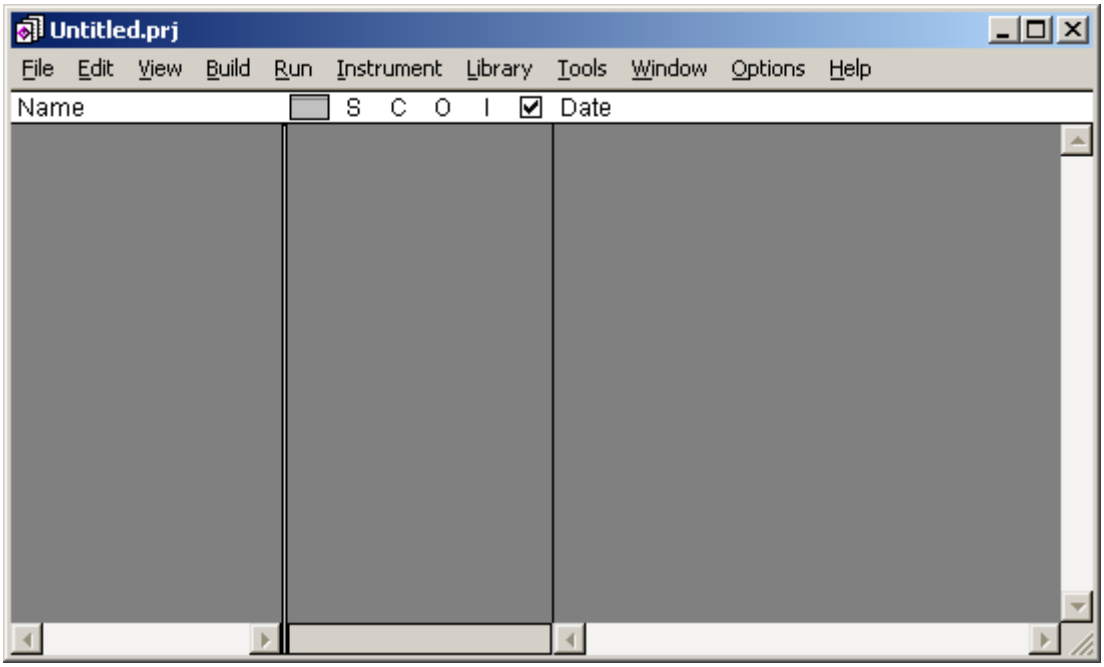

Figure 3.

Your next step is to save the project and to set the project options. Under the file menu select ‘Save Project As…’ and give your project a name. The target executable needs to be able to find the User Interface file (.UIR), so an easy way to do this is to save the CVI project in the same directory as your ATLAS project. If you move the CVI executable and it can’t find the UIR file, you will get a run-time error from the CVI run-time engine.

Next you need to set the target. Under the ‘Build’ menu select ‘Target type’ and ‘Executable’. This will create an .EXE file that can be executed by the ATLAS program. You will still need the CVI run-time engine, but CVI allows you to create a distribution disk so you can install your CVI NAM on another computer that doesn’t have CVI installed.

Your next step is to set any compiler options you need. For the compiler to be able to find the file ‘NAM.H’ you will need to set the include path. Select the ‘Options’ menu and ‘Include Paths…’. If the ‘Specific to Current Project:’ list isn’t active, click on ‘Switch Lists’, then click on ‘Add’. Type in the path name ‘C:\USR\TYX\INCLUDE’. This will enable the compiler to find the required header file. Click on ‘OK’ to exit.

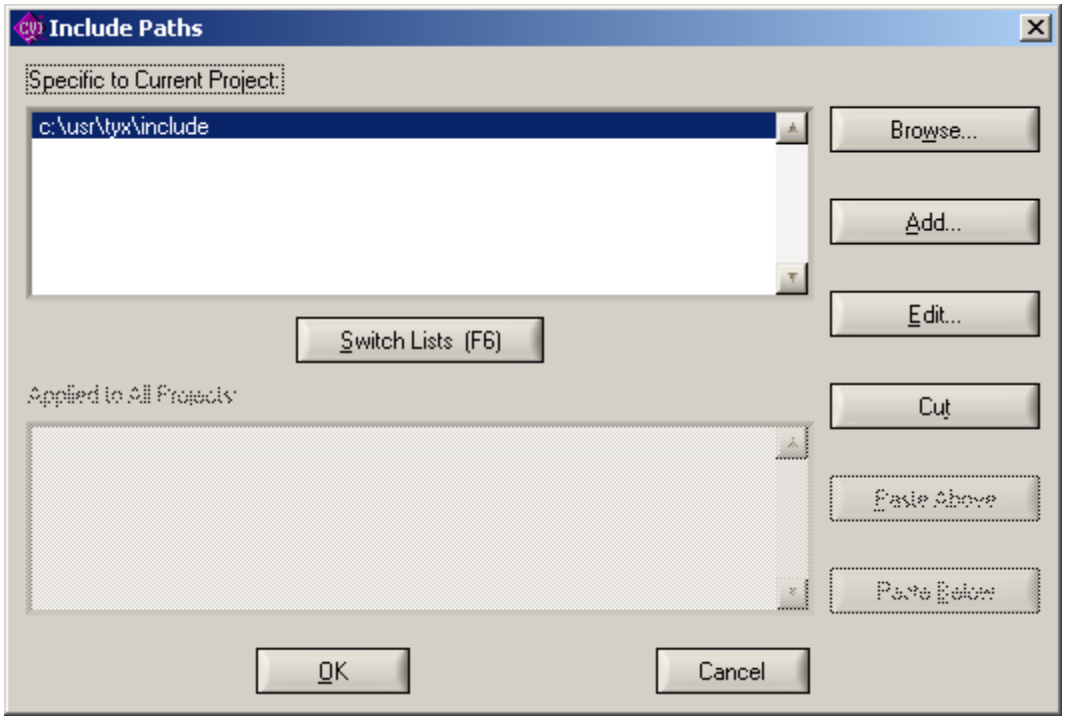

Figure 4.

2.1 Create the User Interface (UIR)

By using CVI we can assume that you want to have a graphical user interface for your NAM. Your next step is to build the user interface. From the UIR editor you can then automatically generate the template of the ‘C’ program. Figure 5 shows a UIR that was created with a series of indicators, which display the name of the ATLAS data, file, the number of arguments passed to the NAM, and the virtual address and value of each argument.

Note the control marked ‘Return to ATLAS’. It is required to have a control to exit the NAM. If you don’t have a ‘Quit’ callback function, you will never return control back to the PAWS RTS.

Once you have completed your UIR, you can generate the ‘C’ file by selecting the ‘Code’ menu, then ‘Generate’, and ‘All Code’. This will create a ‘C’ program

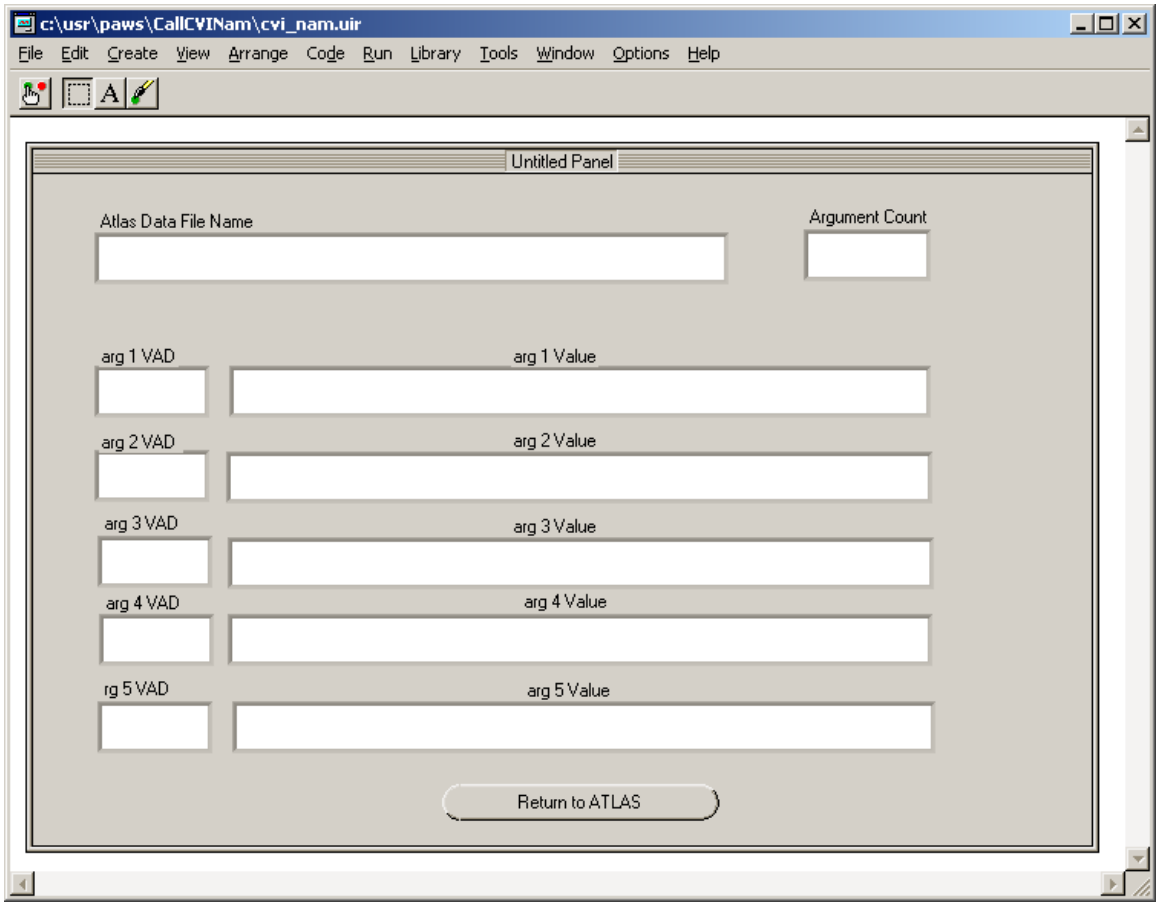

Figure 5.

with a main function, and all the appropriate callback functions for your active controls.

Note: The UIR presented in Figure 5 is just an example and can be freely customized according to the user's needs. You will not find details on how to create UIR in this tutorial and the 'c' code example presented below in 'Complete NAM code' does not have any references to UIR. To run the NAM (Non Atlas Module), which in this case is cvi_nam.exe you don't need to have the UIR. You will be able to see the how the parameters changing form the WRTS terminal as presented in the Figure 6.

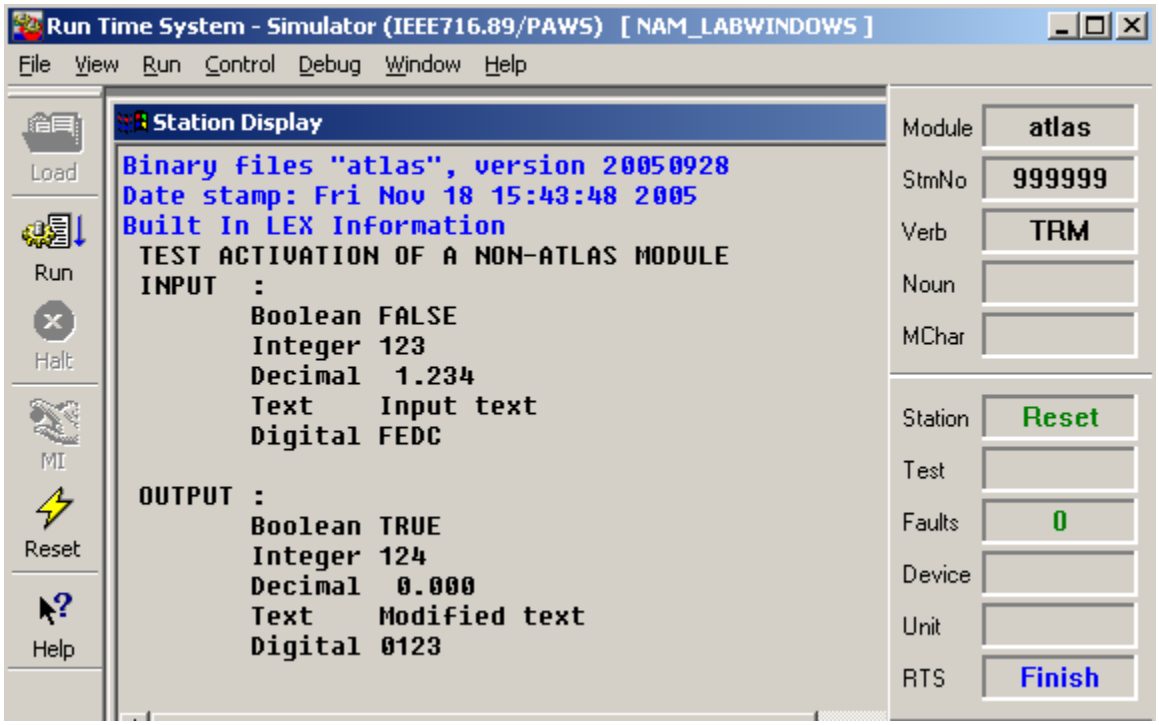

Figure 6.

2.2 Editing the ‘C’ Program

There are four basic steps for reading and writing ATLAS data in a non-ATLAS module:

Open virtual memory

Get the ATLAS data

Write the ATLAS data

Close virtual memory

2.2.1 Opening Virtual Memory

The first step is to include the file ‘nam.h’ in your ‘C’ program, and then to make a call to vmOpen() to enable access to virtual memory.

Note the #include statement and the function vmOpen() returns a negative value for failures.

Note: If you are using C++ to write your non-ATLAS module the virtual memory functions will throw exceptions defined in nam.h. CVI does not support C++ constructs. You can then use try/catch blocks to trap the exceptions.

2.2.2 Retrieving ATLAS Data

There are basically two steps in retrieving data from virtual memory. First, you need to convert the pointer passed in argv[] from a char type to a long. Next you use the function appropriate to the data type you are retrieving. The function vmGetDataType() will return the type of data at the address.

The prototypes of the functions and their associated constants are:

TBool vmGetBool(TLong vad); // constant: NTYPE

TLong vmGetInteger(TLong vad); // constant: ITYPE

TDouble vmGetDecimal(TLong vad); // constant: RTYPE

TInt vmGetText(TLong vad, PChar prTxt, TInt nMax);

// constant: TTYPE

TInt vmGetDigital(TLong vad, PWord prDig, TInt nMax);

// constant: DTYPE

The type definitions are in c:\usr\tyx\include\typedef.h. You may need to include this file in your ‘C’ program to be able to declare variables if your version of CVI doesn’t resolve it from nam.h.

The basic code to retrieve a value from virtual memory follows. This will be expanded on in the example.

```
long vad;

TBool    boolVal;
TLong    intVal;
TDouble  decVal;
PChar    txtVal[MAX_TXT];
```

```
TWord    digVal[MAX_DIG];


vad = atol(argv[2]);
switch(vmGetDataType(vad))
{
   case NTYPE :
      boolVal = vmGetBool(vad);
      break;
   case ITYPE :
      intVal = vmGetInteger(vad);
      break;
   case RTYPE :
      decVal = vmGetDecimal(vad);
      break;
   case TTYPE :
      vmGetText(vad,txtVal,sizeof(txtVal)/sizeof(PChar));
      break;
   case DTYPE :
      vmGetDig(vad, digVal, sizeof(digVal)/sizeof(Tword));
      break;
   default :
      break;
}
```

For the types of boolean, integer, and decimal the return of the function is the value in virtual memory. For the text and digital types, the value is passed by reference into the second argument. The third argument is the max number of characters or bytes to return. This is calculated by dividing the size of the declared array by the size of the individual elements.

Again, these calls can be placed before or after the RunUserInterface() call (not implemented in the example below), but placing them before allows you to retrieve the data and post it to the user interface before processing any events.

2.2.3 Closing Virtual Memory

Note:

VmClose() function is responsible for closing virtual memory, however it cannot be used from the LabWindows / CVI code due to the known limitation of PAWS Developer's Studio. Regardless the fact WRTS is able to recover from it and complete the execution.

In Microsoft Visual Studio environment vmClose() function works without any obstacles.

2.3 Complete NAM code

```
#include <ansi_c.h>

#include "cvi_nam.h"

#include "nam.h"



static int panelHandle;
```

```
int i;


void getVar (TLong vad)
{
        TBool  bVal;
        int       iVal;
        TDouble         rVal;
        char     txt[MAX_TXT];
        TWord dig[MAX_DIG];
        char     buf[MAX_TXT + 80];

        switch (vmGetDataType (vad)) {
        case NTYPE :
                bVal = vmGetBool (vad);
                break;
        case ITYPE :
                iVal = vmGetInteger (vad);
                break;
        case RTYPE :
                rVal = vmGetDecimal (vad);
                break;
        case TTYPE :
                vmGetText (vad, txt, sizeof (txt) / sizeof (char));
```

```
            break;
        case DTYPE :
            vmGetDig (vad, dig, sizeof (dig) / sizeof (TWord));
            break;
        default :
            break;
        }
}

void changeVar (TLong vad)
{
        TBool  bVal;
        int             iVal;
        TDouble         rVal;
        char    txt[MAX_TXT];
        TWord dig[MAX_DIG];

        switch (vmGetDataType (vad)) {
        case NTYPE :
            bVal = vmGetBool (vad);
            bVal = !bVal;
            vmSetBool (vad, bVal);
            break;
```

```
        case ITYPE :
                iVal = vmGetInteger (vad);
                iVal ++;
                vmSetInteger (vad, iVal);
                break;
        case RTYPE :
                rVal = vmGetDecimal (vad);
                rVal -= 1.234;
                vmSetDecimal (vad, rVal);
                break;
        case TTYPE :
                vmGetText (vad, txt, sizeof (txt) / sizeof (char));
                vmSetText (vad, "Modified text");
                break;
        case DTYPE :
                vmGetDig (vad, dig, sizeof (dig) / sizeof (TWord));
                dig[0] ^= 0xffff;
                vmSetDig (vad, dig, 1);
                break;
        }
}

int main (int argc, char *argv[])
```

```
{
	if (InitCVIRTE (0, argv, 0) == 0)
		return -1;	/* out of memory */
	if ((panelHandle = LoadPanel (0, "cvi_nam.uir", PANEL)) < 0)
		return -1;
	DisplayPanel (panelHandle);

	for (i = 0; i < argc; i++)
		printf ("Argument #%d = %s\n", i, argv[i]);
	if (argc > 2) {
		if (vmOpen (argv[1]) < 0) {
			return -1;
		}
		for (i = 2; i < argc; i++)
			getVar (atol (argv[i]));
		for (i = 2; i < argc; i++)
			changeVar (atol (argv[i]));
		for (i = 2; i < argc; i++)
			getVar (atol (argv[i]));
		//vmClose (); this function cannot be used from LabWindows / CVI code
	}

	// RunUserInterface (); use this function when to call UIR when implemented
```

```
        DiscardPanel (panelHandle);

        return 0;

}
```

2.4 Handling Array Data

When you pass an array to a Non-ATLAS module, you can't pass a reference to the array itself as you would with a normal 'C' program. The ATLAS data file is arranged differently, so to pass an array you need to pass two parameters; the starting element of the array and a count. In the ATLAS program the call to the NAM would resemble:

```
123400 PERFORM, 'NAM'('MYARRAY'(1), 100) $
```

This would pass the first element of 'MYARRAY', and a '100' indicating the number of elements to pass. Remember that no bounds checking is done here so you will need to be careful in the ATLAS code to make sure that your array is declared at least as long as the number of elements you are passing.

Array's in ATLAS data file are stored contiguous, meaning that element two of the array is stored immediately after element one, and so on. So all you need to do in the NAM is determine the size of the data item, and increment the address. Remember that in your 'C' program the sizeof() function returns a byte count, and in the ATLAS data file the data addresses are word addresses.

To get the size of each ATLAS data type, get the size of the local type, divide it by two, and add one for the size/type header ATLAS uses with each item. Increment the address by this amount to get to the next element in the array.

The following code listing demonstrates how to retrieve an array of decimal data from the ATLAS data file.

```
#include "nam.h"

#include <iostream.h>

#include <stdlib.h>


int main(int argc,char *argv[])

{

   TDouble decData[100];

   TInt    count, i;

   long    countAddr, decArrayAddr, decArrayStart;


   if(vmOpen(argv[1]) < 0)

   {

      cout << "Cannot open virtual memory file " << \

         argv[1] << endl;

      return 1;

   }


   count = vmGetInteger(atol(argv[3]); //Get Array count

   decArrayAddr = atol(argc[2]);  //Get array start address
```

```
   decArrayStart = decArrayAddr; //Store starting address


   for (i = 0;i < count;i++)
   {
      decData[i] = vmGetDecimal(decArrayAddr);
      decArrayAddr += (sizeof(TDouble) / 2) + 1;
   }


//your code here


}
```

This stripped down example first opens virtual memory, then retrieves the count. Next the address of the first element of the array is retrieved and saved for later if the array is to be returned to the ATLAS data file. The ‘for’ loop cycles through each element of the array retrieving the current value and incrementing the address. Data can be returned to the ATLAS data file using the same technique, except substituting the vmSet... call for the vmGet… call.

2.5 Returning Data to ATLAS

If your LabWindows / CVIâ program alters any of the parameters it retrieved from the ATLAS data file, you will need to return data. The functions used for returning data to ATLAS substitute the name ‘Set’ for ‘Get’. So if you used the function vmGetDecimal() to retrieve your data, you’ll use vmSetDecimal() to return the data to the ATLAS data file.

The function prototypes are:

```
void vmSetBool(TLong vad, TBool bval);

void vmSetInteger(TLong vad, TLong ival);

void vmSetDecimal(TLong vad, TDouble rval);

void vmSetText(TLong vad, PCChar pTxt);

void vmSetDig(TLong vad, PCWord pDig, TInt nWords);
```

In each function the first argument is the virtual address of the parameter. This is the same value derived from argv[n]. The second argument is the value being returned, except for text and digital type where it is a pointer to the array. For digital types there is a third argument, which is the number of bytes in the digital word, which is the number of elements in the array of containing the digital value.

You can return data to the ATLAS program at any point prior to the QuitUserInterface() call in your exit function in the CVI ‘C’ program.

2.6 Building the NAM

Prior to building your NAM you need to add the library NAM.LIB to your project. From the LabWindows project windows select ‘Add files to project….’ from the ‘Edit’ Menu. Select ‘Library (*.lib)…’ and go to the directory ‘C:\USR\TYX\LIB’ to find the file ‘NAM.LIB’. When you are finished the project window should look like the following:

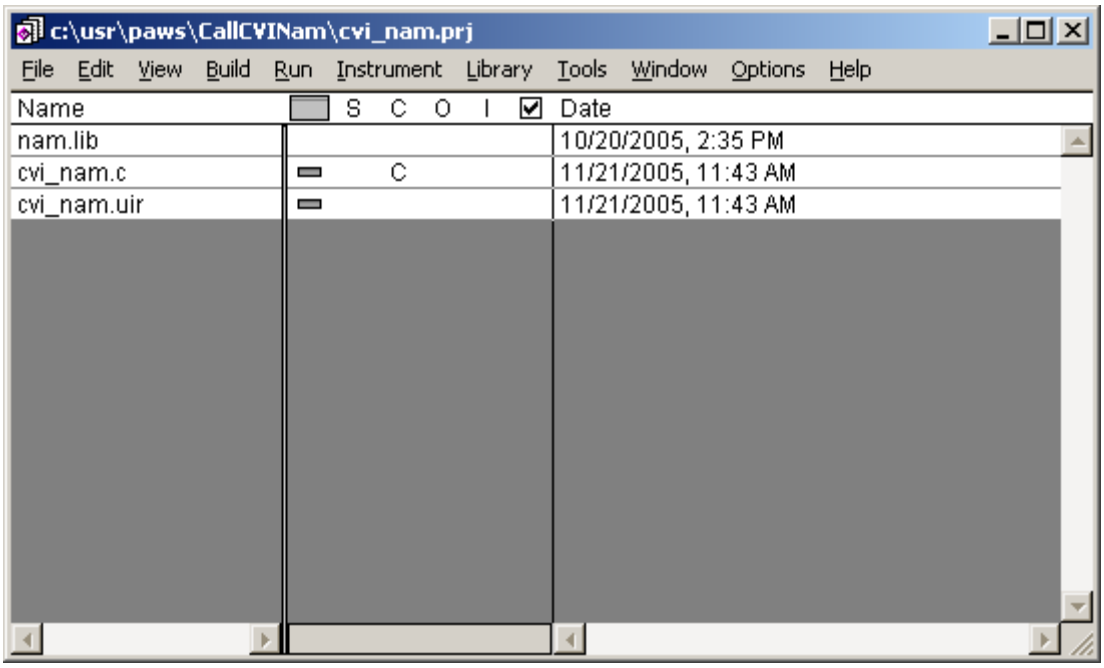

Figure 7.

You are now ready to build the Non-ATLAS module. First chose the needed configuration by checking Release / Debug from the Build | Configuration. Then select 'Create Release / Debuggable Executable' also from the 'Build' menu. The file 'cvi_nam.exe' should be created in the directory you specified. Remember that the EXE must be able to find the UIR file (if used), so it's best to create the CVI project in the same directory as the ATLAS project.

If you need to move the NAM to a computer that doesn't have LabWindows / CVI installed, you can create a distribution kit by selecting 'Create Distribution Kit…' from the 'Build' menu. This will create an Install Shield script and install a copy of the CVI run-time engine along with your Non-ATLAS Module.

3 Setting RTS NAM Options

The final step in creating and running your Non-ATLAS Module is setting the run-time options in the RTS. This allows you to set the startup mode for Non-ATLAS modules. Normal mode will create a console window if the NAM is a console application. Minimized starts the NAM minimized with a program bar icon. Hide will not create a console window. Choose this option if you create a NAM with no user interface or interaction. For the CVI NAM, choose ‘Normal’.

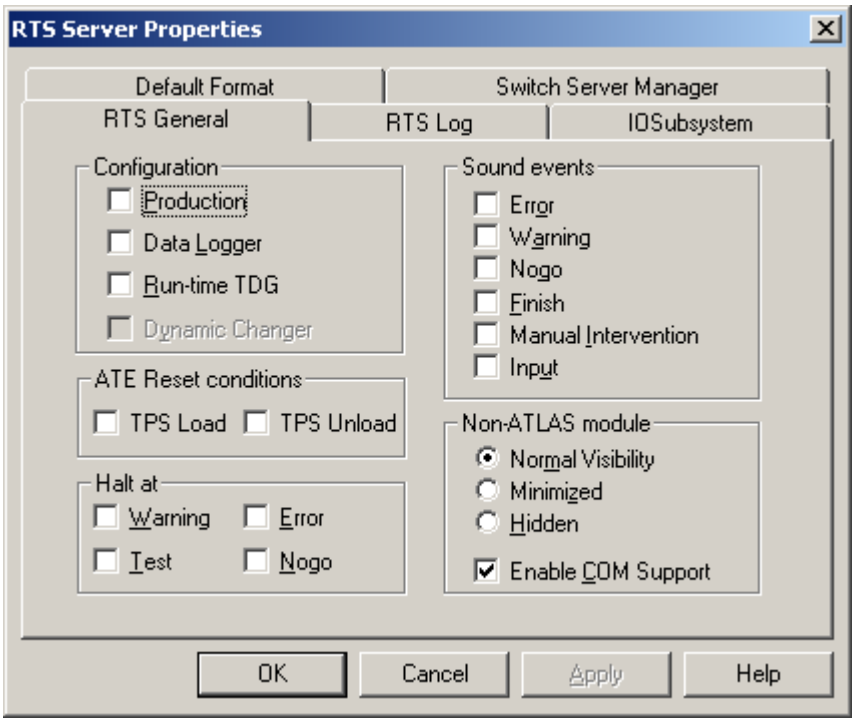

Figure 8.

At this point you are ready to run your ATLAS program. Non-ATLAS modules can be executed from the ATLAS Simulator or Run Time System, so you don’t need access to the ATE instrumentation to test and run your NAM. Open wrts.exe, load *.paw project and run it. It will call your nam_cvi.exe (NAM) automatically.