

Constitutive Model Plug-in for RS3

1. Introduction

In this version of RS3, users can define their own constitutive model and integrate the model into the program by using a dynamic-linking library (dll). The name of the dll should be RS3UDM.dll and needs to be placed in the installation folder of the program. The program will then detect and load the dll. One or more materials can be included in the dll. An example dll including two materials is included in the UDM folder in the RS3 installation folder. The details on how to create a dll are provided in the next section.

The dll should be written in C++ language although other languages can be used. However, the compatibility will not be warranted.

2. Programme Structure

2.1 Pre-requisite

All the headers files included in the UDM folder must be included in the dll in order to compile. (Algebra.h; Definition.h; GaussPoint.h;Material.h; Matsolid.h; MatUserDefined.h; mkl_cblas.h; mkl_lapackc.h; mkl_types.h;SaturationFunction.h and Tensors.h). The dll also needs to be linked with the RS3UDM.lib.

Note: RS3UDM.lib was compiled with the release configuration. If you compile with the debug configuration, you may not get the correct values for debugging.

The dll consists of classes for each material and several export functions. The material classes are inherited from the base class name CMatUserDefined. The export functions are created to provide the information of the material before the calculation is actually carried out.

2.2 Export Functions

The following functions must be provided in the dll. Function format and functionality will be explained together. Please note that those examples are taken from the example cpp file included in the installation.

1. Return number of materials in the dll:

```
extern "C" __declspec(dllexport) int GetNumberOfMaterial()
{
    return 2;
}
```

This function returns the number of materials in the dll. Since we have two materials, it should return

2.

2. Return names of the materials in the dll:

```
extern "C" __declspec(dllexport) char* GetMaterialName(int id)
{
    char *matName;
    if (id == 0) {matName = "Elastic";}
    else if (id == 1) {matName = "Mohr";}
}
```

```

else
    matName = "No material was found";
return matName;
}

```

This function returns the name of the material based on the input *id*. In this example the function will return Elastic if *id* equals to 0 and Mohr if *id* equal to 1.

Note: The user should provide the *id* starting from 0 as the program uses C index style.

3. Return number of parameters in the constitutive model:

```

extern "C" __declspec(dllexport) int GetNumberOfConstitutiveParameter(std::string
matName)
{
    int nVars = 0;
    if (matName == "Elastic") {nVars = 2;}
    else if (matName == "Mohr") {nVars = 5;}
    return nVars;
}

```

This function returns the number of the parameters of the materials based on the input *matName*. The user should return the number of parameters used in the material. In the Elastic material, there are only two parameters (Young modulus and Poisson's ratio) and in the Mohr material, there are five parameters (Young modulus, Poisson's ratio, friction angle, cohesion and dilation angle).

4. Return the name of the parameters in the constitutive model:

```

extern "C" __declspec(dllexport) char* GetNameOfConstitutiveParameter(std::string
matName, int paramId)
{
    char* paramName;
    if (matName == "Elastic") {
        switch (paramId){
            case 0:
                paramName = "Young_modulus";
                break;
            case 1:
                paramName = "Poisson_ratio";
                break;
            default:
                paramName = "No param found";
                break;
        }
    }
    else if (matName == "Mohr") {
        switch (paramId){
            case 0:
                paramName = "Young_modulus";
                break;
            case 1:
                paramName = "Poisson_ratio";
                break;
        }
    }
}

```

```

        case 2:
            paramName = "Cohesion";
            break;
        case 3:
            paramName = "Friction_Angle";
            break;
        case 4:
            paramName = "Dilation_Angle";
            break;
        default:
            paramName = "No param found";
            break;
    }
}
else
    paramName = "No material was found";
return paramName;
}

```

This function returns the name of the parameters of the materials based on the two input *matName* and *paramId*. *matName* is one of the name of the material that the user provided in the *GetMaterialName()* function. Similar to other id, *paramId* also needs to be started at 0 as C index style.

5. Return a pointer to the material class

```

extern "C" __declspec(dllexport) CMatSolid* GetMaterial(size_t id, std::string
names)
{
    CMatSolid* mat;
    if (names=="Elastic"){
        mat = new Elastic(id);
    }
    if (names=="Mohr"){
        mat = new Mohr(id);
    }
    return mat;
}

```

This function returns the pointer of the class based on the input id and names of the materials. The input *id* is the internal variable of RS3 and it should not be modified by the user. The variable *names* is one of the material names that are provided by the user.

2.3 Class Structures

The general ideas are that the programs will provide the incremental strains, previous stresses and strains, state variables and failure information and the user will provide the current stresses, state variables and failure information for every iteration. Those procedures will be carried out at the class scope.

Besides that main function, the user also needs to initialize specific parameters for the material such as the number of state variables, the material matrix is symmetric or not or the tangential stiffness is required for the material. Those parameters will be declared at the material constructors.

All constitutive models must inherit from the *CMatUserDefined* (material) class. The implementations are in the RS3UDM.lib. In addition to the material classes, the library also includes classes to implement points, vectors and matrices (Algebra.h), stress and strain tensors (Tensors.h).

Generally, the following functions are needed to be defined in the class:

1. *Elastic(size_t id);*

There are parameters inherit from the main class that needed to be declared for the materials:

bool *isTangential*: The material is calculated using the tangential stiffness or not. If set to true, the user needs to provide the plastic stiffness matrix.

bool *isSymmetric*: The global stiffness matrix is symmetric or not. It is needed for the program to determine if an unsymmetric global matrix is needed. However, even in the case the material matrix is not symmetric, a symmetric global one is still able to converge.

int *nStateVars*: number of the state variables. For example, the Mohr-Coulomb model may exhibit hardening or softening behaviour. In those cases, the failure envelope will depend on strain. Generally a scalar 'equivalent plastic strain' is used. Thus, the *nStateVars* should be set to 1 in those cases.

An example implementation is:

```
Elastic::Elastic(size_t id) : CMatUserDefined(id)
{
    isTangential = false;
    isSymmetric = true;
    modelMode = TUNNEL;
    nStateVars = 1;
}
```

2. *~Elastic(void);* Destructor of the class

3. *void ComputeElasticMatrix(CMatrix& Emat, const CStress3D stresses, const CStrain3D strains, const std::vector<double> stateVariables) const;*

This sets the input matrix *Emat* to the elastic stiffness matrix. The input matrix *Emat* has the size of 6x6. The *CMatrix* class is defined in *Algebra.h*

The other input is a reference to stresses and strains and state variables. This is necessary because the elastic stiffness matrix may depend on stress or some other state variable stored on the gauss point.

The size of the array *stateVariables* is *nStateVars* as declared in the constructors.

Storage for a matrix is also provided with the gauss point. This is basically for speed, if the user wishes to store some stiffness matrix instead of calculating it each time it is needed through *GetEMatrix3DPlastic*.

An example implementation is:

```
void Elastic::ComputeElasticMatrix(CMatrix& Emat, const CStress3D stresses, const
CStrain3D strains, const std::vector<double> stateVariables) const
{
    Emat.ZeroMatrix();

    double c44 = _dE / (2.0 + 2.0*_dNu);
    double c12 = _dE * _dNu / ((1.0 + _dNu)*(1.0 - 2.0*_dNu));
    double c11 = c12 + 2.0*c44;

    Emat(0,0) = c11;
    Emat(0,1) = c12;
    Emat(0,2) = c12;
    Emat(1,0) = c12;
    Emat(1,1) = c11;
    Emat(1,2) = c12;
    Emat(2,0) = c12;
    Emat(2,1) = c12;
    Emat(2,2) = c11;
    Emat(3,3) = c44;
    Emat(4,4) = c44;
    Emat(5,5) = c44;
}
```

4. **Void ComputePlasticMatrix(CMatrix& Emat, const CStress3D stresses, const CStrain3D strains, const std::vector<double> stateVariables) const;**

Similar to ComputeElasticMatrix() but it will be called when the *ERS3_FailureType** *failureInfo* is set to failure (RS3_NONE, RS3_SHEAR, RS3_TENSION, RS3_SHEAR_AND_TENSION, RS3_CRITICAL, RS3_COMPRESSION)

5. **virtual void UpdateGaussPointStress(CStress3D* stresses, CStrain3D* strains, std::vector<double>* stateVariables, ERS3_FailureType* failureInfo, const CStrain3D& deltaStrains) const;**

This is the main function that contains the relationship between the stresses and strains of the material. For the case of a classical elasto-perfect plastic Mohr coulomb material model this will include the following steps:

- Work out the elastic stress by taking the existing input stresses from the program and adding the elastic stiffness matrix times the input strain increment
- Check if stress is past the yield surface
- If not past yield, set the stress to the elastic stress and leave the function
- If past yield, bring stress back to the yield surface and set stresses to this value
- Set failure variables if failure has occurred.

An example implementation is:

```
void Elastic::UpdateGaussPointStress(CStress3D* stresses, CStrain3D* strains,
std::vector<double>* stateVariables, ERS3_FailureType* failureInfo, const
CStrain3D& deltaStrains) const
{
    if (stateVariables->size()==0) stateVariables->resize(6);
```

```

    CMatrix Emat(6,6);
    ComputeElasticMatrix(Emat,*stresses, *strains, *stateVariables);
    CVectorND stressIncr = Emat*deltaStrains.GetStrainVoigt();
    *stresses += stressIncr;
}

```

6. `void SetParameters(const std::vector<double>& parameters);`

This function is the interface that allows the main program to set the private constitutive model parameters in your class. The input is a vector of double-precision numbers (The `CVectorND` class is defined in the library – *Algebra.h*). The implementation of the function should use these values to set your constitutive model variables. It is very important that the order of the parameter output in the `GetNameOfConstitutiveParameter()` in section **Export Functions** matches the order of variables in this function.

An example implementation is:

```

void Elastic::SetParameters(const std::vector<double>& parameters)
{
    _dE = parameters[0];
    _dNu = parameters[1];
}

```

7. `void DeleteMaterial(void);`

This is necessary so that the material object can delete itself when it is no longer in use to prevent memory leaks. An example implementation is:

```

void Elastic::DeleteMaterial(void)
{
    delete this;
}

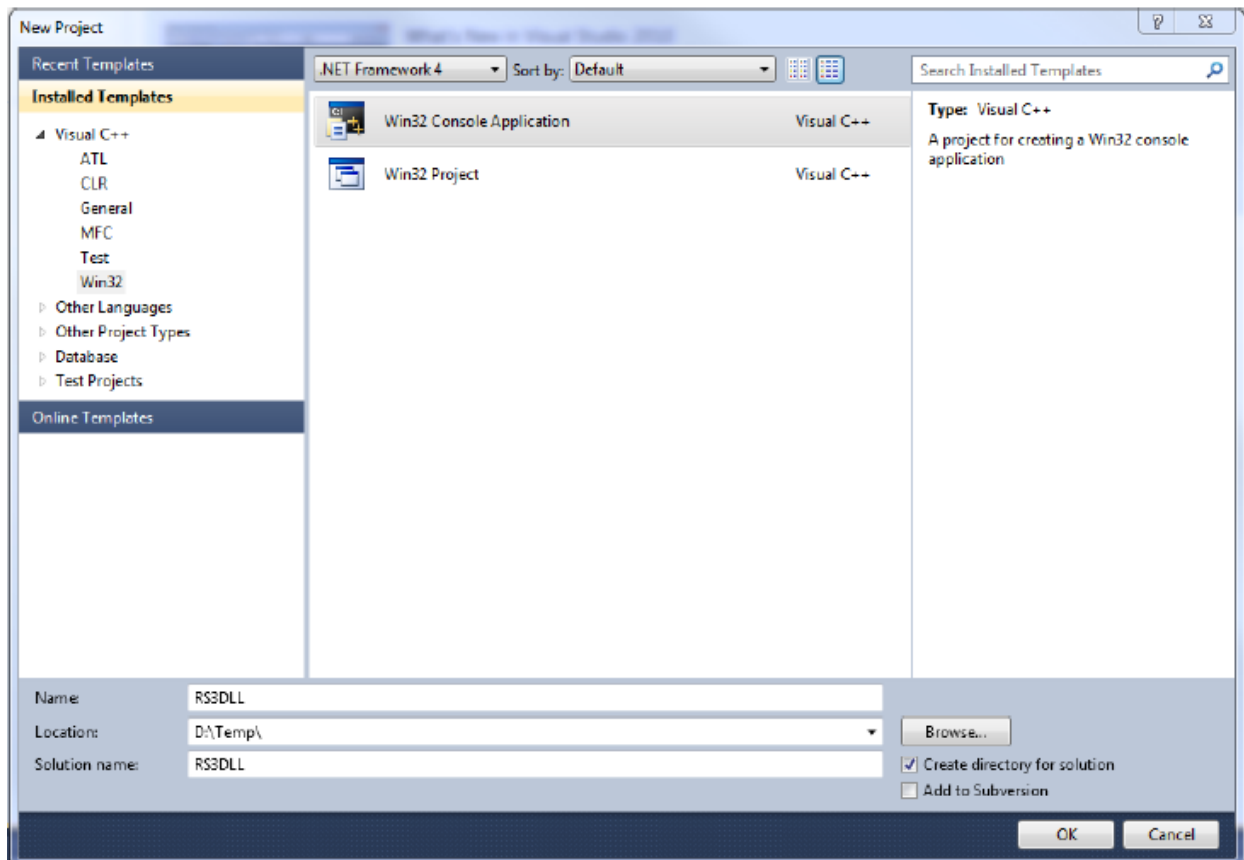
```

3. Predefined Math Functions from the RS3 Library

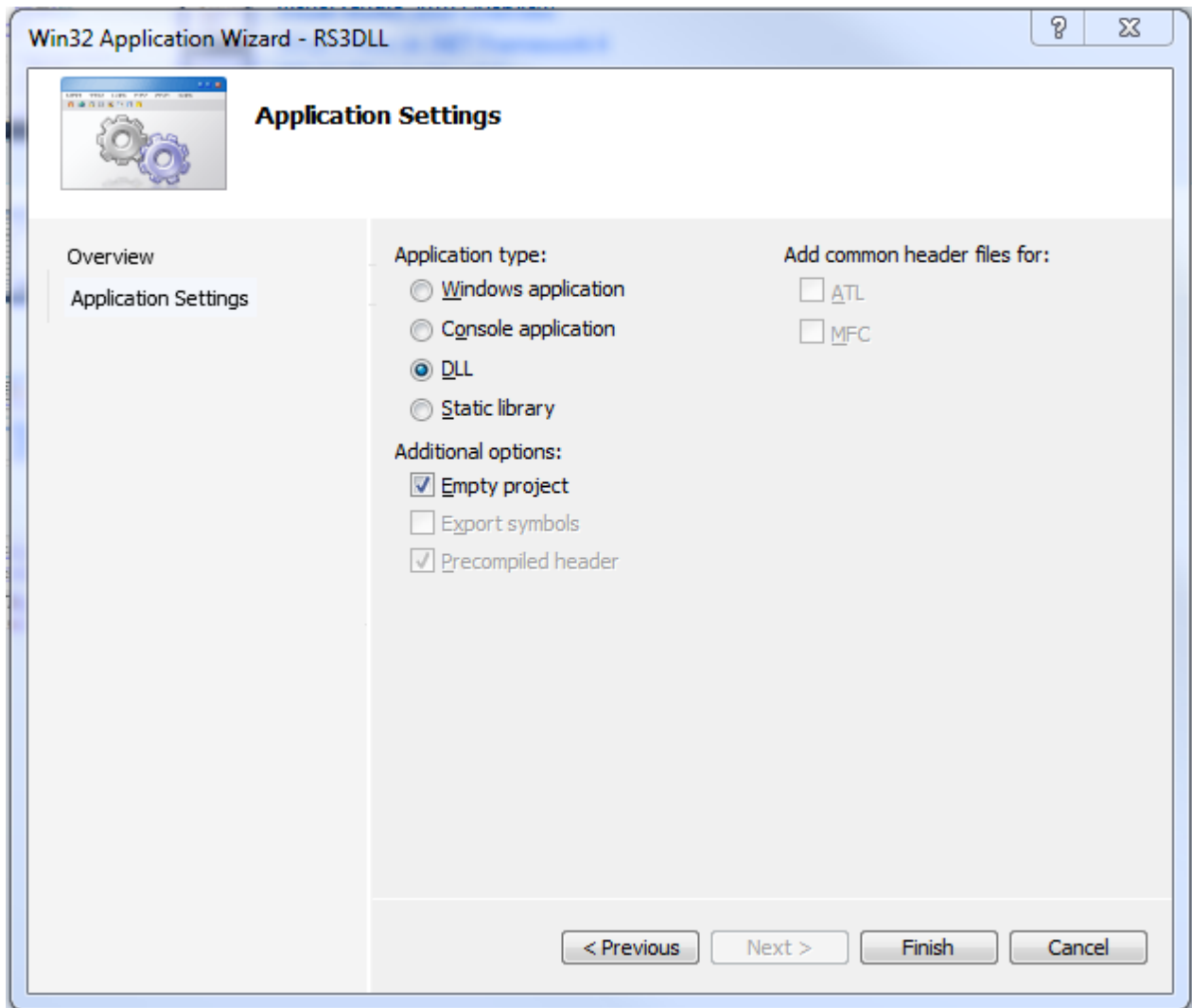
Refer to the `Tensor.h` and `Algebra.h` for the implementation and its functionality.

4. Create a DLL Project with Microsoft Visual Studio 2010

Select New Project. Under Visual C++ Projects, type in RS3DLL in the name and select OK.



Click on Application Settings and under Application type: select DLL. Under Additional Options: select Empty project. Click Finish.

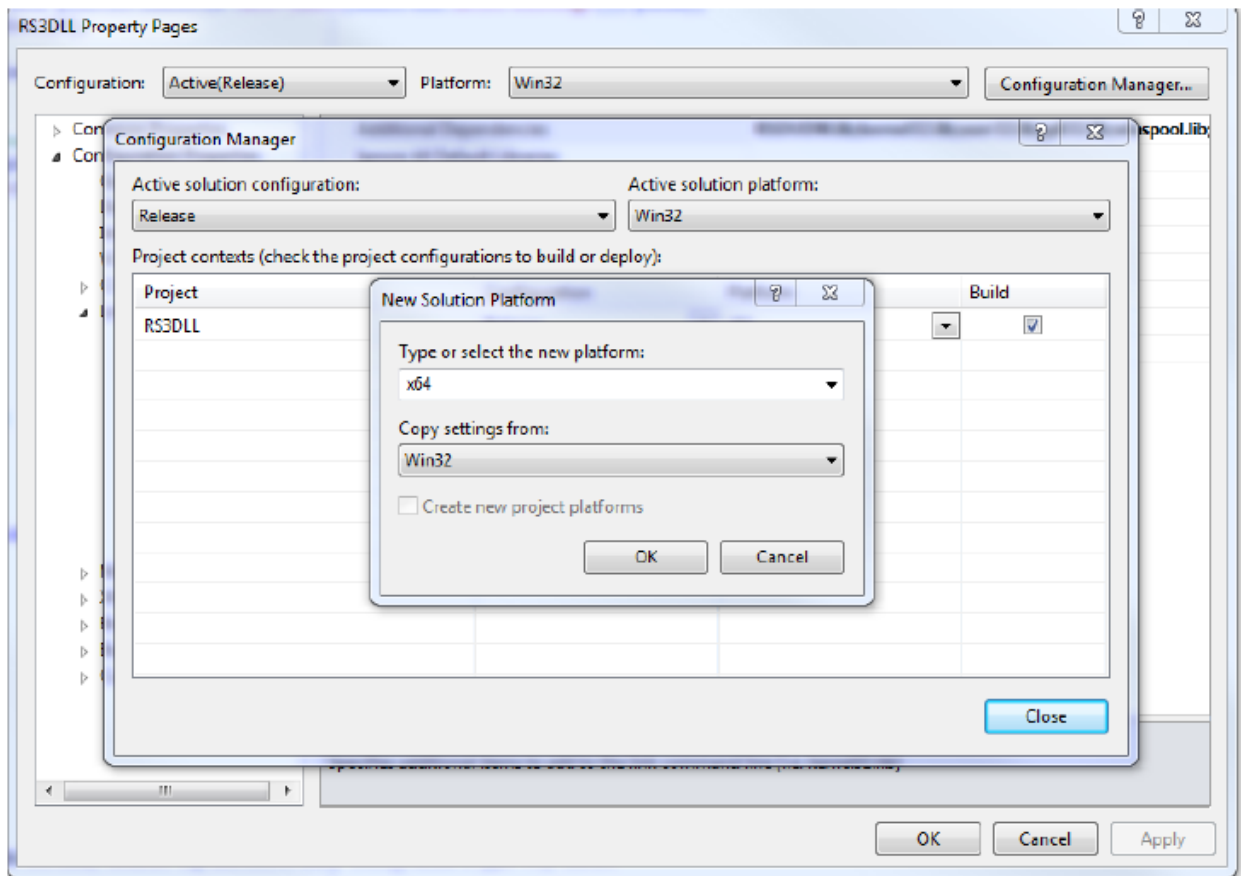


4.1 Using the RS3 Library

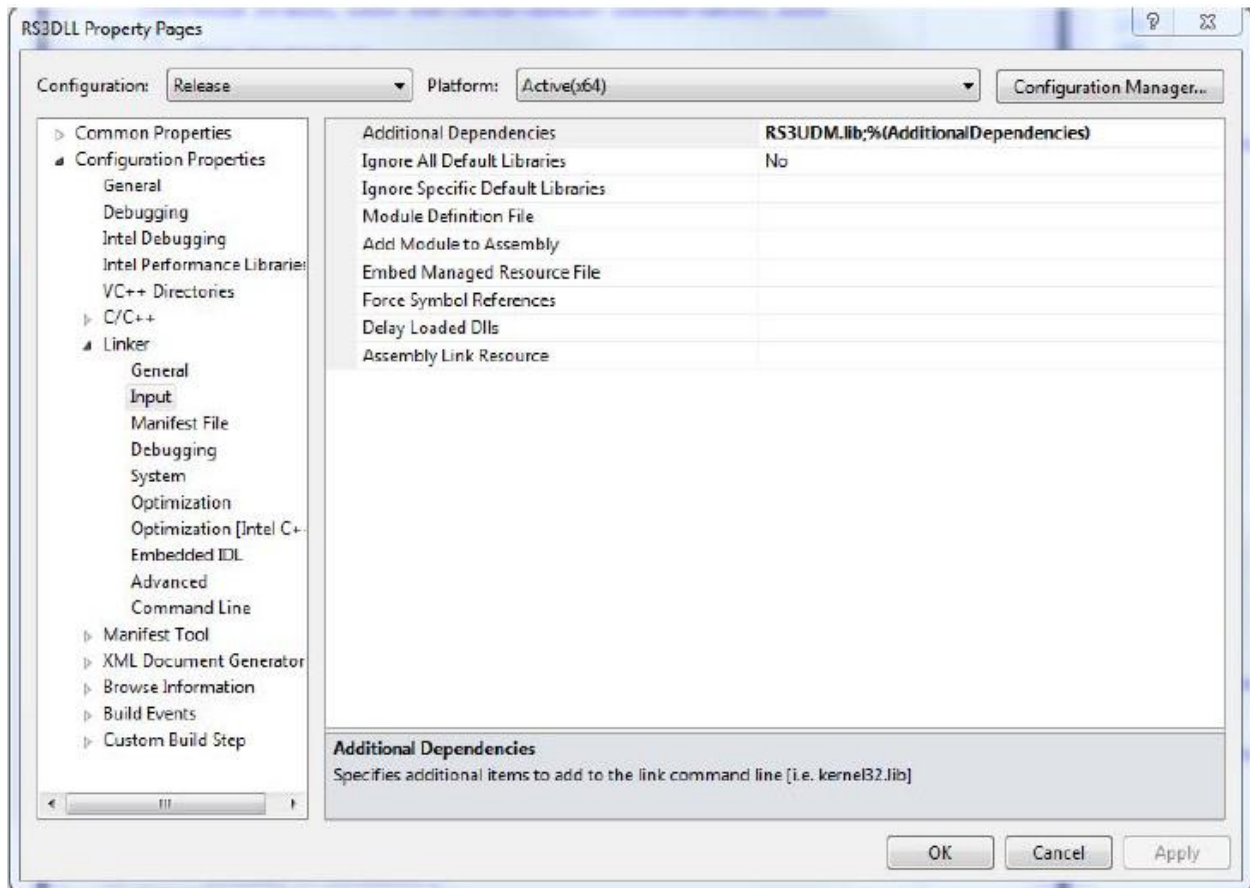
Now add the following files to the project: {Algebra.h; Definition.h; GaussPoint.h;Material.h; Matsolid.h; MatUserDefined.h; mkl_cblas.h; mkl_lapackc.h; mkl_types.h;SaturationFunction.h and Tensors.h}. Those files are included in the UDM folder in the installation folder of RS3.

Do this by putting these files in the same directory as your project file (.vcxproj) and then in Visual Studio right click on the project name on the left and choose Add->Existing Item. Choose those files and click Open.

Only 64-bit version is supported in RS3. Thus we need to compile a 64 bit version of the dll. Go to Configuration Manager on the top right of the dialog. Click on Platform->New. Pick x64 from the drop down menu and click OK.



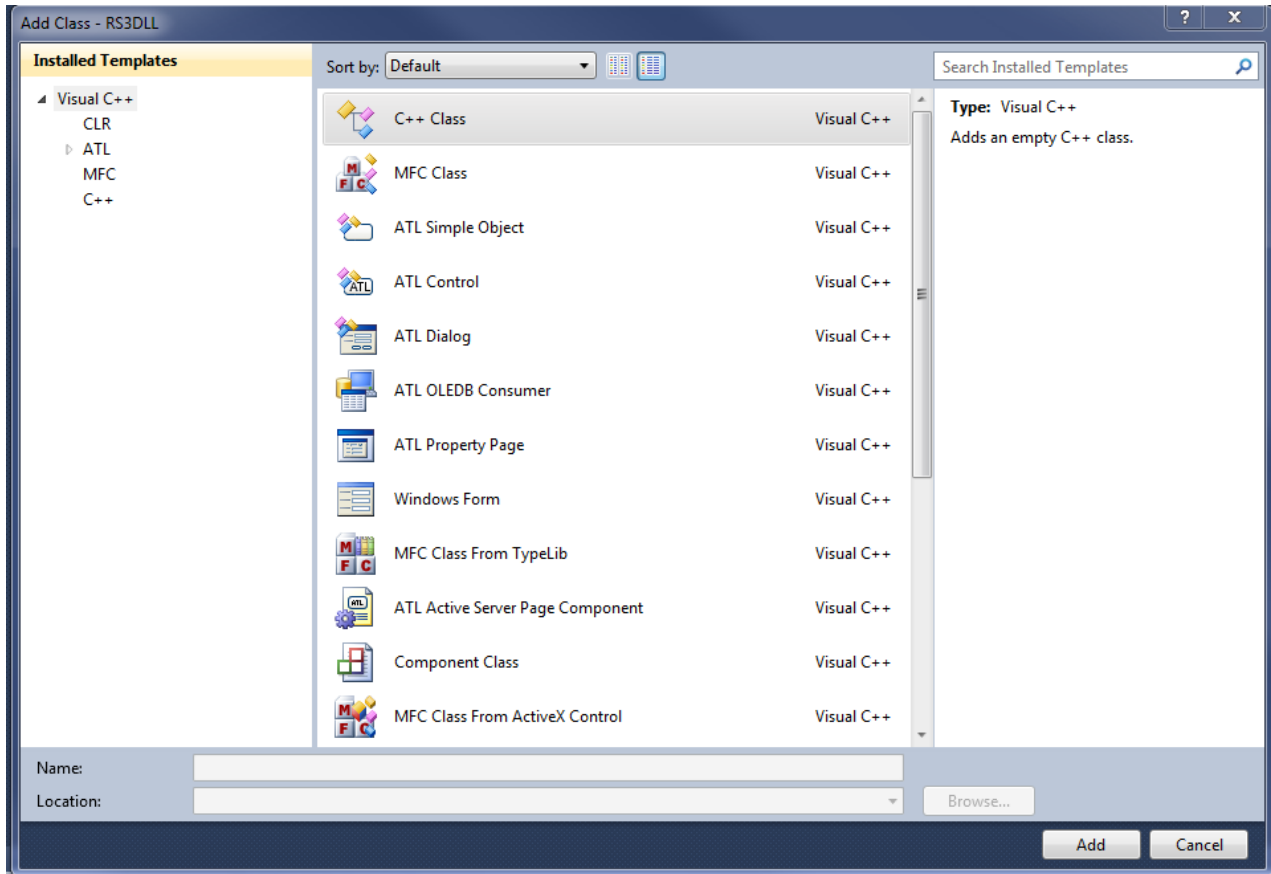
The implementation of these classes is in the file RS3UDM.lib. The file also needs to be in the project folder. Right click on the project on the left and select Linker → Input. Beside Configuration at the top of the window, select All Configurations from the pull-down menu. Now in the box next to Additional Dependencies type RS3UDM.lib.



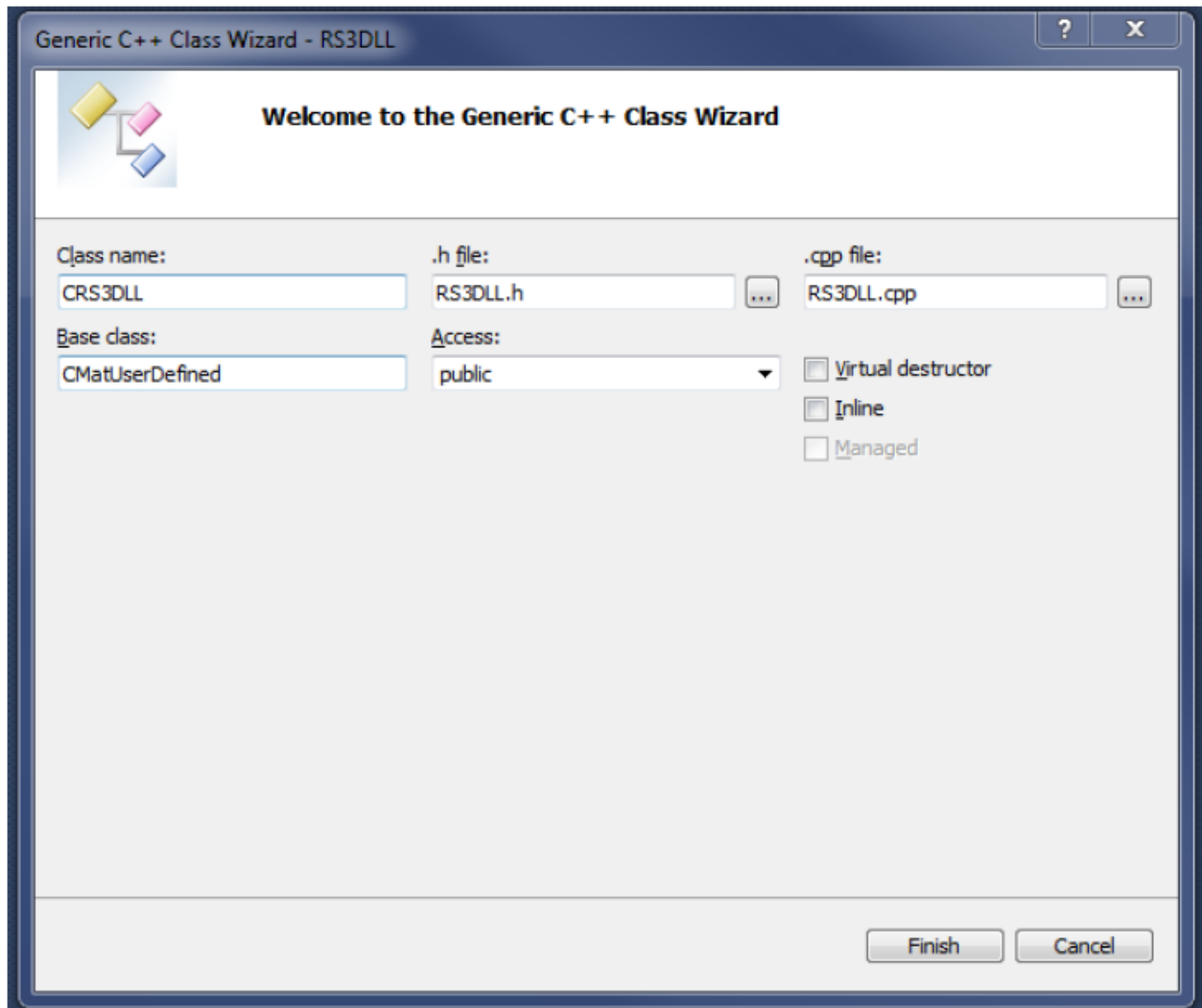
IMPORTANT NOTE: RS3UDM.lib was compiled with the release configuration. If you compile with the debug configuration, you may experience errors.

4.2 RS3DLL Implementation

Right click on the project on the left and select Add -> Class



Type CRS3DLL in the Class name and CMatUserDefined in the Base class name and click Finish.



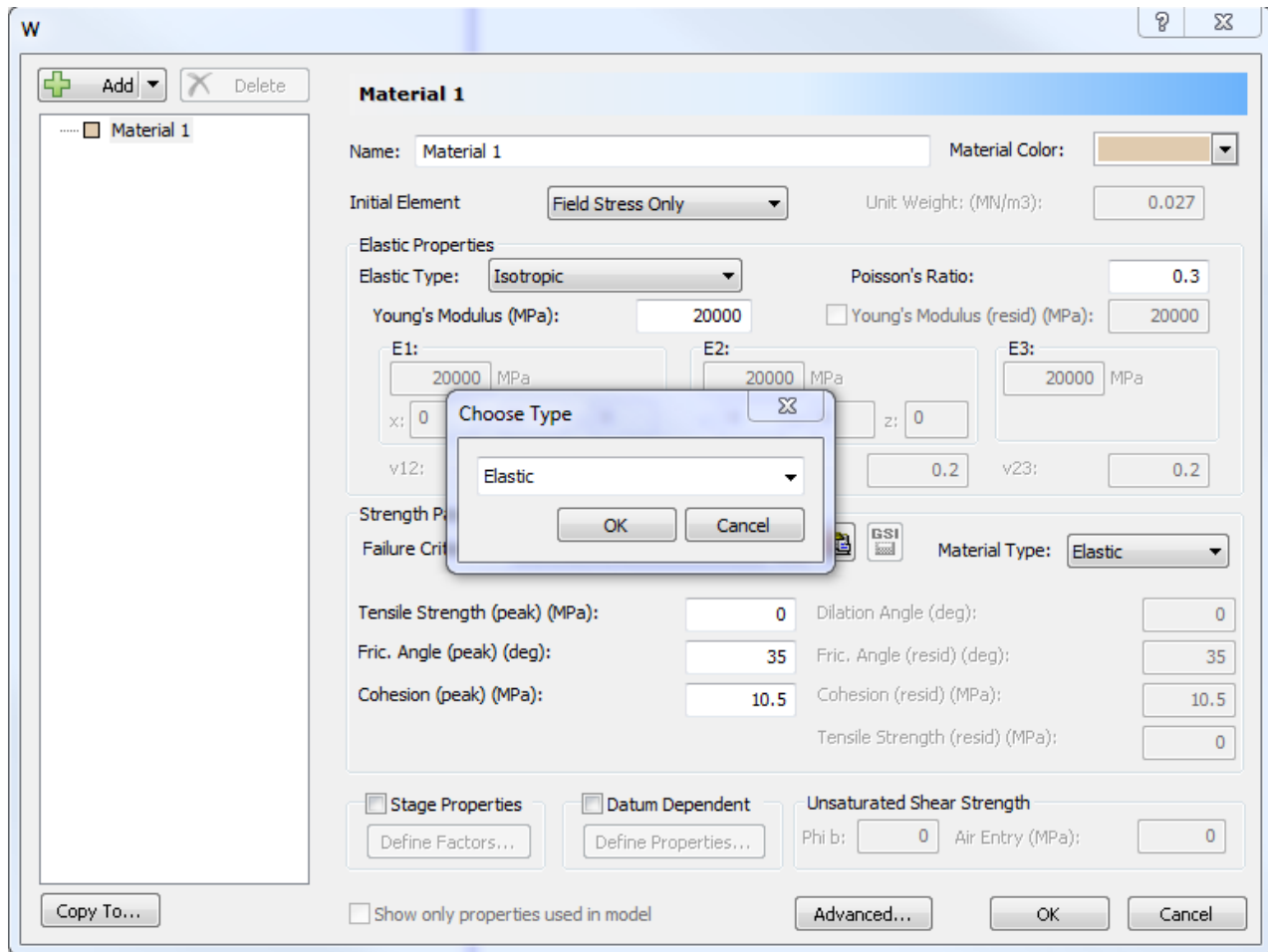
In the header file should include all of the implementation of the Export Function sections. The implementation mentioned in the Class structure should also be included in the header and source files. Now copy the content of the RS3DLL.h and RS3DLL.cpp included with the installation into the RS3DLL.h and RS3DLL.cpp in the project.

4.3 Compilation

Now go to Build->Build Solution. You need to select 64 bit platform. Assuming that you have included the linker, you should now be able to compile the program. The result will be a file with extension .dll. As mentioned above, the program may not work properly if you compile in debug mode.

4.4 Running

Put your dll file in RS3 installation folder. Now open the modeller. Go to Materials & Staging tab, Properties->Define Materials -> Add -> Add User Defined Material. You should see all of the materials that you include in the dlls.



Congratulation! You have created your first user defined model for RS3.