Writing User Defined Constitutive Model

Created By: Roozbeh Geraili Mikola, PhD, PE
Email: adonis4geo@outlook.com
Web Page: www.geowizard.org

Introduction

Since components in ***ADONIS*** are modular and self-contained, they can be designed and developed separately. Thus, users can define their own constitutive model and integrate the model into the ***ADONIS*** program by using a dynamic-linking library (dll) with relatively little dependence on other modules. Different material classes may be defined to describe different material properties or behavior. These classes are derived from an abstract class **ConstitutiveModel**, which defines their common behavior. Various constitutive models are represented as virtual functions in classes, derived from the **ConstitutiveModel** base class. The main function of the constitutive model is to return new stresses, given strain increments. Any derived constitutive model class must provide actual functions to replace the virtual member functions in the **ConstitutiveModel** base class. The generated dll file needs to be placed in the installation folder of the program. The program will then detect and load the dll automatically. All the files referenced in this section are contained in the "***\ADONIS\PluginFiles***" directory.

Export Functions

The following functions must be provided in the dll.

```
// entry point into a dynamic-link library (DLL)
int __stdcall DllMain(void *,unsigned, void *) {
  return 1;
}
// export class instance
extern "C" __declspec(dllexport) ConstitutiveModel *createInstance() {
  UserConmodelElastic *m = new UserConmodelElastic();
  return m;
}
```

Member Functions

Any derived constitutive-model class must provide actual functions to replace the virtual member functions in **ConstitutiveModel**. These functions perform the operations described below.

const char* getConModelName() const

This function returns a string containing the name of the constitutive model as the user will refer to it with the **MODEL** command. For example, **"udelastic"** would be a valid string in C++. This must be a unique name, and is used for synchronizing Save/Restore. This name is also used in the file name of the DLL described below.

int getPropertyNum() const

This function returns the number of model properties.

const char* getPropertyNames() const

This function returns a string containing the names of model properties. The following string is a valid example: {**"shear, bulk"**}. Property names are delimited by commas (',').

const char* getPropertyTypes() const

This function returns a string containing the typrs of model properties. Property type could be either "value" or "table". The following string is a valid example: {**"value, value"**}. Property types are delimited by commas (',').

double getProperty(int index) const

A value should be returned for the model property of sequence number **index** (previously defined by a **getProperty()** call, with **index** = 0 denoting the first property).

double getDefaultProperty(int index) const

A value should be returned for the default model property of sequence number n. This value will be assigned to each parameter whenever property default button in the property dialog (GUI) is pressed.

const char* getPropertyUnits(int index) const

This function returns a string containing the units of model properties. The following string is a valid example: {**"M/LT2,M/LT2"**}. Property names are delimited by commas (','). 6 different unit types should be provided:

1) user-defined (Dimension Symbol)
2) SI: m-pa-N/m3
3) SI: m-kpa-kN/m3
4) SI: m-Mpa-MN/m3
5) Imperial: ft-psf-lbf/ft3
6) Imperial: in-psi-lb/in3

void setProperty(int index, double val)

The supplied value of **index** is the sequence number (starting with 0) of the property name previously specified by means of a **getProperties()** call. The model object is required to store the supplied value in an appropriate private member variable.

ConstitutiveModel *clone()

A new object, of the same class as the current object, must be created, and a pointer to it of type **ConstitutiveModel** returned. This function is called whenever program trys to assign the same model to another element.

Double getConfinedModulus() const

The model object must return a value for its best estimate of the maximum confined modulus. This is used by program to compute the stable timestep. For a linear elastic model, the confined modulus is $K + 4G/3$.

double getShearModulus() const

The model object must return a value for its best estimate of the current tangent shear-modulus.

double getBulkModulus() const

This function should return its best estimate of the current tangent bulk-modulus.

void initialize(StressState *st)

This function is called once for each model object (i.e., for each gauss point) when the ***ADONIS*** cvlces, and at the beginning of the **run()** method if **setIfNeedInitialize()** returns **true**. The model object may perform initialization of its property or state variables, or it may do nothing.

void update(StressState *st)

This function is called for each gauss point in the finite element at each cycle. The model must update the stress tensor from strain increments. The structure **st** (which is a pointer to **StressState** class**)** contains the current stress components and the computed strain increment components.

int **getSaveNum()** const

This function return the number of parameters that needs to be saved. The parameters will be provided by **setSaveValue** function**.**

void **setSaveValue(**int **index,** double **val)**

The supplied value of **index** is the sequence number (starting with 0) of the property that need to be saved. These parameters are the one the will be saved when program tries to store the model in the file. These parameters could be any parameters in the class other than the one specified in the list of properties using **setProperty**.

double **getSaveValue(**int **index)**

A value should be returned for the model stored property of sequence number **index** (previously defined by a **setSaveValue()** call, with **index** = 0 denoting the first property).

Optional Functions

The following function are optional to modify. If no modifications has made the default parameters will be used.

const char* getPropertyDescription(int index) const

This function returns a string containing the description of each property.

bool canFail() const

This function returns false if **solve("elastic")** command is being executed.

void setCanFail(bool b)

This function indicates whether model will be able to perform plastic correction.

bool needInitialize() const

This function returns true if initialization is needed.

void setIfNeedInitialize(bool b)

This function indicates whether initialization needs to be called again.

bool supportsPropertyScaling() const

This Function returns **true** if property scaling is supported for factor-of-safety calculations

void scaleProperties(const double &,const std::vector<int> &)

This function will be used when "solve fos" is being called to scale the properties for factor of safety calculation. Please note that, this function is still under development and changing it will not have any effect.

bool isPropertyReadOnly(int index) const

Return True if property of index should be considered read only, and not allowed to be set by the user.

State Indicators of Zones

Each gauss point in the element has a member variable that maintains its current state indicator. The member variable has integer value that can be used to represent a maximum of 11 distinct states (starting from 0). The state indicators are used by built-in constitutive models to denote plastic failure of a zone. See list below for integer assignment and the corresponding failure state for built-in constitutive models.

```
//   0:Elastic
//   1:ShearNow
//   2:ShearPast
```

```
//  3:TensionNow
//  4:TensionPast
//  5:VolumeNow
//  6:VolumePast
//  7:JointShearNow
//  8:JointShearPast
//  9:JointTensionNow
//  10:JointTensionPast
```

Pre-requisite

All the header files included in the UDCM folder must be included in the dll to compile. (*conmodel_global.h*, *constitutivemodel.h*, *ctable.h* and *stressstate.h*). The dll also needs to be linked with the libconmodel.lib (for 32-bit version). Note: The static library files were compiled with the **release** configuration. If you compile with the debug configuration, you may not get the correct values for debugging.

Creating User-Defined Model DLLs

ADONIS is built using **Qt Creator**, a cross platform development environment, which is part of the SDK for the **Qt** GUI application development. Constitutive model plug-in DLLs need to link against the **Qt** runtime library to be compatible at runtime. **Qt** is a free and open-source widget toolkit for creating graphical user interface (GUI) as well as cross-platform applications that run on various software and hardware platforms. **Qt** is currently being developed by The **Qt** Company, a publicly listed company, and the **Qt** Project under open-source governance, involving individual developers and organizations working to advance **Qt**. **Qt** is available under both commercial licenses and open source GPL 2.0, GPL 3.0, and LGPL 3.0 licenses. If you haven't installed the **Qt** already, please go ahead and follow the steps mentioned in the **Appendix A** and install it then follow the following steps to create user-defined model DLL:

1) Unzip all files in the ***udcm_example_mingw32.zip*** (located at ***PluginFiles*** folder) to a desired location. The zip file includes all the necessary files. All the header files and static library links have been created for this example file. In case you want to change the name of the model or header file please follow the sequences below.
2) Open Qt Creator. Go to File->Open File or Project...

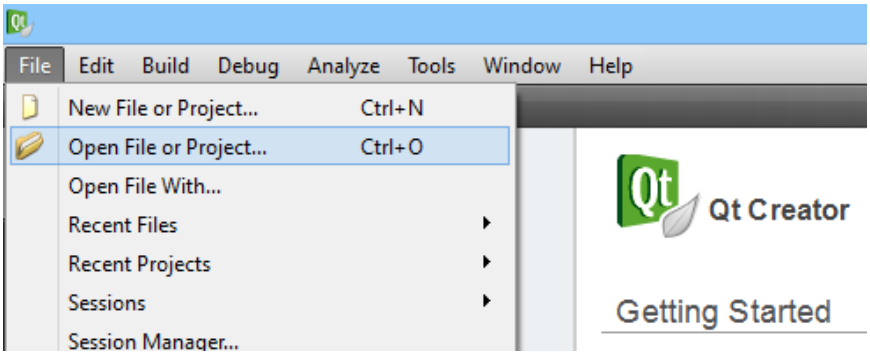

Figure 1- Open File or Project from menu

3) Then navigate to the project you wish to open and click on the .pro file (i.e., *udcm_example.pro*).

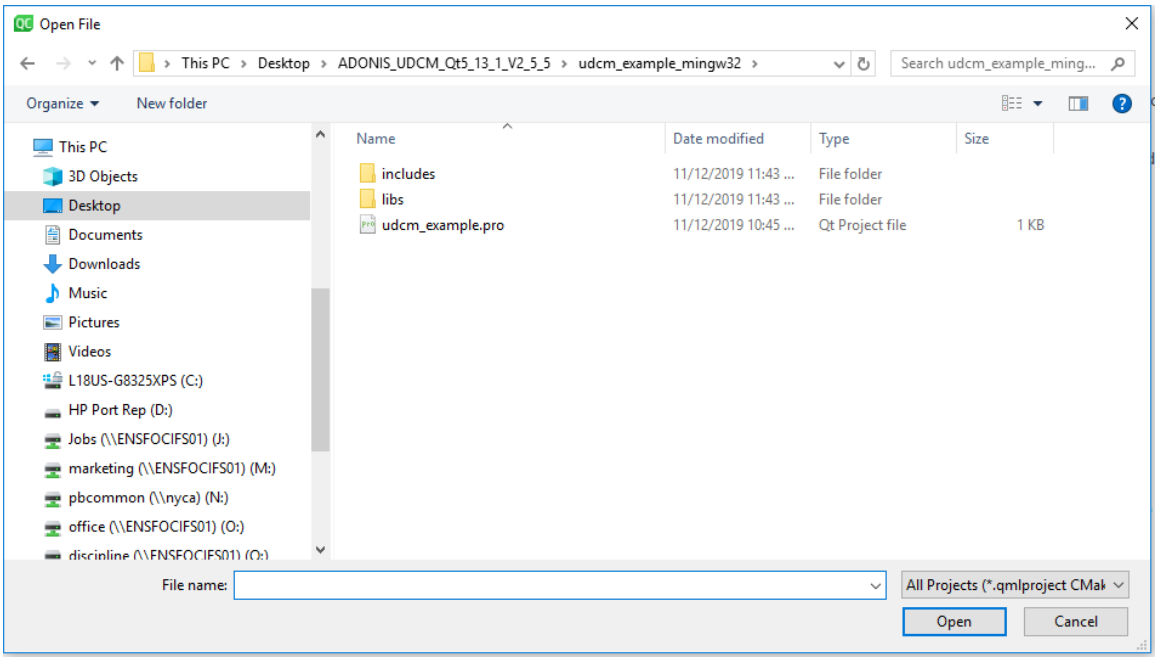

Figure 2- Select project from

4) Select "**MinGW 32-bit**" compiler and press "**Configure Project**"

Figure 3- Configure project. Make sure to select MinGW 32-bit compiler.

5) To rename the files (i.e. .h or .cpp), right-click on each file and select "**Rename…**" and type in your desired name.

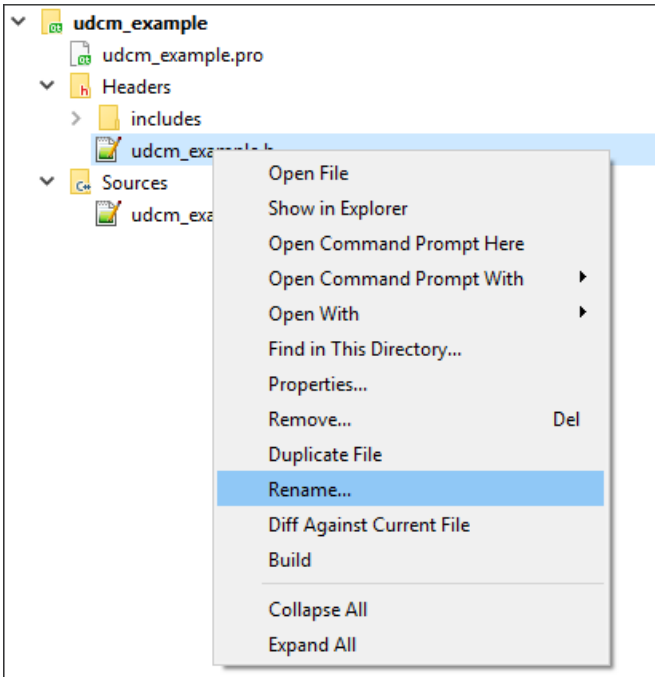

Figure 4- Rename existing files.

6) To rename the output dll file name double click on project file (i.e. .pro) and modify the target name (i.e. TARGET = ucdm_example).

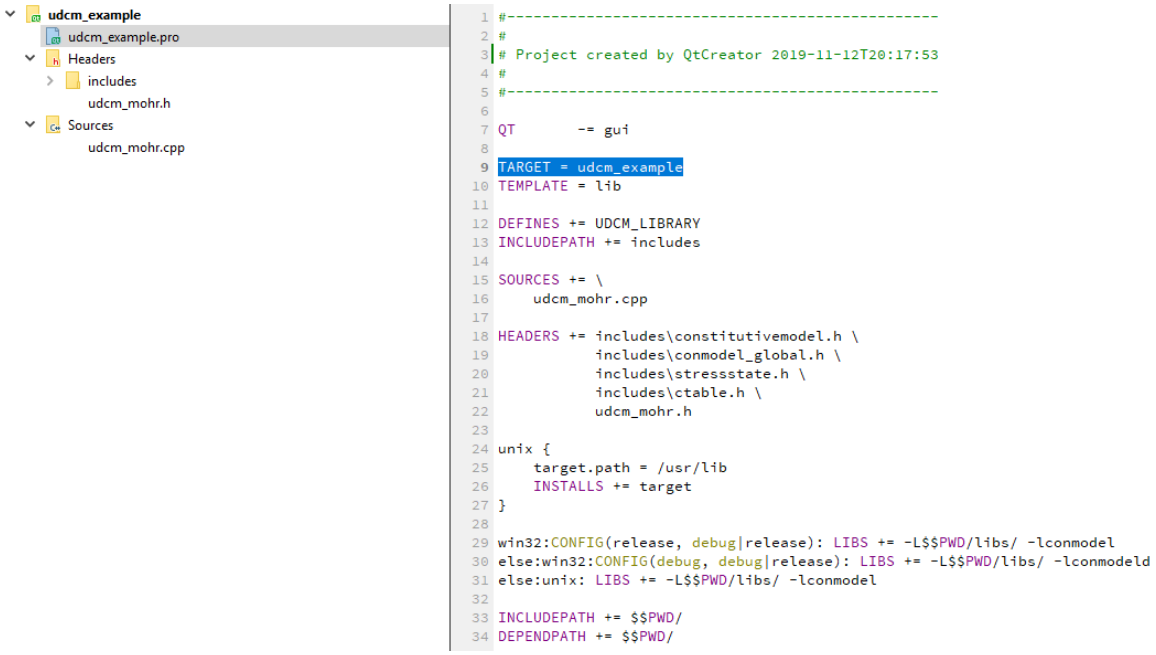

Figure 5- Modify project file.

7) In case you want to rename the class name, make sure you rename the “UDCM_Example” to “UDCM_Mohr” in both header and source file (i.e. *udcm_mohr.h* and *udcm_mohr.cpp*).

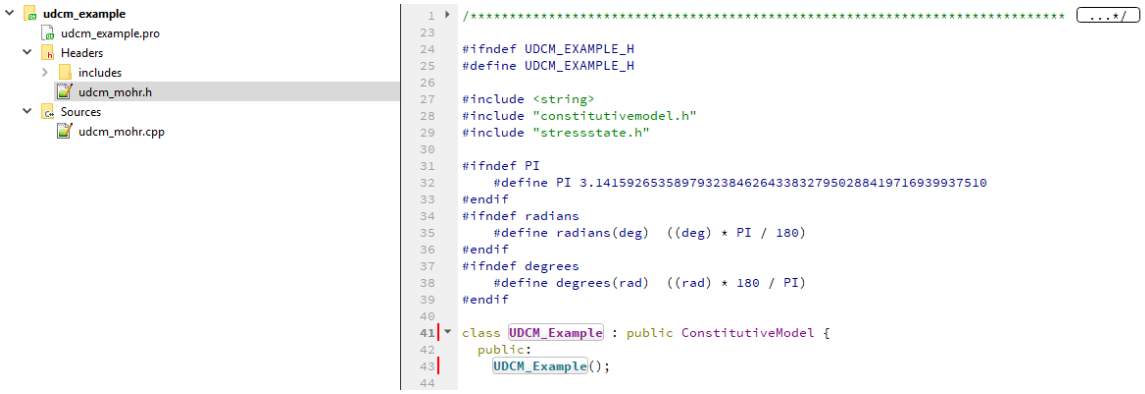

Figure 6- Modify class name based on new selected name.

8) To change the project’s name, close the **Qt Creator** and rename the project file (i.e. *udcm_example.pro*). Remove the “*udcm_example.pro.user*” file if exists then reopen the renamed project and configure again after opening the project (like step 4).

Compilation

After modifying the library functions make sure that you select the **“Release”** mode as build type. Now go to Build->Build Project and build the project. 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.

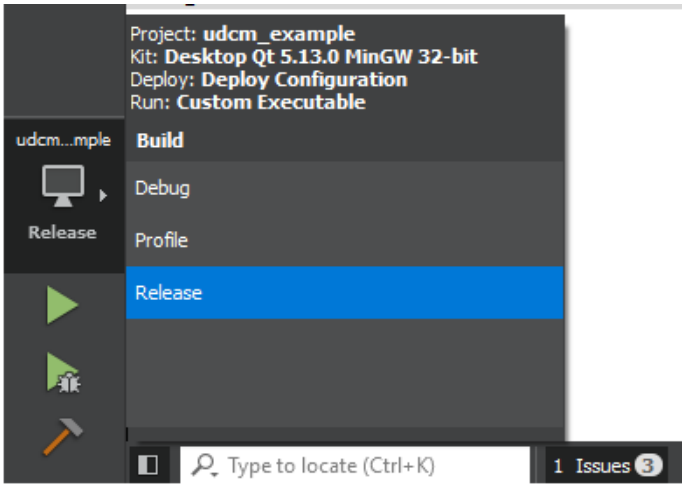

Figure 7- Select release mode and build the project.

Running

Copy the generated dll file from the location provided at “Build Setting” page (shown in the Figure 8) then put your dll file in the **plugins** folder which is in the installation folder (i.e.

\ADONIS\exe32\plugins). ***ADONIS*** will automatically recognize the user defined constitutive model in the restart. You should see all of the materials that you included in the material list.

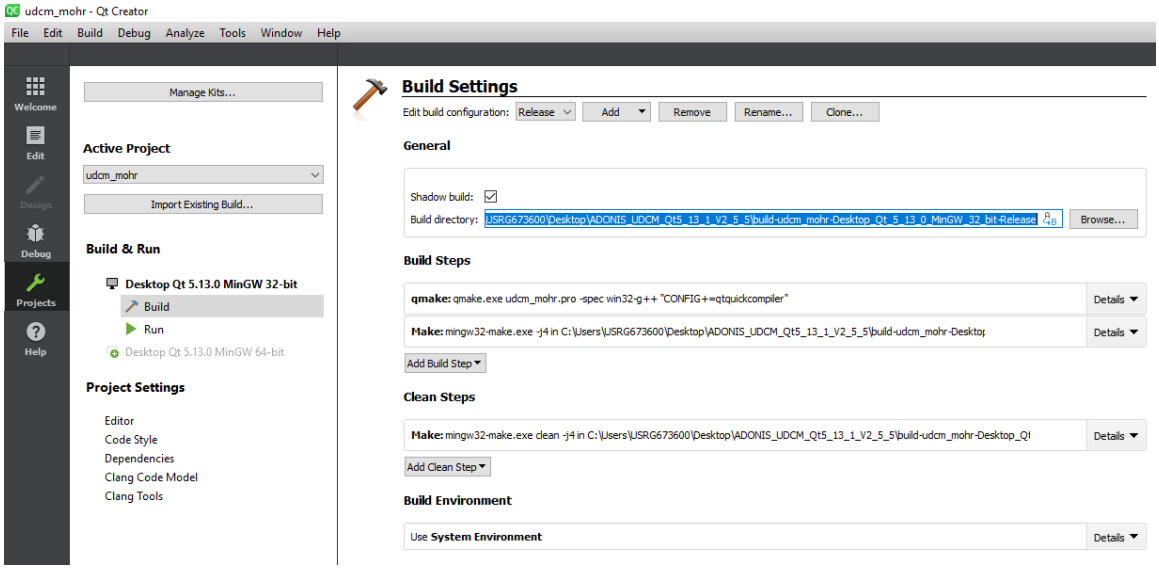

Figure 8- Find the location of the generated dll file from Projects page.

APPENDIX A

Download Qt and Qt Creator

To get started with the non-commercial version for free, go to http://qt-project.org/downloads to see something like what the following screenshot then select "**Go open source**":

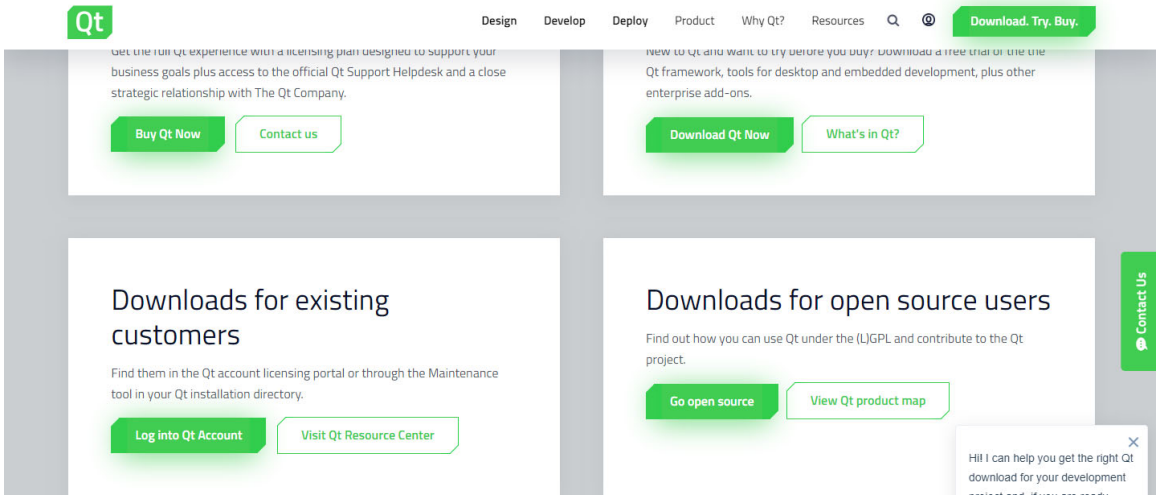

Figure A1- Go to Qt Project web page.

At the bottom of the page click on "**Download the Qt Online Installer**"

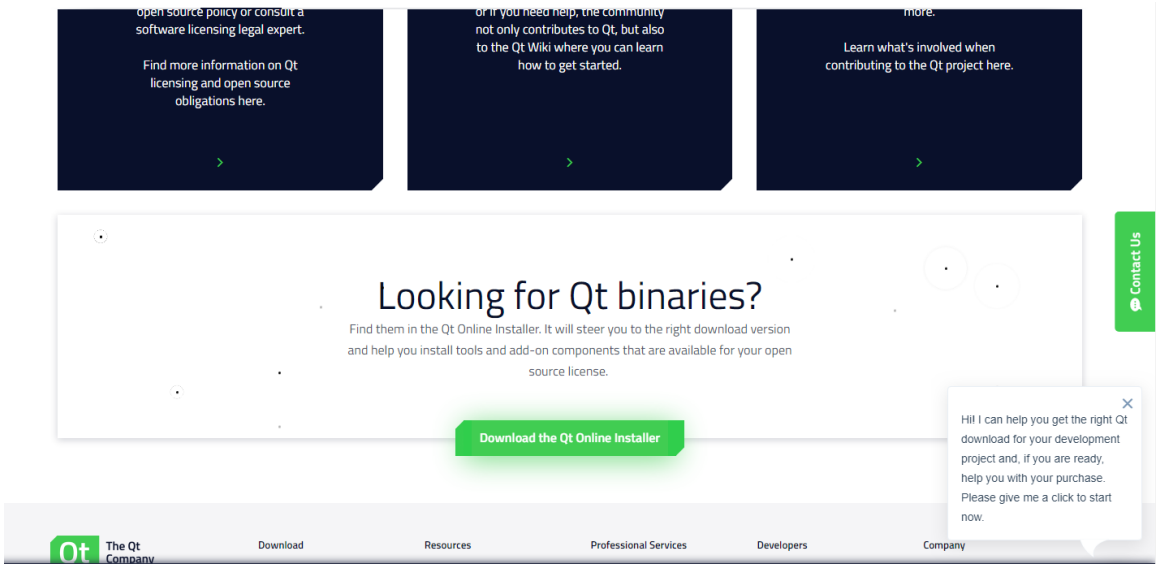

Figure A2- Click on "Download the Qt online Installer"

Next click on “**Download**”. Your download should start automatically.

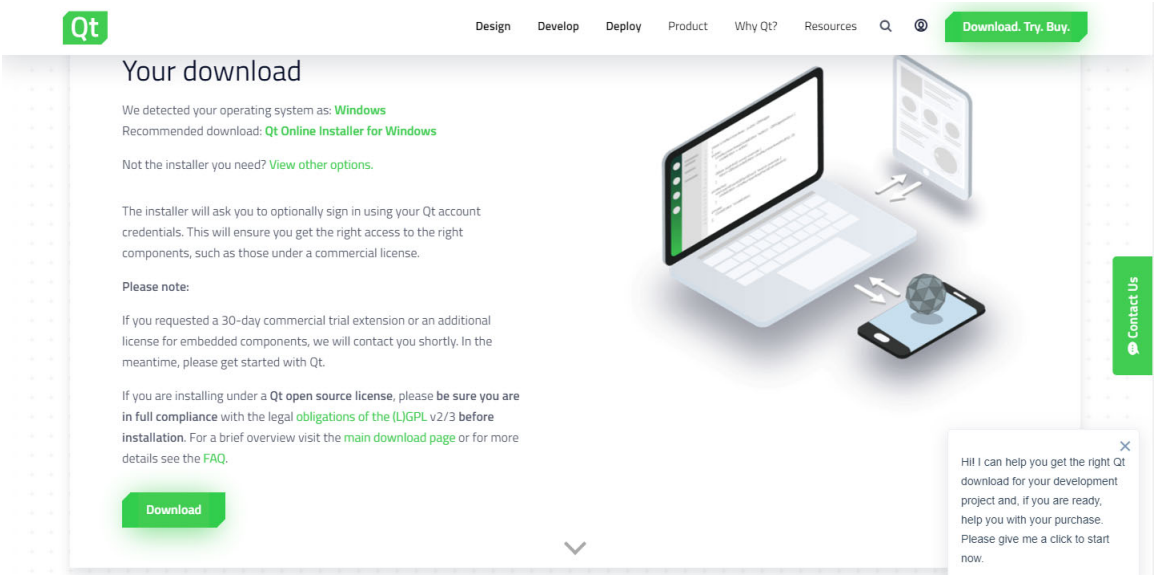

Figure A3- Download the online installer file.

Install/Setup Qt Creator

In general, the default settings have been used when installing Qt Creator. Please note that a Qt account is required when installing Qt Creator as shown in Figure A4. Choose an existing account or create a new account. There is no cost associated with a Qt account.

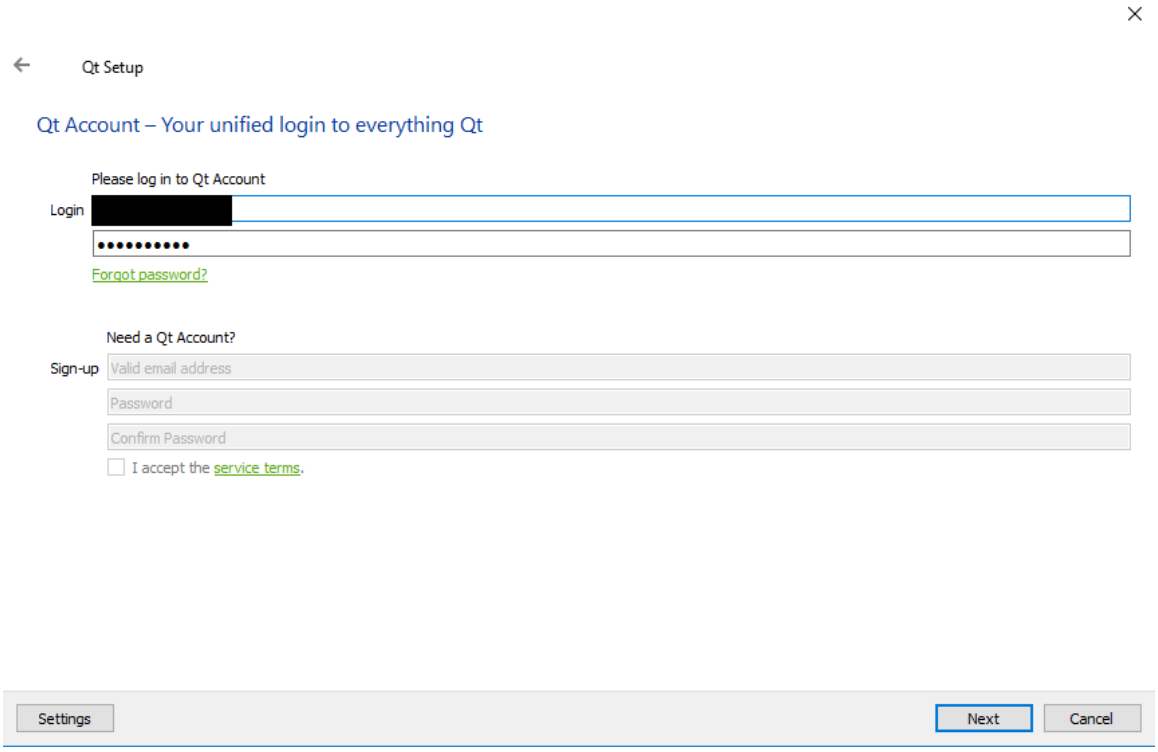

Figure A4- Qt account is required for installation.

The default components to install were selected as shown in Figure A5. Please make sure the **MinGW 32-bit** and **MinGW 64-bit** are selected. If you plan to choose other components select these at this step. Please note that some components may only be available when using a commercial version of Qt.

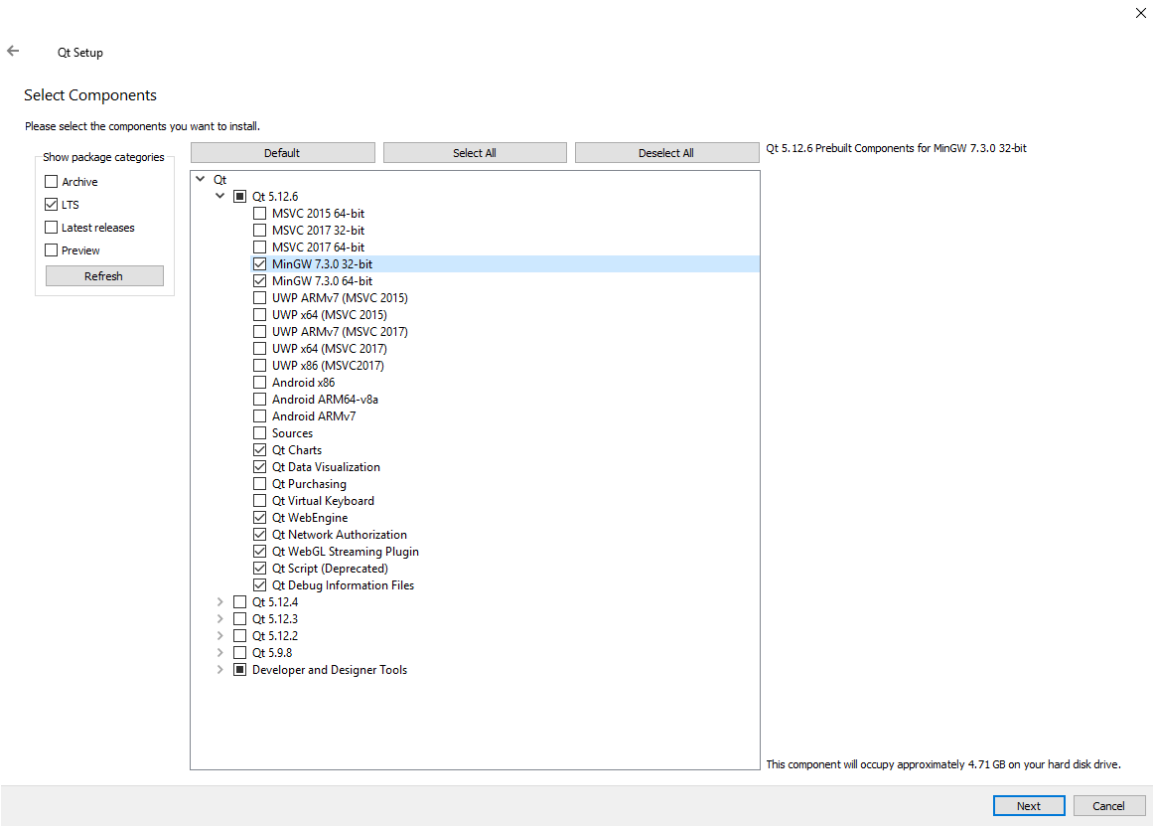

Figure A5- Select the installation components. Make sure MinGW-32bit compiler is selected.

When Qt Creator has been installed select to launch it and click the “Finish” button and then start building your constitutive model.