Tutorial – Creating a User-Defined Model

Last Updated: June 11, 2013

PowerWorld Corporation
2001 South First St
Champaign, IL 61820
(217) 384-6330
http://www.powerworld.com
info@powerworld.com

Overview

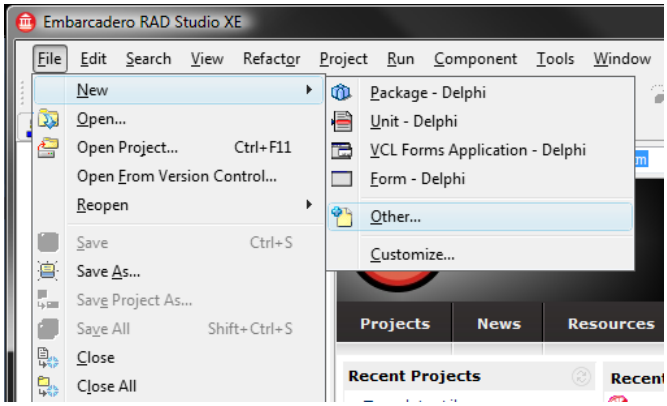
This tutorial details the creation of a DLL file containing a user-defined model for Transient Stability Analysis in PowerWorld Simulator. It is catered towards both beginners and expert programmers. Working knowledge of PowerWorld Simulator and a programming language of your choice is assumed.

Pascal, C++, and Fortran are used in this tutorial. Instructions may slightly differ across different versions of the IDE used. In the Getting Started sections, instructions are shown how to setup a project for GovernorUDM. For any other kind of model, the user can rename accordingly.

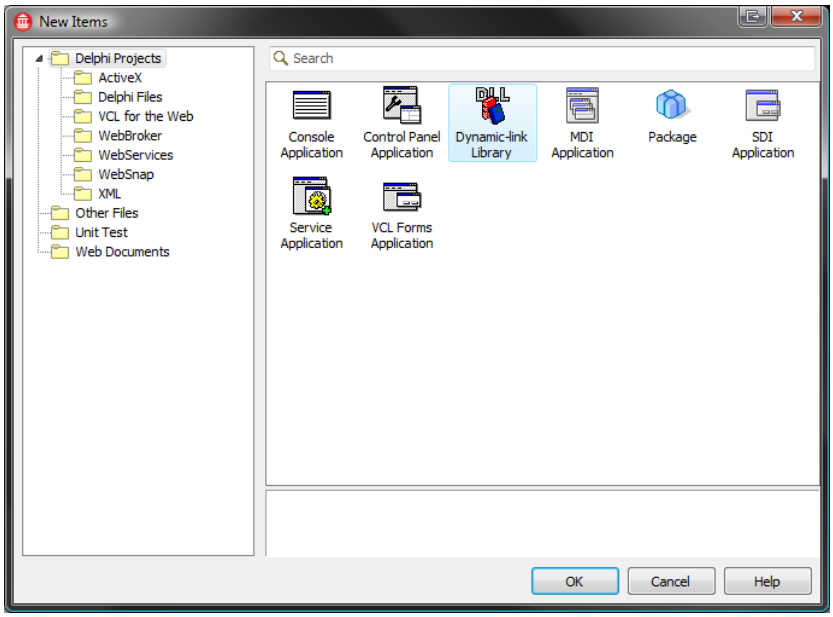
1. Object Pascal (Embarcadero® Delphi® XE Version 15.0)

Getting Started

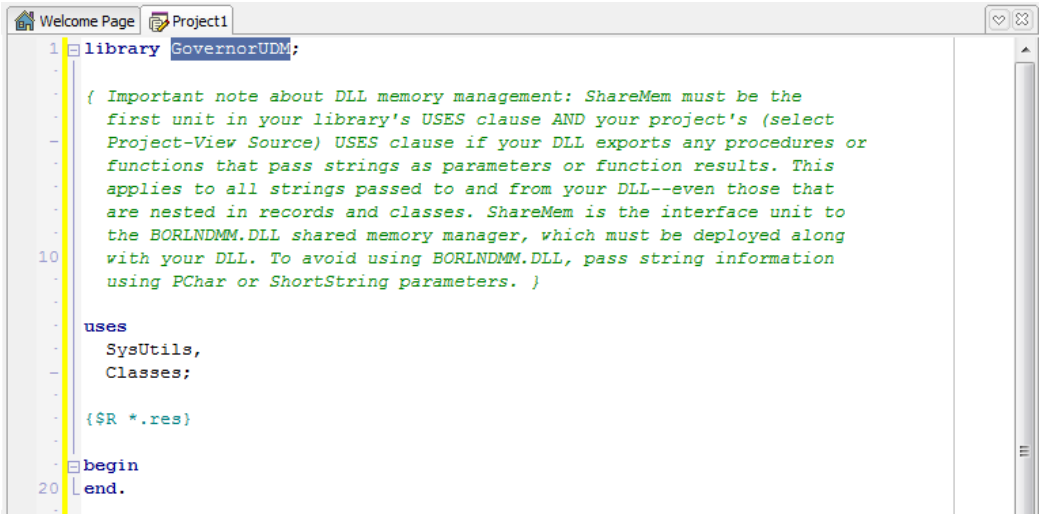
1. Start Delphi, and navigate to File > New > Other…



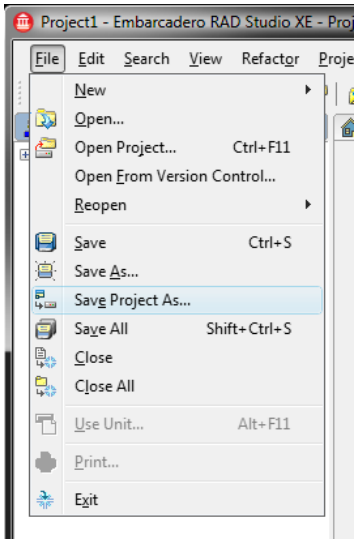
2. From Delphi Projects, choose Dynamic-link Library, and click OK.



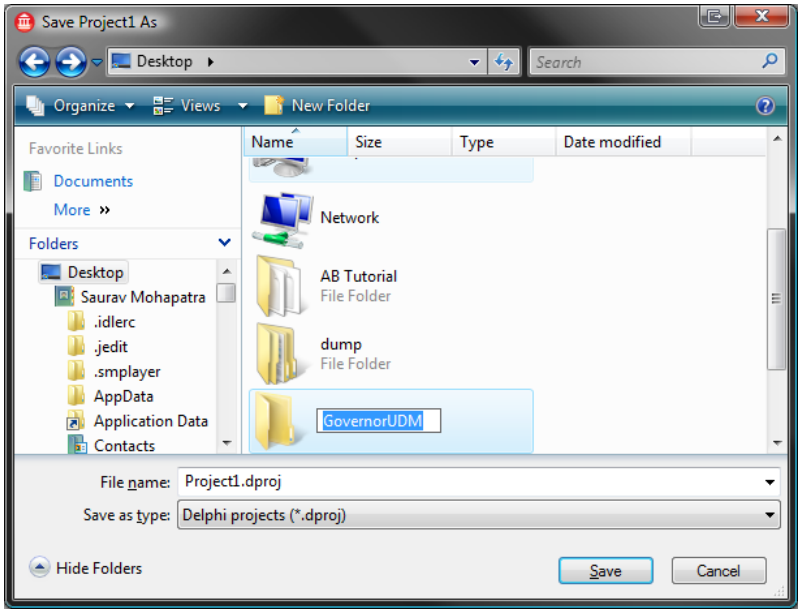
3. Delphi will automatically load a template. Rename the library to GovernorUDM. This is the name of the .dll file that will be created later, i.e., GovernorUDM.dll will be created when this project is eventually built.



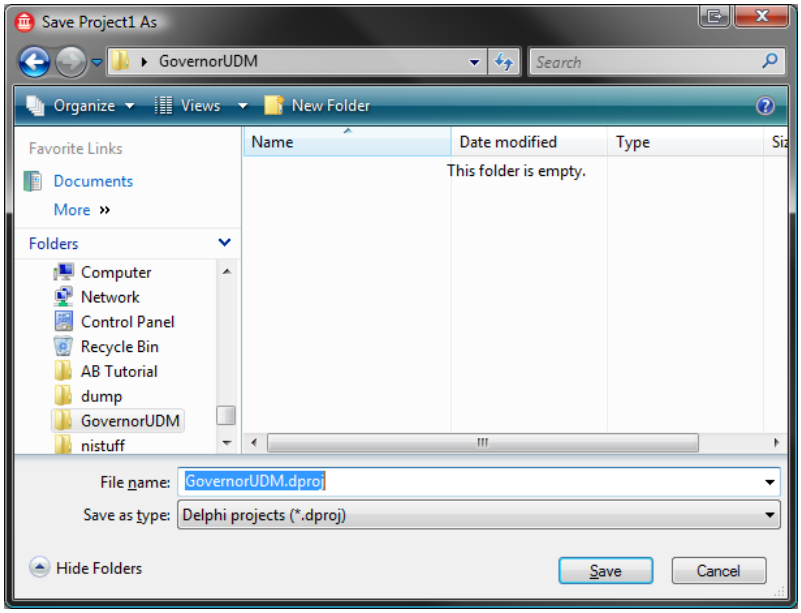
4. Navigate to File > Save Project As…



5. Navigate to the Desktop. Create a new folder, and rename it as GovernorUDM.

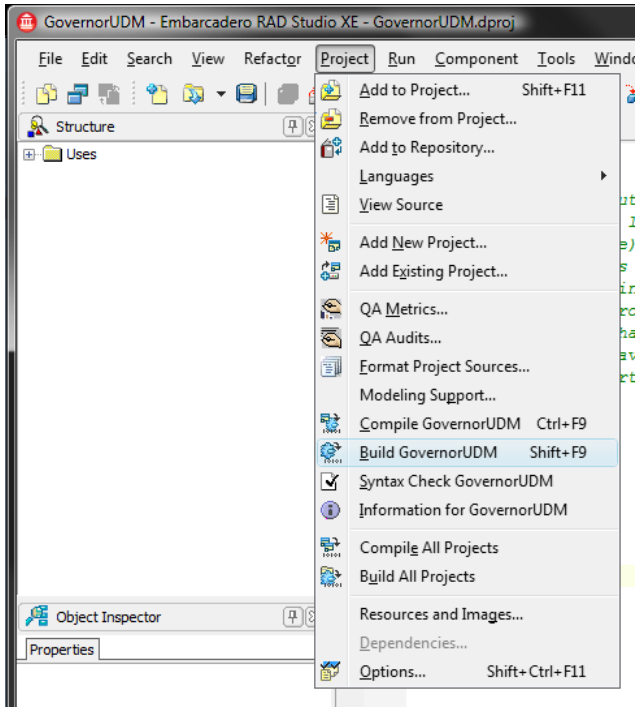


6. Open the GovernorUDM folder, and change the file name to GovernorUDM.dproj. Click Save.

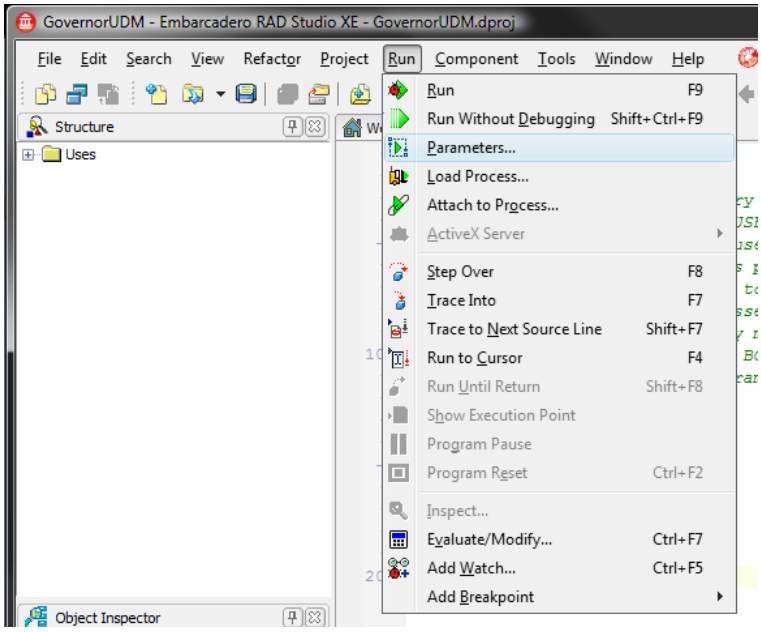


7. The project is now ready to use. Refer to the template, and sample models written in Pascal to populate the contents of this project to build a user-defined governor model of your choice.

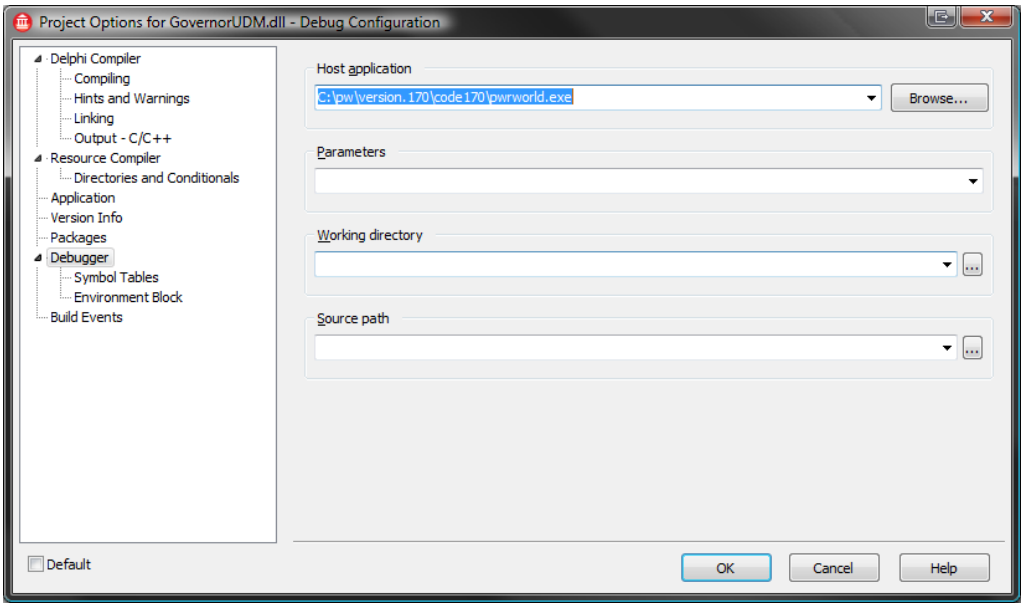
8. Once the content has been populated, the project can be built by navigating to Project > Build GovernorUDM.



9. Delphi also provides an option to debug the .dll file created from the Delphi side. To set this up, navigate to Run > Parameters…



10. The Debugger option should automatically be selected in the left pane. In the Host application box, populate the full path of the pwrworld.exe file. Click OK. This will allow the project to get attached to the pwrworld.exe process and breakpoints can thereafter be set within this project.



Preliminaries

Type definitions and pointers

Structure of integers

```
type
  TStructureOne = record // name of this structure type
    nIntegerOne: Integer; // name of the first element of this structure type
    …
    nIntegerN: Integer; // name of the nth element of this structure type
  end;
  PStructureOne = ^TStructureOne;
  // name of the pointer to this structure type
```

Structure of pointers to array of floating point numbers

```
type
  TDoubleArrayOne = array[0 .. (j1 - 1)] of Double;
  // name of this array type with j1 elements
  …
  TDoubleArrayM = array[0 .. (jM - 1)] of Double;
  // name of this array type with jM elements

  TStructureTwo = record // name of this structure type
    DoublesOne: ^TDoubleArrayOne {PDouble};
    // name of the first element of this structure type
    …
    DoublesM: ^TDoubleArrayM {PDouble};
    // name of the mth element of this structure type
  end;
  PStructureTwo = ^TStructureTwo;
  // name of the pointer to this structure type
```

Array of floating point numbers

```
type
  TDoubleArray = array[0 .. (k - 1)] of Double;
  // name of this array type with k elements
  PDoubleArray = ^TDoubleArray; // name of the pointer to this structure type
```

Function definitions

Exported functions

```
function functionName1(ArgInA: <Type>; … ArgInZ: <Type>): <Type>; stdcall;
var
  // Declare variables here
begin
  // Insert content here
end;

procedure procedureName1(ArgInA: <Type>; … ArgInZ: <Type>); stdcall;
var
  // Declare variables here
begin
  // Insert content here
end;
```

Functions internal to the .dll file

```
function functionName2(ArgInA: <Type>; … ArgInZ: <Type>): <Type>;
var
  // Declare variables here
begin
  // Insert content here
end;

procedure procedureName2(ArgInA: <Type>; … ArgInZ: <Type>);
var
  // Declare variables here
begin
  // Insert content here
end;
```

Dereferencing

Pointer to structure of integers

```
with StructureOne^ do begin
  // Receiving
  IntOne := nIntegerOne;
  …
  IntN := nIntegerN;

  // Retruning
  nIntegerOne := IntOne;
  …
  nIntegerN := IntN;
end;
```

Pointer to structure of pointers to array of floating point numbers

```
with StructureTwo^ do begin
  // Receiving
  DoubOne1 := DoublesOne^[0];
```

```
  …
  DoubMOnej1 := DoublesOne^[(j1 - 1)];
  …
  DoubM1 := DoublesM^[0];
  …
  DoubMjM := DoublesM^[(jM - 1)];

  // Retrurning
  DoublesOne^[0] := DoubOneOne;
  …
  DoublesOne^[(j1 - 1)] := DoubMOnej1;
  …
  DoublesM^[0] := DoubM1;
  …
  DoublesM^[(jM - 1)] := DoubMjM;
end;
```

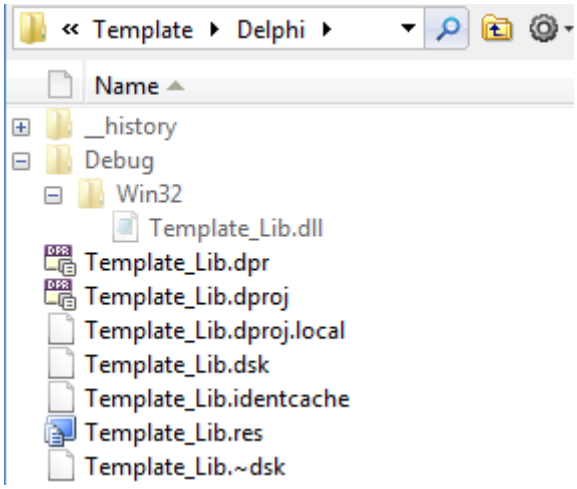
Pointer to array of floating point numbers

```
// Receiving
Doub1 := DoubleArray^[0];
…
Doubk := DoubleArray^[(k - 1)];

// Returning
DoubleArray^[0] := Doub1;
…
DoubleArray^[(k - 1)] := Doubk;
```

Making functions available in the export directory of the .dll file

```
exports
  functionName1, // continue list with a comma
  procedureName1,
  functionName2,
  procedureName2; // end list with a semi-colon
```

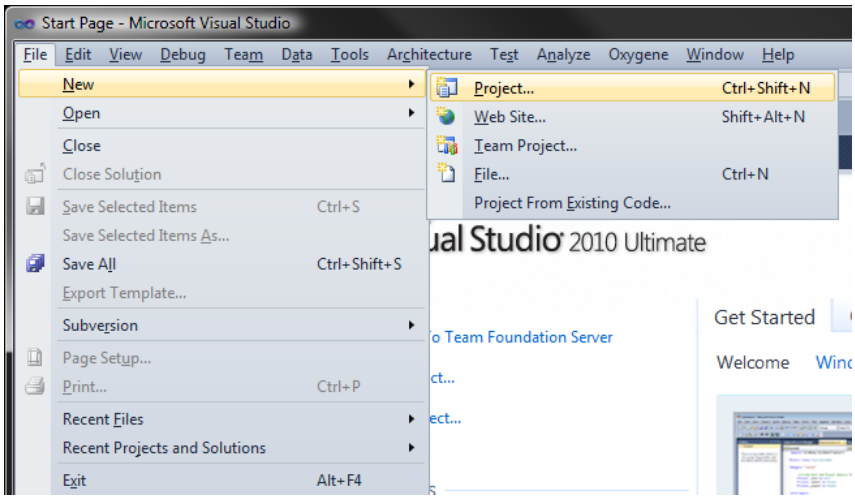
Project file/directory structure



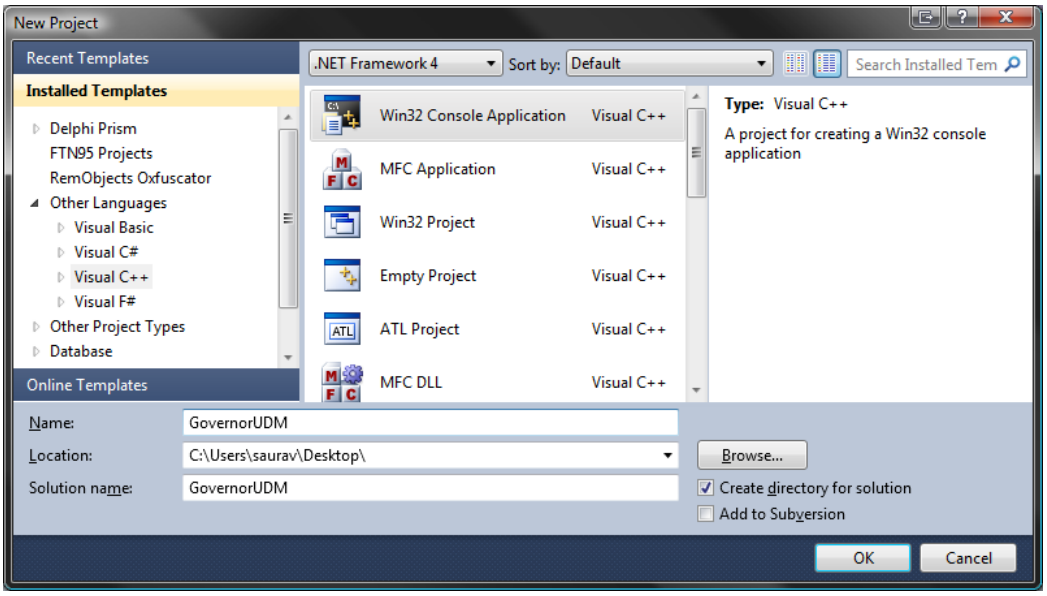
2. Visual C++ (Microsoft® Visual Studio 2010)

Getting Started

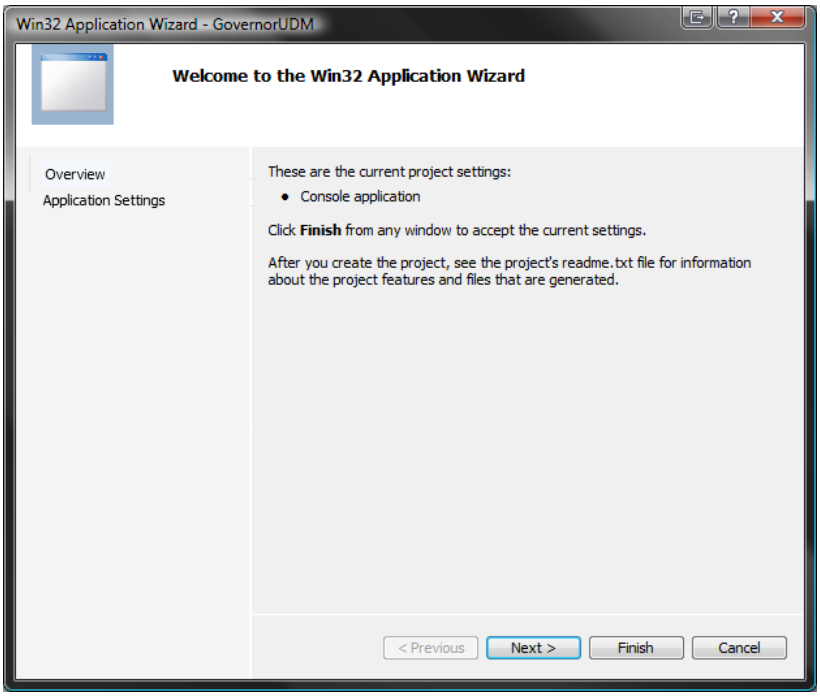
1. Start Visual Studio, and navigate to File > New > Project…



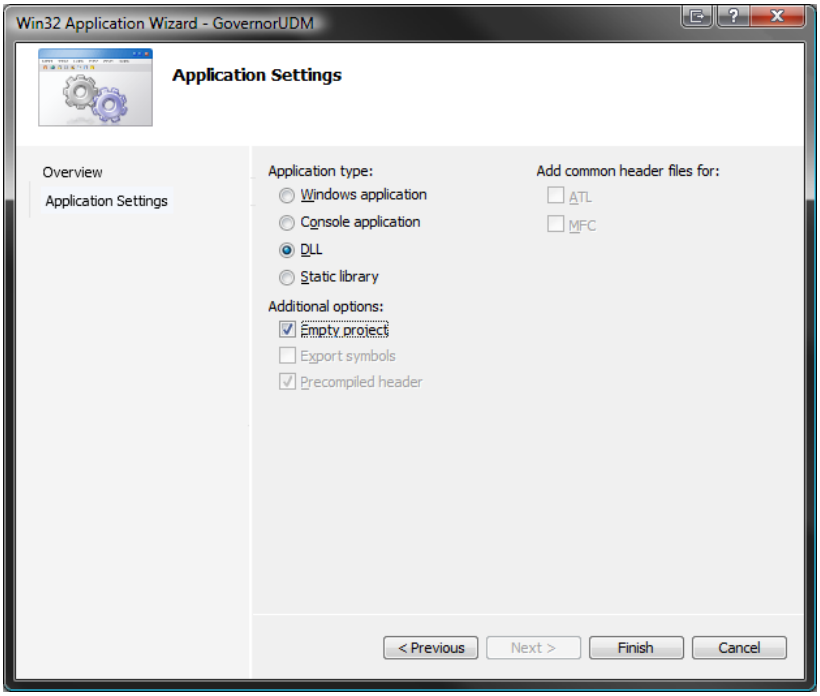
2. From Installed Templates, navigate to Other Languages > Visual C++. Choose Win32 Console Application. Set Desktop as the Location. Type GovernorUDM in the Name, and Solution name boxes. This is the name of the .dll file that will be created later, i.e., GovernorUDM.dll will be created when this project is eventually built. Ensure Create directory for solution is checked, and Click OK.



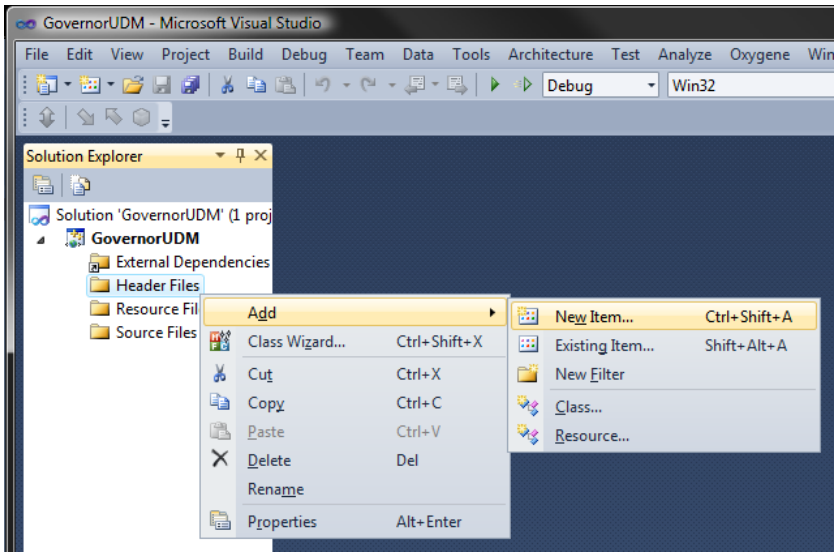
3. Click Next >.



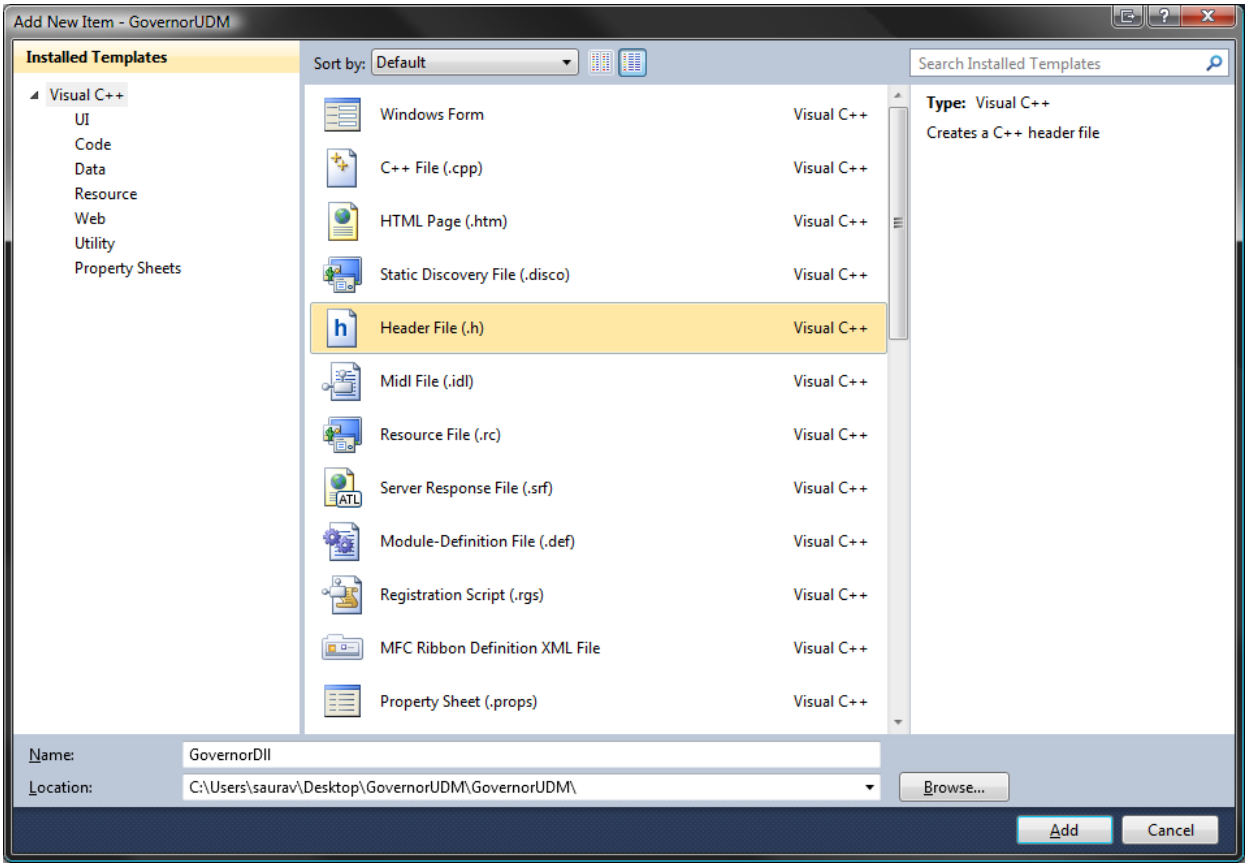
4. Select DLL, place a check mark at Empty Project, and click Finish.



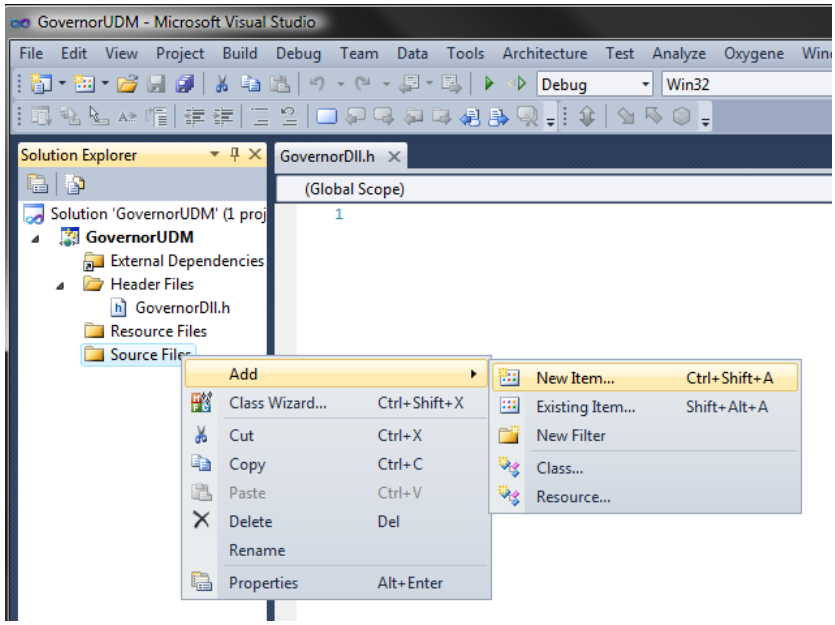
5. In the left pane, right click on Header Files, and navigate to Add > New Item…



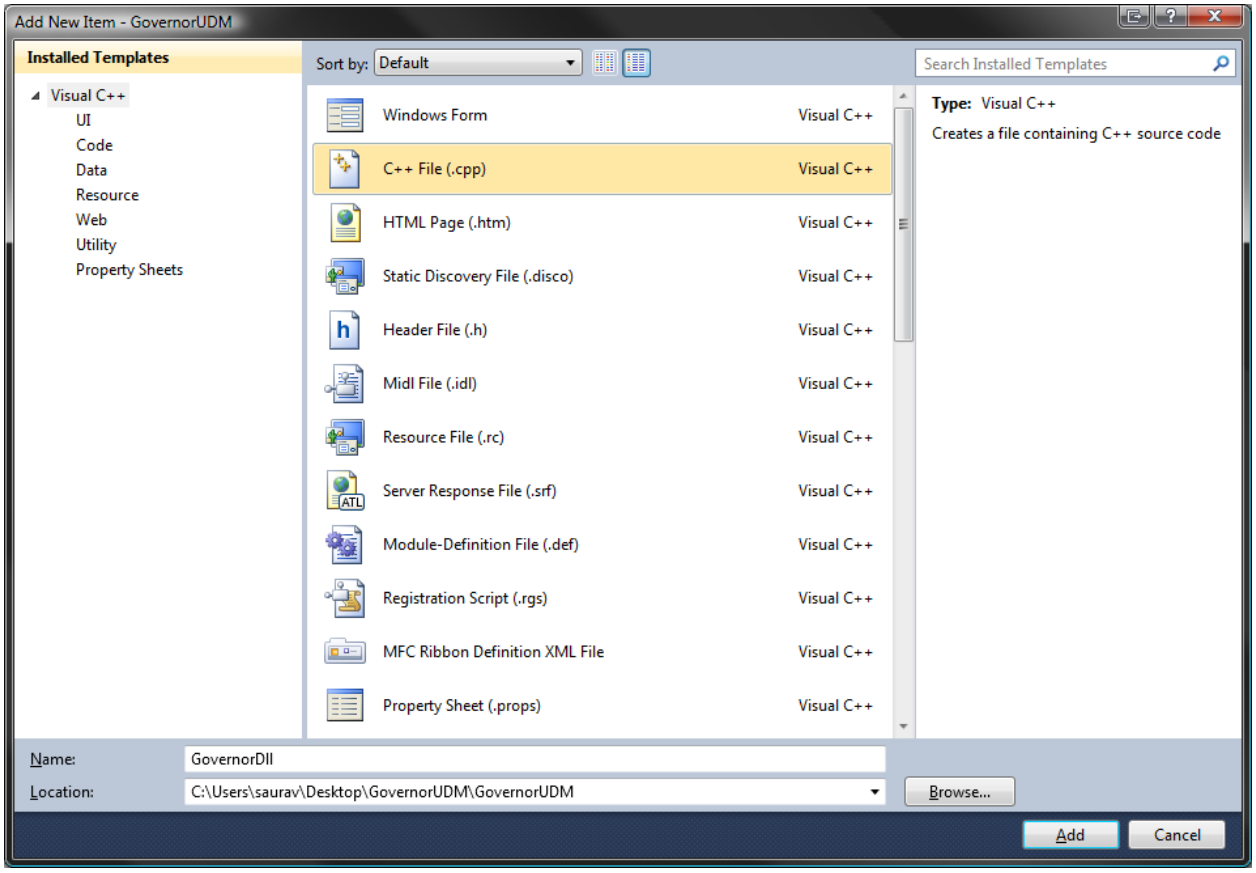
6. Choose Header File (.h), name it GovernorDll, and click OK.



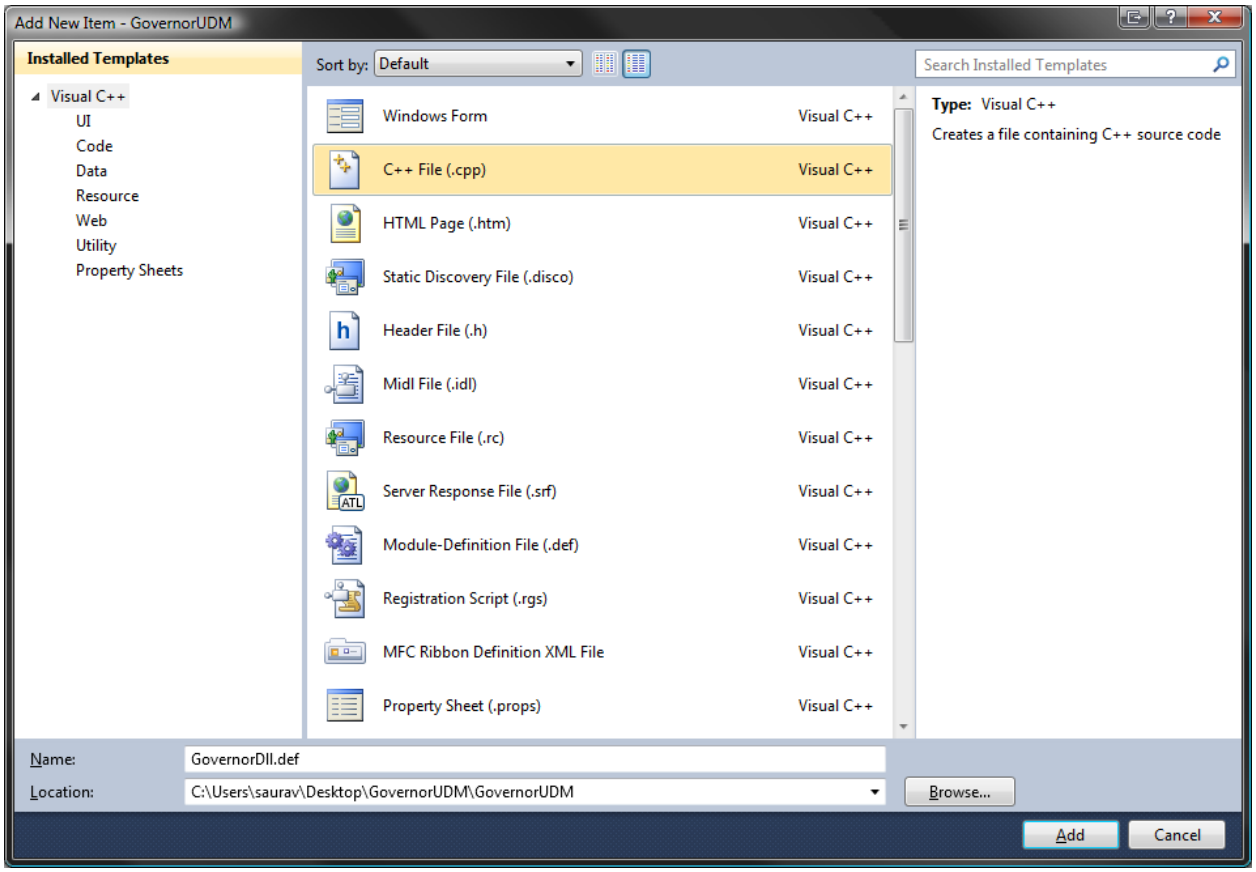
7. In the left pane, right click on Source Files, and navigate to Add > New Item…



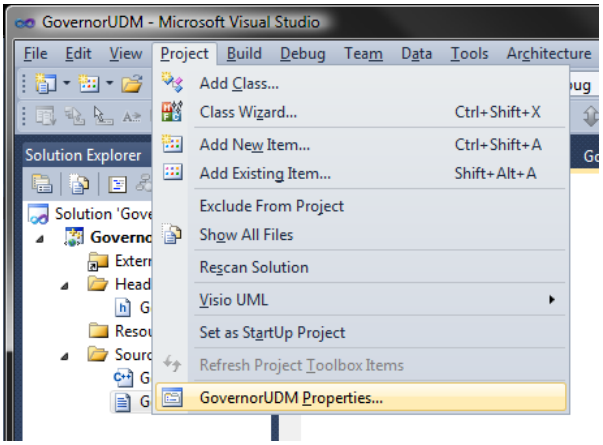
8. Choose C++ File (.cpp), name it GovernorDll, and click OK.



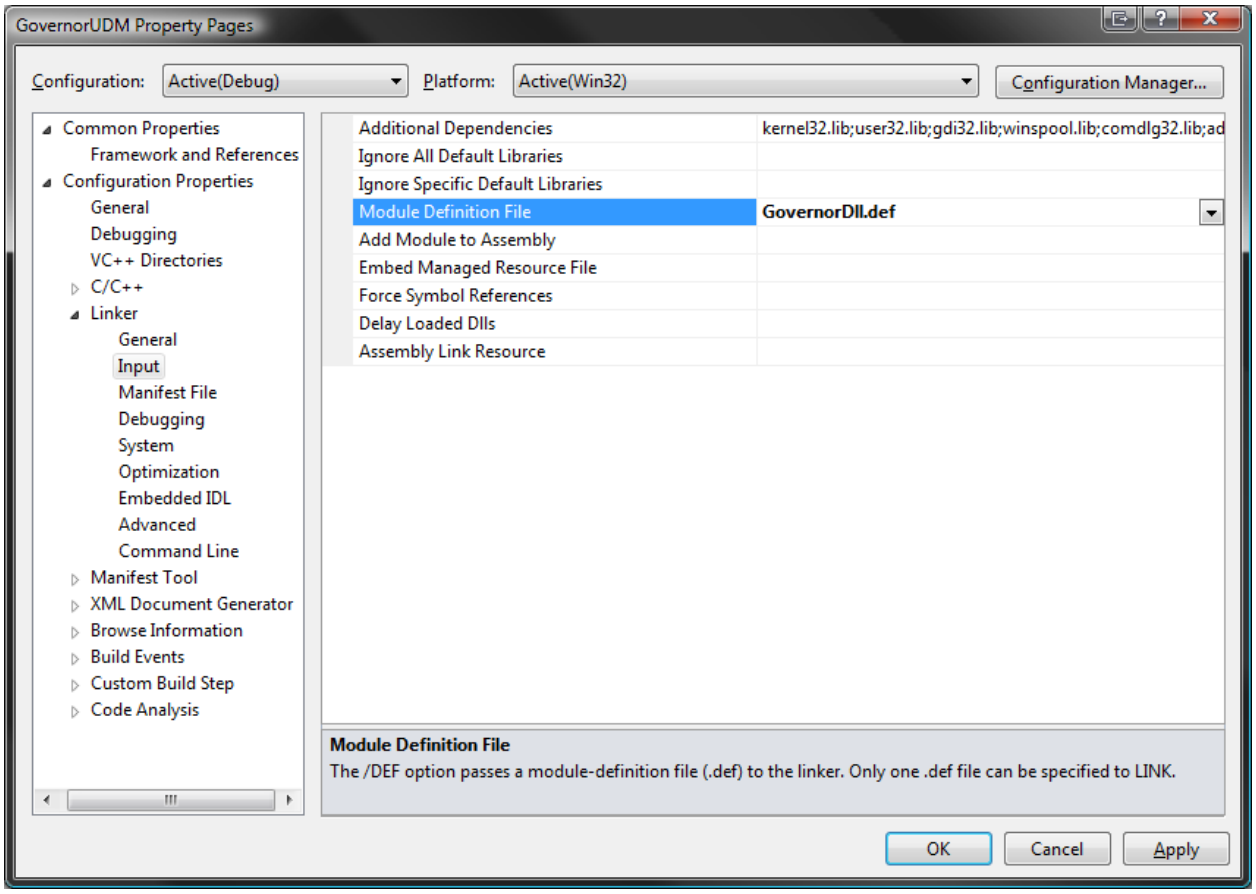
9. Repeat step 7. Choose C++ File (.cpp), name it GovernorDll.def, and click OK.



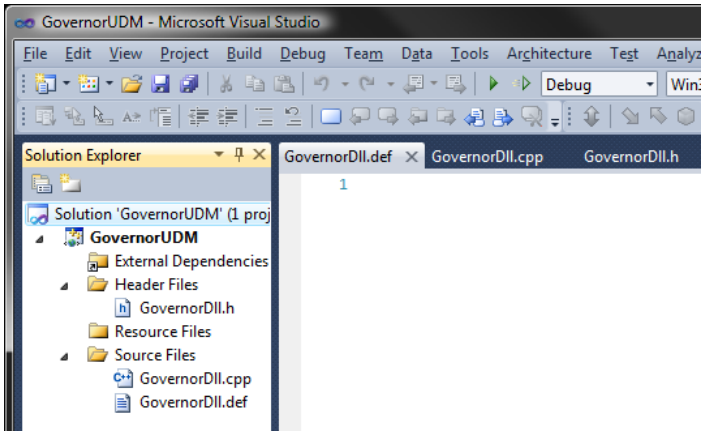
10. Navigate to Project > GovernorUDM Properties…



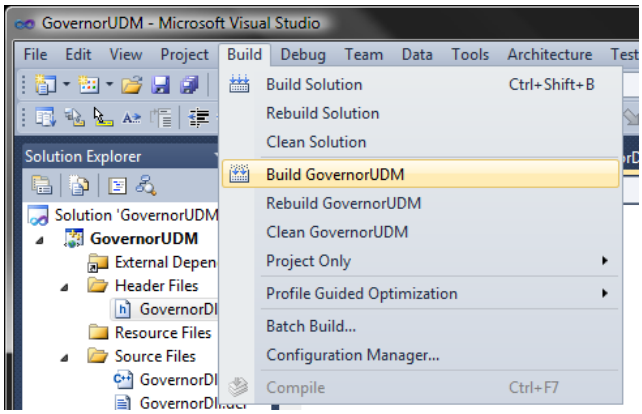
11. For the .dll file to have the appropriately named functions in its export directory, C++ linker needs to be told the exact names of these functions. Navigate to Configuration Properties > Linker > Input. In the Module Definition File box, type GovernorDll.def. Click OK.



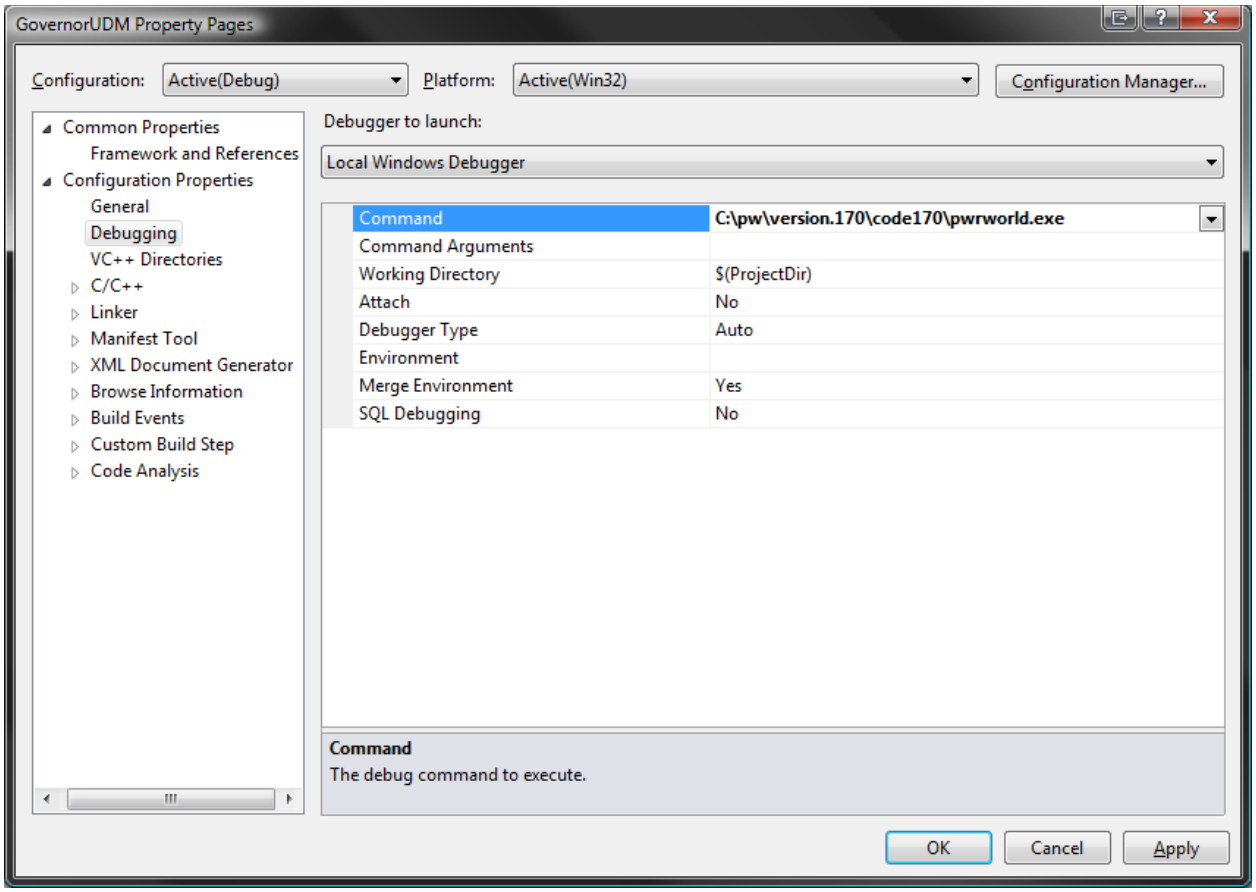
12. The project is now ready to use. Refer to the template, and sample models written in C++ to populate the contents of this project to build a user-defined governor model of your choice.



13. Once the content has been populated, the project can be built by navigating to Build > Build GovernorUDM.



14. Visual Studio provides an option to debug the .dll file created from the C++ side. To set this up, repeat step 10. Navigate to Configuration Properties > Debugging. In the Command box, populate the full path of the pwrworld.exe file. Click OK. This will allow the project to get attached to the pwrworld.exe process and breakpoints can thereafter be set within this project.



Preliminaries

Type definitions and pointers

Structure of integers

```
struct TStructureOne { // name of this structure type
    int nIntegerOne; // name of the first element of this structure type
    …
    int nIntegerN; // name of the nth element of this structure type
};
```

Structure of pointers to array of floating point numbers

```
typedef double TDoubleArrayOne[j1]; // name of this array type with j1 elements
…
typedef double TDoubleArrayM[jM]; // name of this array type with jM elements

struct TStructureTwo { // name of this structure type
    TDoubleArrayOne* /*double**/ DoublesOne;
    // name of the first element of this structure type
    …
    TDoubleArrayM* /*double**/ DoublesM;
    // name of the mth element of this structure type
};
```

Array of floating point numbers

```
typedef double TDoubleArray[k]; // name of this array type with k elements
```

Function declarations

Exported functions

```
extern "C" <return type> __stdcall functionName1(<type> InArgA, … <type> InArgZ);

extern "C" void __stdcall functionName2(<type> InArgA, … <type> InArgZ);
```

Functions internal to the .dll file

```
<return type> functionName3(<type> InArgA, … <type> InArgZ);

void functionName4(<type> InArgA, … <type> InArgZ);
```

Function definitions

Exported functions

```
extern "C" <return type> __stdcall functionName1(<type> InArgA, … <type> InArgZ)
{
    // Declare variables here
    // Insert content here
}

extern "C" void __stdcall functionName2(<type> InArgA, … <type> InArgZ)
{
    // Declare variables here
    // Insert content here
}
```

Functions internal to the DLL file

```
<return type> functionName3(<type> InArgA, … <type> InArgZ)
{
```

```
    // Declare variables here
    // Insert content here
}

void functionName4(<type> InArgA, … <type> InArgZ)
{
    // Declare variables here
    // Insert content here
}
```

Dereferencing

Pointer to structure of integers

```
// Receiving
IntOne = (*StructureOne).nIntegerOne;
…
IntN = (*StructureOne).nIntegerN;

// Returning
(*StructureOne).nIntegerOne = IntOne;
…
(*StructureOne).nIntegerN = IntN;
```

Pointer to structure of pointers to array of floating point numbers

```
// Receiving
DoubOne1 = (*((*StructureTwo).DoublesOne))[0];
…
DoubMOnej1 = (*((*StructureTwo).DoublesOne))[(j1 - 1)];
…
DoubM1 = (*((*StructureTwo).DoublesM))[0];
…
DoubMjM = (*((*StructureTwo).DoublesM))[(jM - 1)];

// Returning
(*((*StructureTwo).DoublesOne))[0] = DoubOne1;
…
(*((*StructureTwo).DoublesOne))[(j1 - 1)] = DoubMOnej1;
…
(*((*StructureTwo).DoublesM))[0] = DoubM1;
…
(*((*StructureTwo).DoublesM))[(jM - 1)] = DoubMjM;
```

Pointer to array of floating point numbers

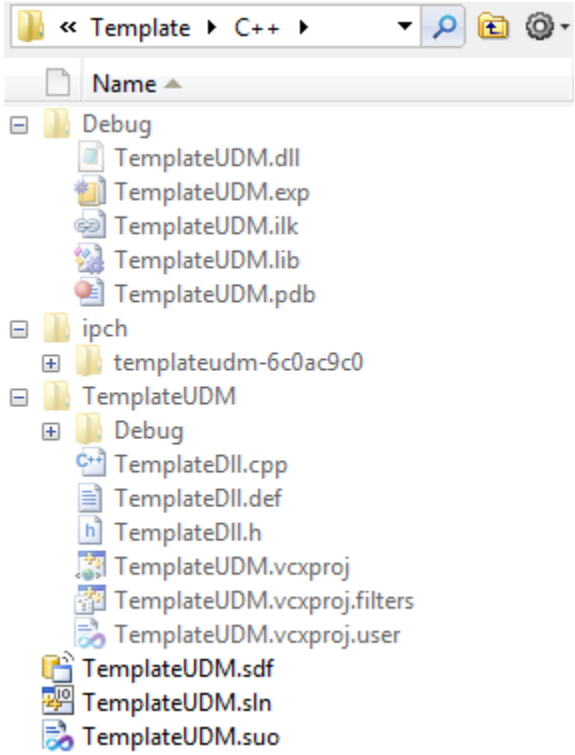
```
// Receiving
Doub1 = (*DoubleArray)[0];
…
Doubk = (*DoubleArray)[(k - 1)];

// Returning
(*DoubleArray)[0] = Doub1;
…
(*DoubleArray)[(k - 1)] = Doubk;
```

Making functions available in the export directory of the .dll file

```
LIBRARY <DllName> ; This is the name of the .dll file that will be produced
EXPORTS ; semi-colons are used for comments in .def file
    functionName1 @1 ; @<number> removes decorations
```

```
functionName2 @2 ; one line for each function name
functionName3 @3
functionName3 @4
```

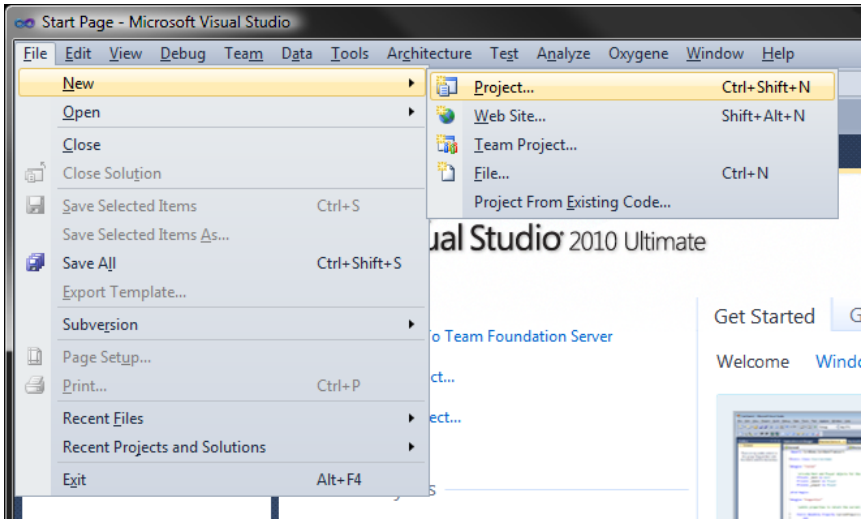
Project file/directory structure



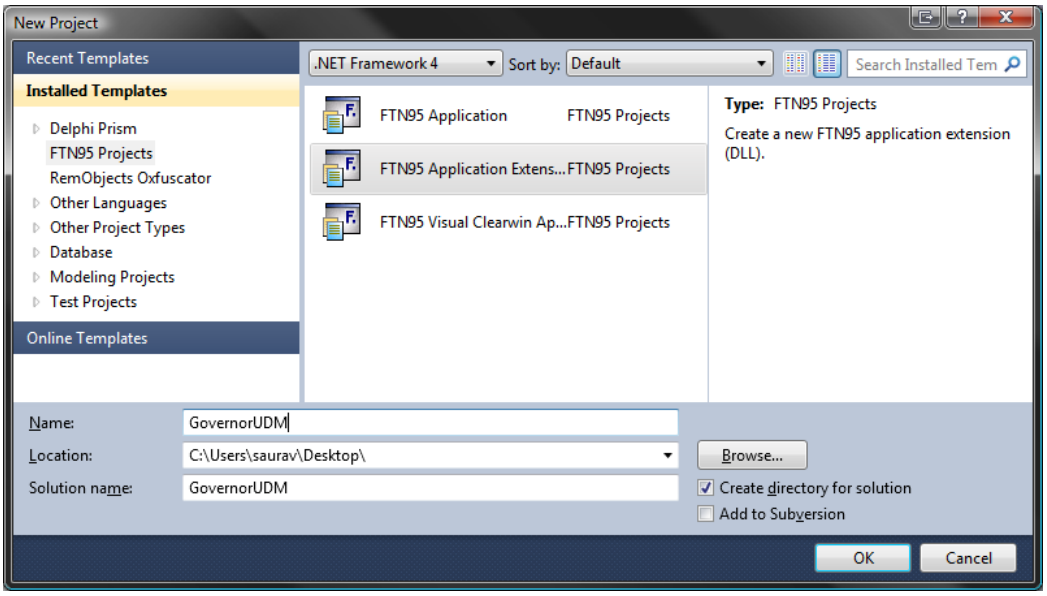
3. Fortran (Microsoft® Visual Studio 2010 with Silverfrost FTN95 plug-in)

Getting Started

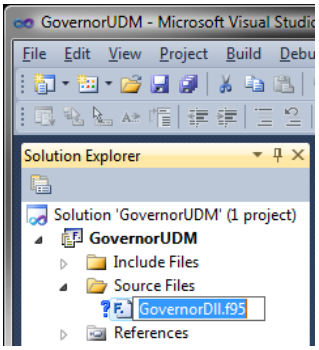
1. Start Visual Studio, and navigate to File > New > Project…



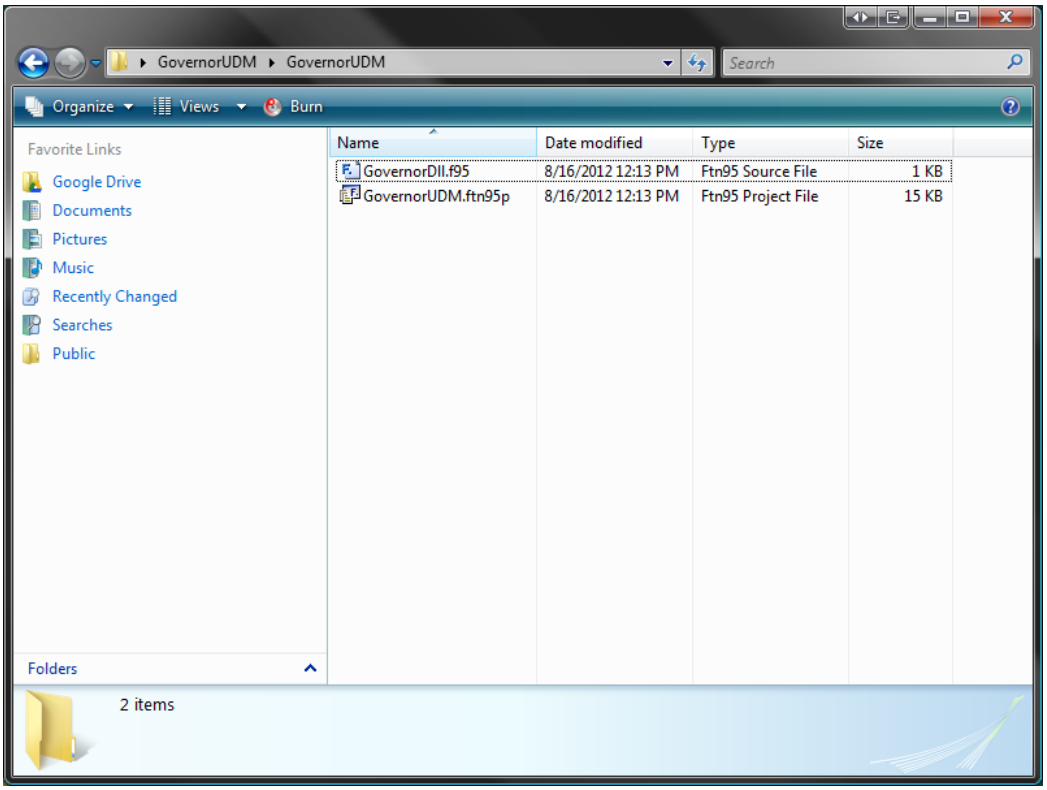
2. From Installed Templates, navigate to FTN95 Projects. Choose FTN95 Application Extension. Set Desktop as the Location. Type GovernorUDM in the Name, and Solution name boxes. This is the name of the .dll file that will be created later, i.e., GovernorUDM.dll will be created when this project is eventually built. Ensure Create directory for solution is checked, and Click OK.



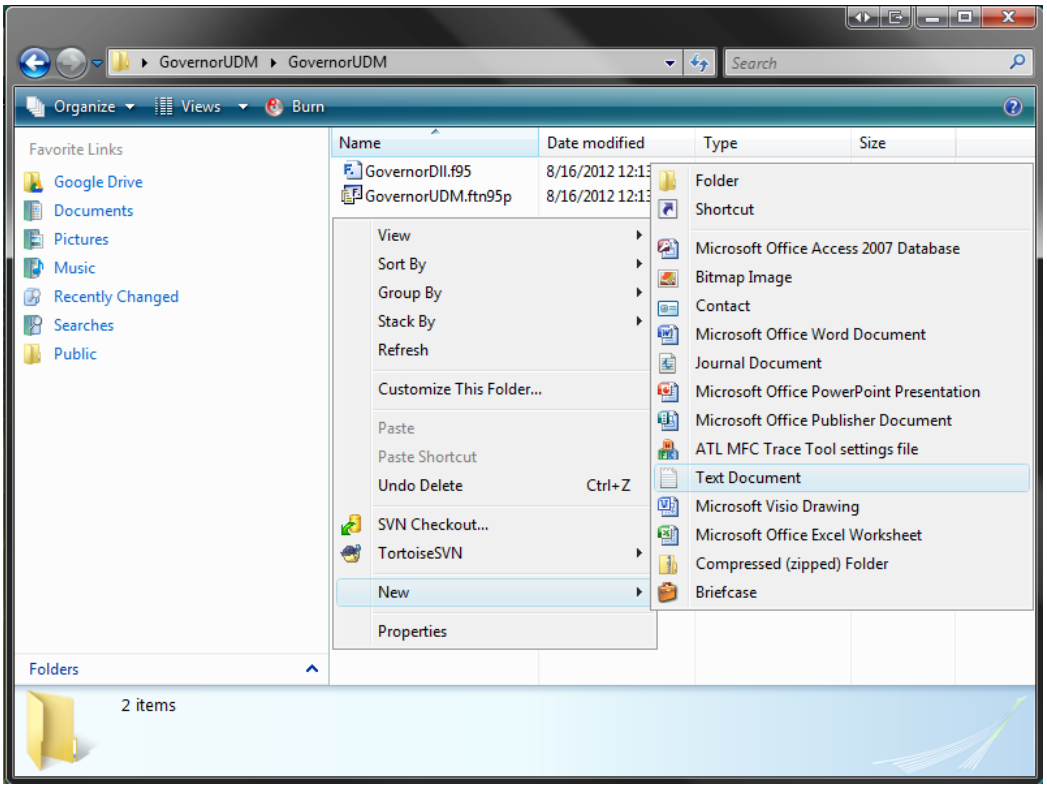
3. Visual Studio will automatically load a template. In the left pane, navigate to GovernorUDM > Source Files. Rename the file to GovernorDll.ftn95.



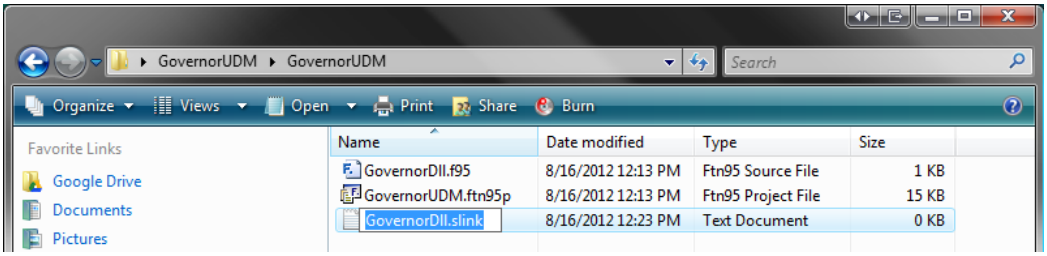
4. Using Windows Explorer, navigate to the Desktop, and then to GovernorUDM\GovernorUDM. The following two files should be visible in this directory.



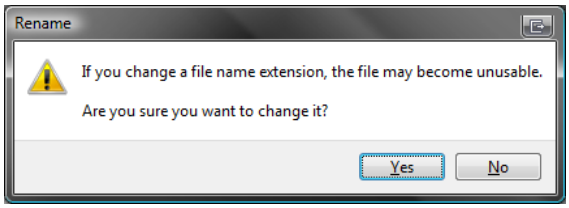
5. Right click in the directory, and navigate to New > Text Document.



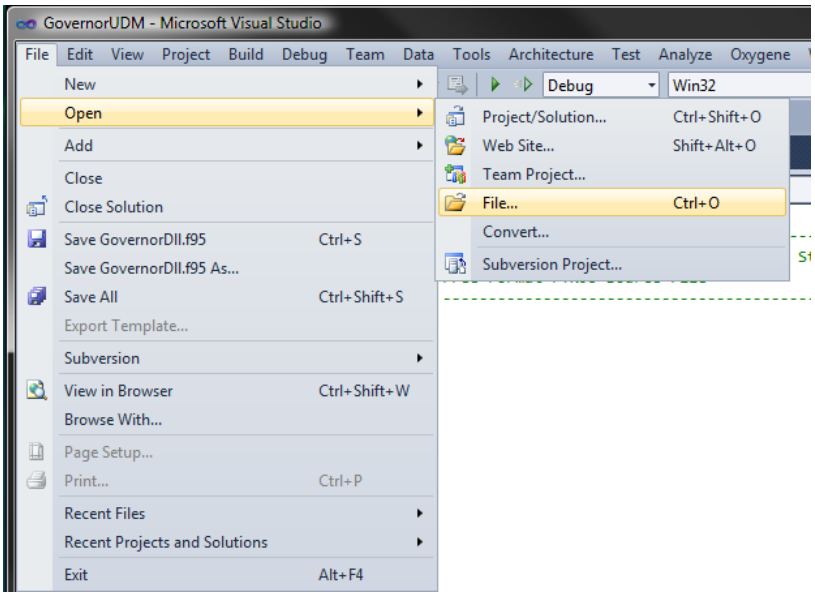
6. Rename the .txt file to GovernorDll.slink, and press Enter on the keyboard.



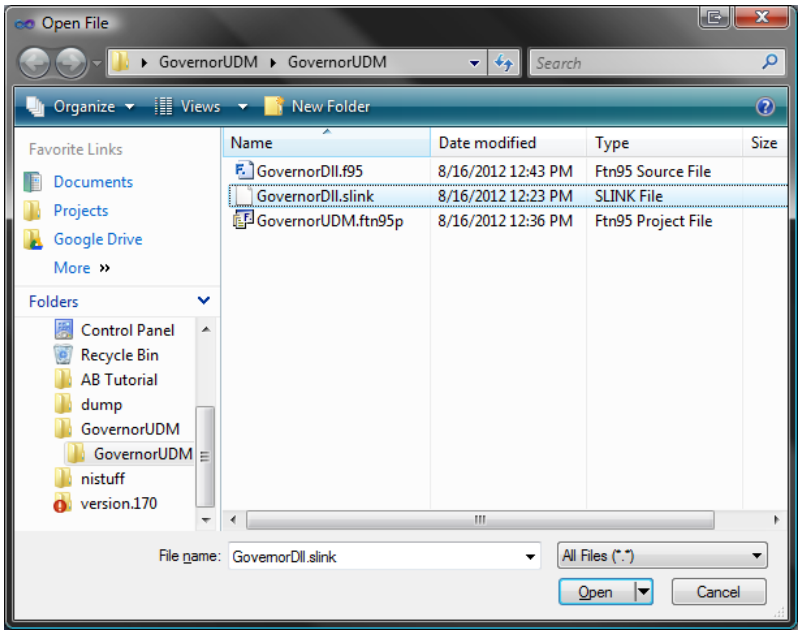
7. Ignore the warning, and click Yes.



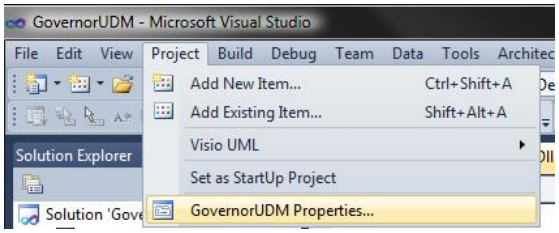
8. Return to Visual Studio, and navigate to File > Open > File…



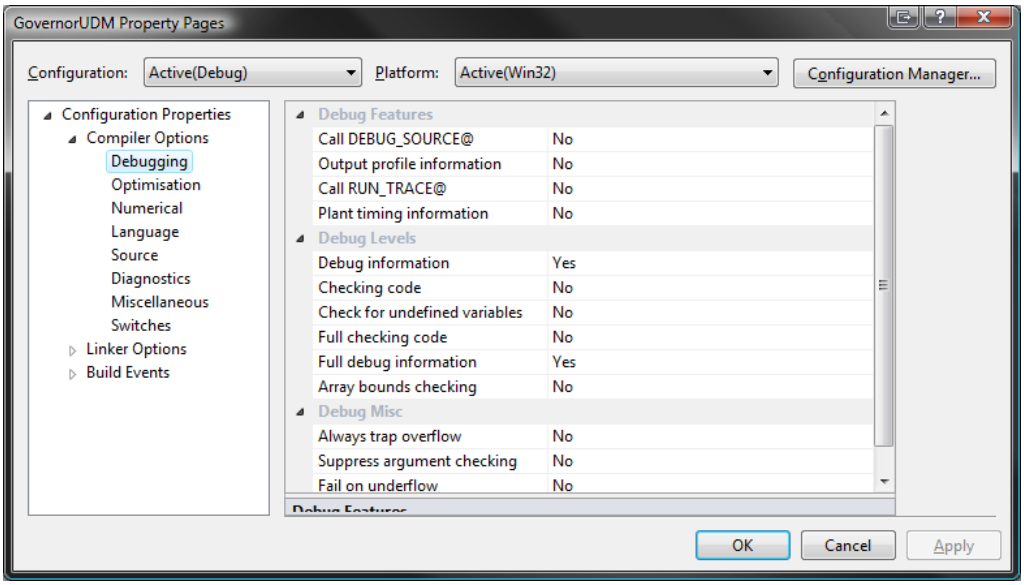
9. Navigate to the Desktop, and then to GovernorUDM\GovernorUDM. Select GovernorDll.slink, and click Open. Now, this .slink file is open in the same window, but is not added to the project yet. This will be done later in step 11.

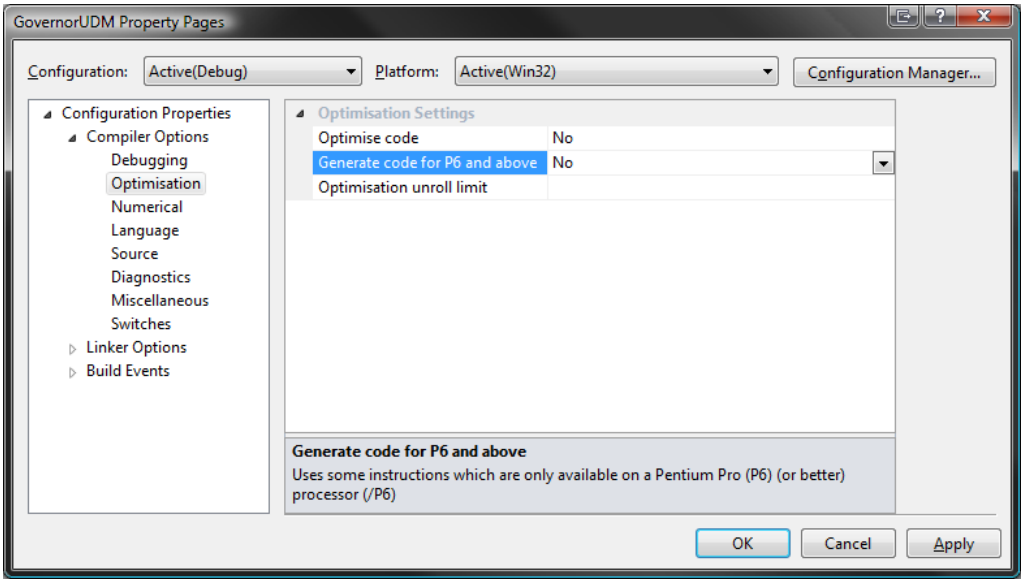


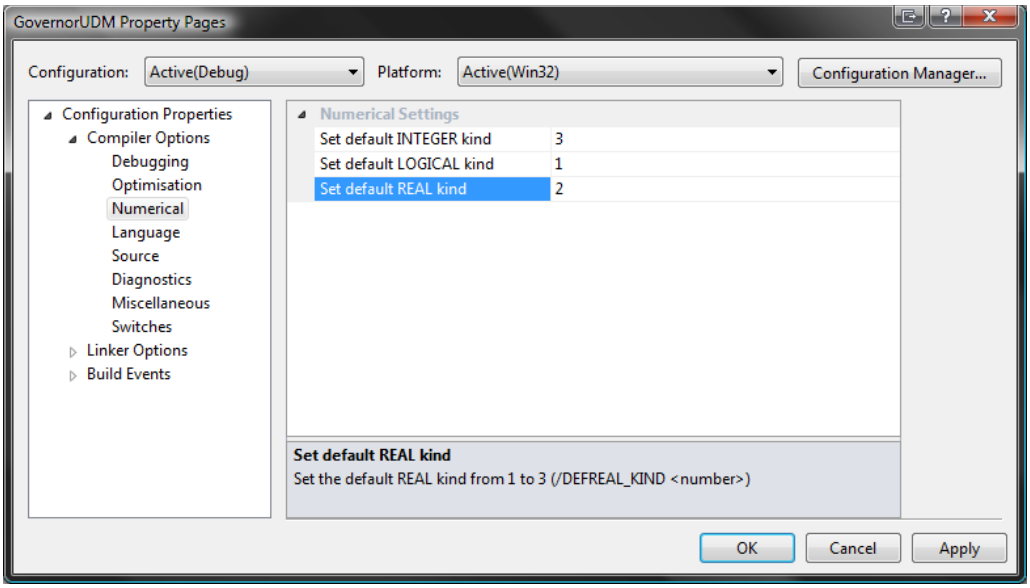
10. Navigate to Project > GovernorUDM Properties…

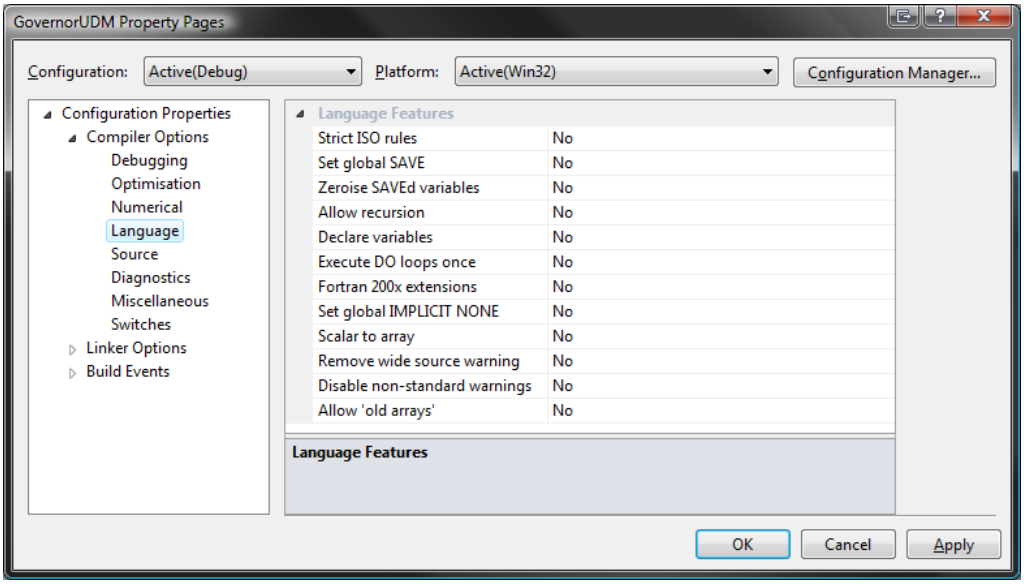


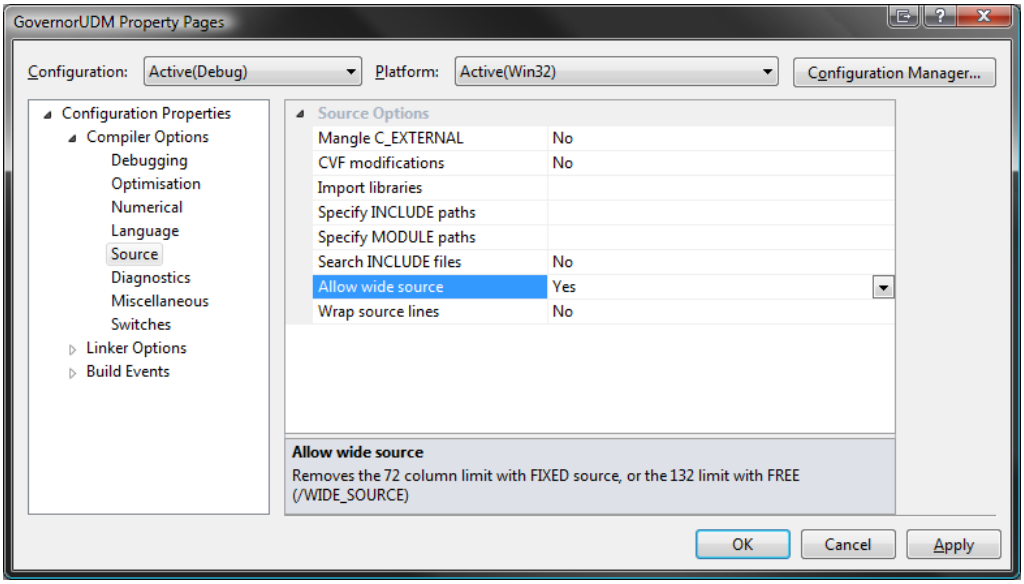
11. Navigate to Configuration Properties > Compiler Options, and go through each of the options, starting with Debugging, through Switches. Then, the ones in Linker Options, followed by Build Events. Ensure all options are set as shown in the screen captures below.

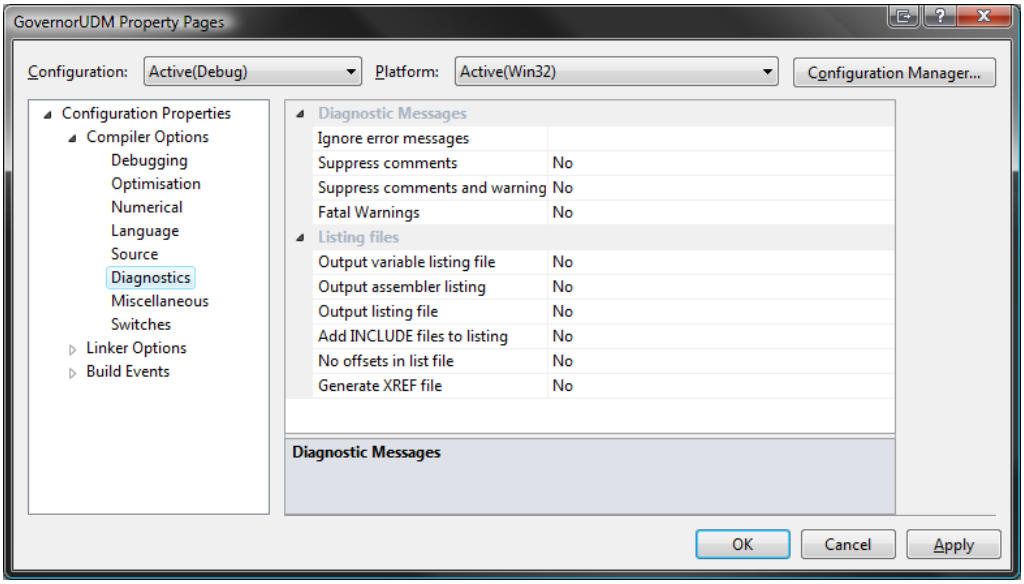


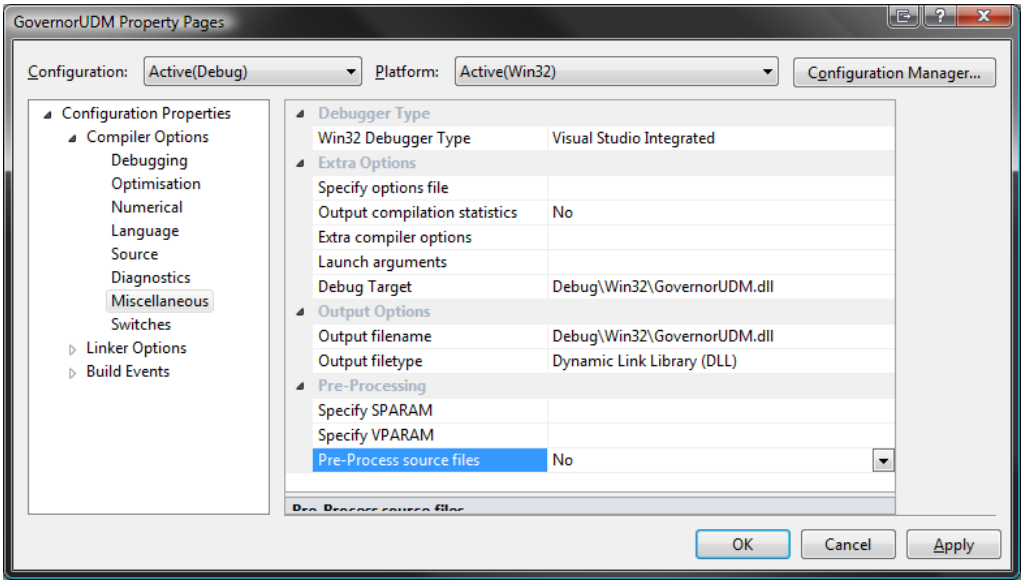


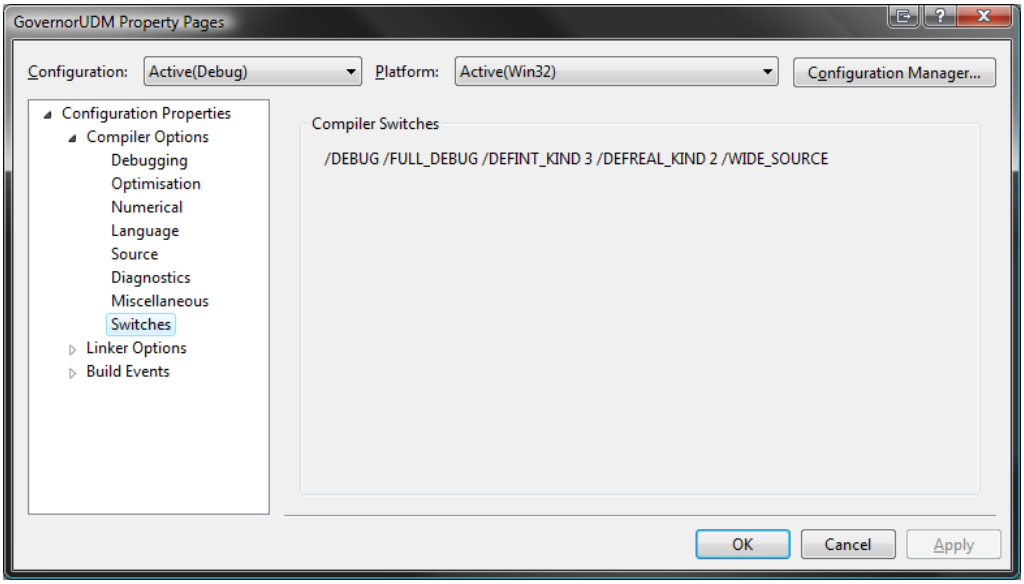

GovernorUDM Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Configuration Properties
Compiler Options
Debugging
Optimisation
Numerical
Language
Source
Diagnostics
Miscellaneous
Switches
Linker Options
Build Events
Numerical Settings
Set default INTEGER kind 3
Set default LOGICAL kind 1
Set default REAL kind 2
Set default REAL kind
Set the default REAL kind from 1 to 3 (/DEFREAL_KIND <number>)
OK
Cancel
Apply

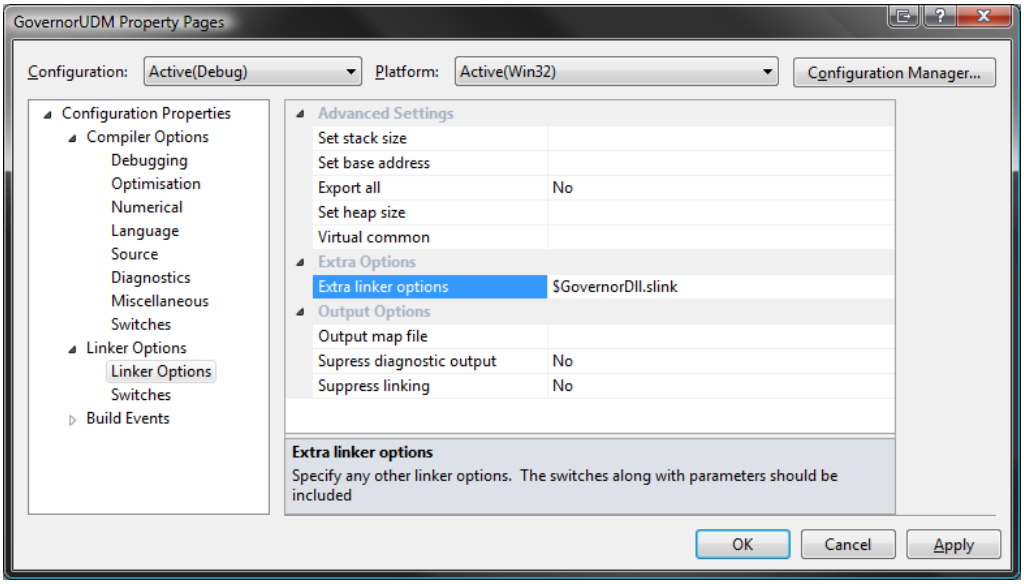

GovernorUDM Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Configuration Properties
Compiler Options
Debugging
Optimisation
Numerical
Language
Source
Diagnostics
Miscellaneous
Switches
Linker Options
Build Events
Language Features
Strict ISO rules No
Set global SAVE No
Zeroise SAVEd variables No
Allow recursion No
Declare variables No
Execute DO loops once No
Fortran 200x extensions No
Set global IMPLICIT NONE No
Scalar to array No
Remove wide source warning No
Disable non-standard warnings No
Allow 'old arrays' No
Language Features
OK
Cancel
Apply

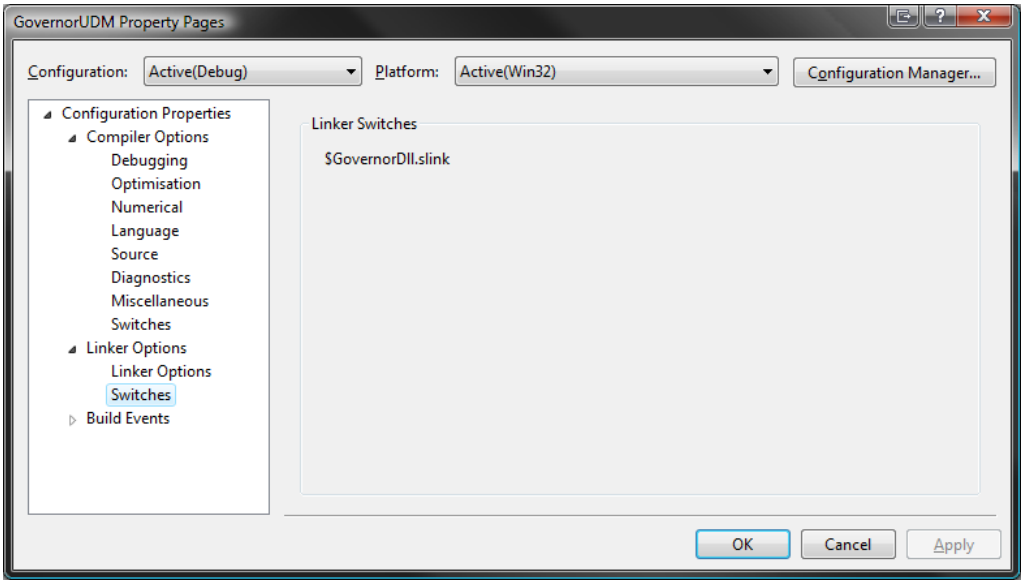

GovernorUDM Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Configuration Properties
Compiler Options
Debugging
Optimisation
Numerical
Language
Source
Diagnostics
Miscellaneous
Switches
Linker Options
Build Events
Source Options
Mangle C_EXTERNAL No
CVF modifications No
Import libraries
Specify INCLUDE paths
Specify MODULE paths
Search INCLUDE files No
Allow wide source Yes
Wrap source lines No
Allow wide source
Removes the 72 column limit with FIXED source, or the 132 limit with FREE (/WIDE_SOURCE)
OK
Cancel
Apply

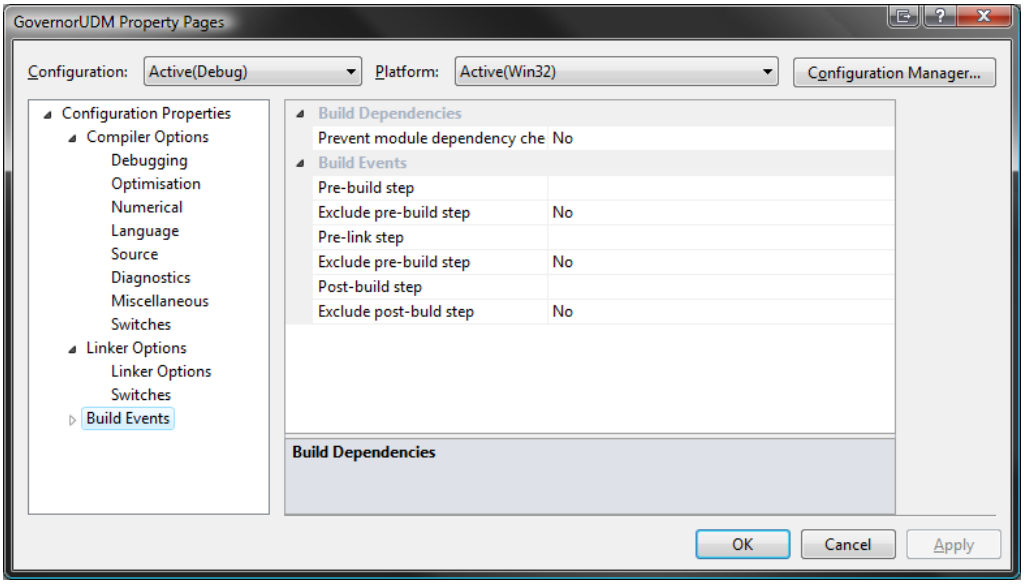

GovernorUDM Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Configuration Properties
Compiler Options
Debugging
Optimisation
Numerical
Language
Source
Diagnostics
Miscellaneous
Switches
Linker Options
Build Events
Diagnostic Messages
Ignore error messages
Suppress comments No
Suppress comments and warning No
Fatal Warnings No
Listing files
Output variable listing file No
Output assembler listing No
Output listing file No
Add INCLUDE files to listing No
No offsets in list file No
Generate XREF file No
Diagnostic Messages
OK
Cancel
Apply


GovernorUDM Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Configuration Properties
Compiler Options
Debugging
Optimisation
Numerical
Language
Source
Diagnostics
Miscellaneous
Switches
Linker Options
Build Events
Debugger Type
Win32 Debugger Type
Visual Studio Integrated
Extra Options
Specify options file
Output compilation statistics
No
Extra compiler options
Launch arguments
Debug Target
Debug\Win32\GovernorUDM.dll
Output Options
Output filename
Debug\Win32\GovernorUDM.dll
Output filetype
Dynamic Link Library (DLL)
Pre-Processing
Specify SPARAM
Specify VPARAM
Pre-Process source files
No
OK
Cancel
Apply

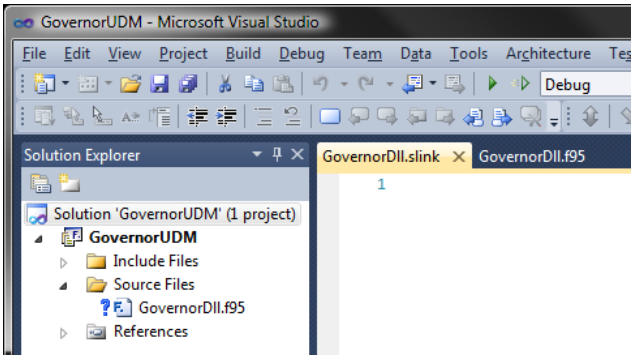

GovernorUDM Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Configuration Properties
Compiler Options
Debugging
Optimisation
Numerical
Language
Source
Diagnostics
Miscellaneous
Switches
Linker Options
Build Events
Compiler Switches
/DEBUG /FULL_DEBUG /DEFINT_KIND 3 /DEFREAL_KIND 2 /WIDE_SOURCE
OK
Cancel
Apply


GovernorUDM Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Configuration Properties
Compiler Options
Debugging
Optimisation
Numerical
Language
Source
Diagnostics
Miscellaneous
Switches
Linker Options
Linker Options
Switches
Build Events
Advanced Settings
Set stack size
Set base address
Export all
No
Set heap size
Virtual common
Extra Options
Extra linker options
$GovernorDll.slink
Output Options
Output map file
Supress diagnostic output
No
Suppress linking
No
Extra linker options
Specify any other linker options. The switches along with parameters should be included
OK
Cancel
Apply

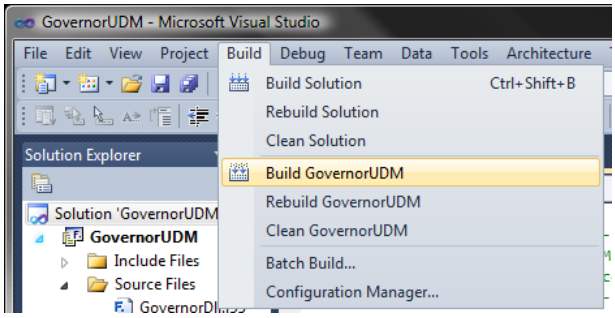

GovernorUDM Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Configuration Properties
Compiler Options
Debugging
Optimisation
Numerical
Language
Source
Diagnostics
Miscellaneous
Switches
Linker Options
Linker Options
Switches
Build Events
Linker Switches
$GovernorDll.slink
OK
Cancel
Apply



12. The project is now ready to use. Refer to the template, and sample models written in Fortran to populate the contents of this project to build a user-defined governor model of your choice.



13. Once the content has been populated, the project can be built by navigating to Build > Build GovernorUDM.



14. While it may be possible, a method for debugging on the Fortran side hadn't been successfully tested at the time this tutorial was written.

Preliminaries

Type definitions and pointers

Structure of integers

```
type :: TStructureOne ! name of this structure type
    integer :: nIntegerOne ! name of the first element of this structure type
    …
    integer :: nIntegerN ! name of the nth element of this structure type
end type
```

Structure of pointers to array of floating point numbers

```
type :: TDoubleArrayOne ! name of this array type with j1 elements
    real :: Array(j1)
end type
…
type :: TDoubleArrayM ! name of this array type with jM elements
    real :: Array(jM)
end type

type :: TStructureTwo ! name of this structure type
    sequence
    type(TDoubleArrayOne), pointer :: DoublesOne
    ! name of the first element of this structure type
    …
    type(TDoubleArrayM), pointer :: DoublesM
    ! name of the mth element of this structure type
end type
```

Note: For structures of mixed types, attention needs to be given byte alignment and sequence of elements in these structures. The keywords `align` and `sequence` could be utilized.

Array of floating point numbers

```
type TDoubleArray ! name of this array type with k elements
    real :: Array(k)
end type
```

Function definitions

Exported functions

```
f_stdcall <return type> function functionName1(ArgInA, … ArgInZ) result(ArgOut)
    use Header
    <type> :: ArgInA
    …
    <type> :: ArgInZ
    ! Declare variables here
    ! Insert content here
end function

f_stdcall subroutine subroutineName1(ArgInA, … ArgInZ)
    use Header
    <type> :: ArgInA
    …
    <type> :: ArgInZ
    ! Declare variables here
    ! Insert content here
end subroutine
```

Functions internal to the .dll file

```
<return type> function functionName2(ArgInA, … ArgInZ) result(ArgOut)
    use Header
    <type> :: ArgInA
    …
    <type> :: ArgInZ
    ! Declare variables here
    ! Insert content here
end function

subroutine subroutineName2(ArgInA, … ArgInZ)
    use Header
    <type> :: ArgInA
    …
    <type> :: ArgInZ
    ! Declare variables here
    ! Insert content here
end subroutine
```

Dereferencing

Pointer to structure of integers

```
! Receiving
IntOne = StructureOne%nIntegerOne
…
IntN = StructureOne%nIntegerN

! Returning
StructureOne%nIntegerOne = IntOne
…
StructureOne%nIntegerN = IntN
```

Pointer to structure of pointers to array of floating point numbers

```
! Receiving
DoubOne1 = StructureTwo%DoublesOne%Array(1)
…
DoubMOnej1 = StructureTwo%DoublesOne%Array(j1)
…
DoubM1 = StructureTwo%DoublesM%Array(1)
…
DoubMjM = StructureTwo%DoublesM%Array(jM)

! Returning
StructureTwo%DoublesOne%Array(1) = DoubOne1
…
StructureTwo%DoublesOne%Array(j1) = DoubMOnej1
…
StructureTwo%DoublesM%Array(1) = DoubM1
…
StructureTwo%DoublesM%Array(jM) = DoubMjM
```

Pointer to array of floating point numbers

```
! Receiving
Doub1 = DoubleArray%Array(1)
…
Doubk = DoubleArray%Array(k)
```

```
! Returning
DoubleArray%Array(0) = Doub1
…
DoubleArray%Array(k) = Doubk
```

Making functions available in the export directory of the .dll file

```
// double-slashes are used for comments in .slink file
// LHS denotes the name with which the function will be available in the export directory
of the dll
// RHS denotes the name with which the compiler automatically creates the functions, the
Salford compiler for FTN95 makes them allCaps by default
export functionName1 = FUNCTIONNAME1
export subroutineName1 = SUBROUTINENAME1
export functionName2 = FUNCTIONNAME2
export subroutineName2 = SUBROUTINENAME2
```

Project file/directory structure