

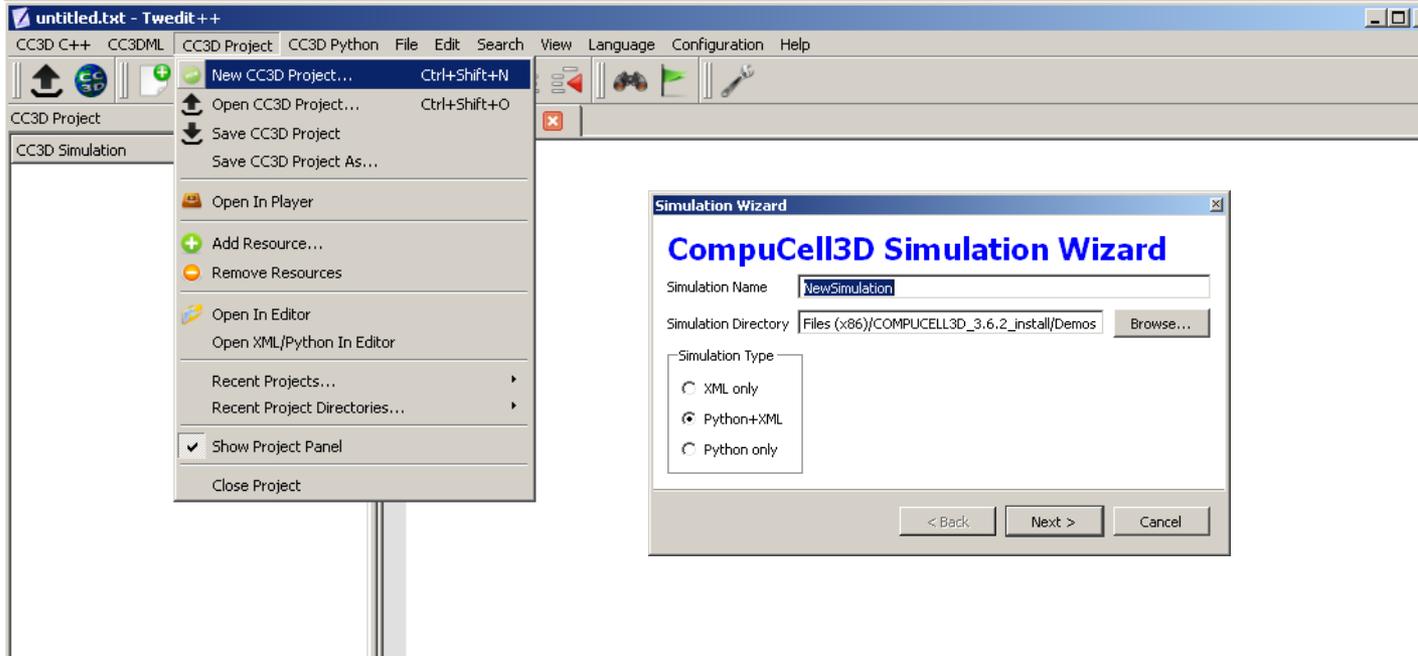
Introduction to Twedit++-CC3D

Maciej Swat

Why use Twedit++-CC3D

- CompuCell3D customized version of twedit++ programmer's editor
- Simplifies development of simulations by providing Simulation wizard, Python and CC3DML code assistants
- Speeds-up significantly development of CC3D C++ extension modules (plugins and steppables). It sometimes makes sense to replace slow Python steppable with fast C++ counterpart

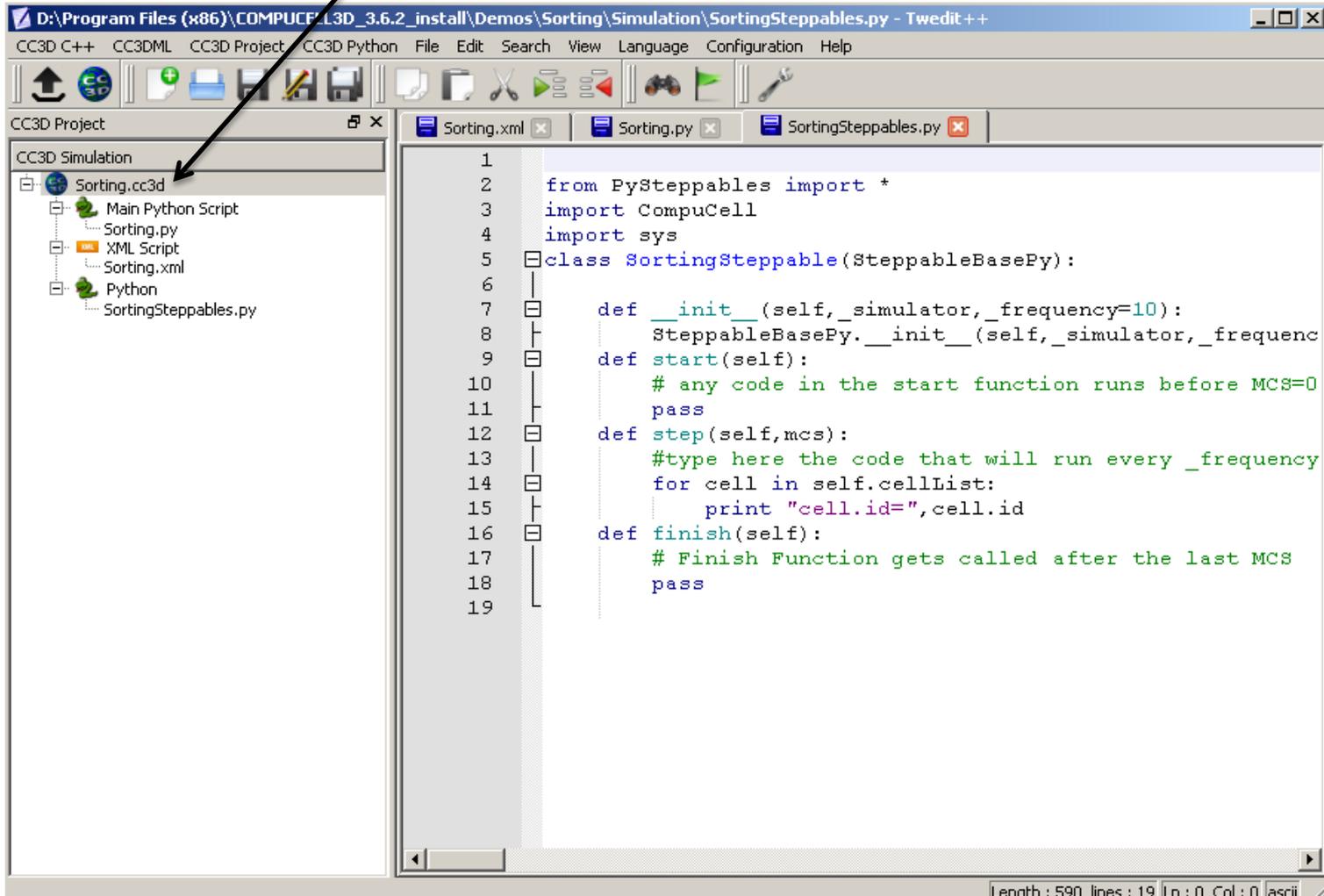
Starting New CC3D project



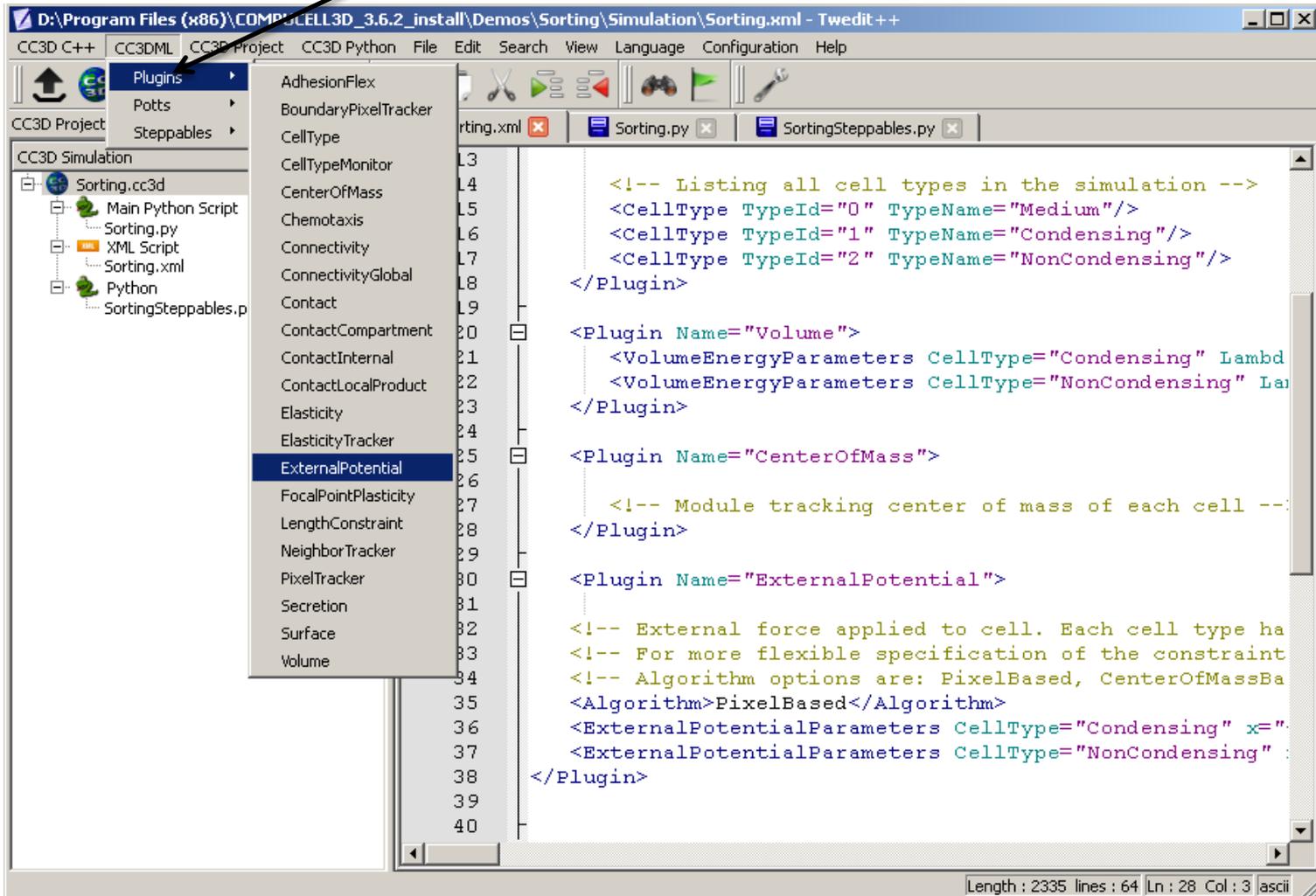
Simulation Wizard

- In simulation wizard you specify cell types, diffusing fields, cell properties etc...
- If you forget something , do not worry, Twedit++-CC3D code assistants will help you put everything you need into your simulation code

Double-clicking on project name
opens all project files in
Twedit++



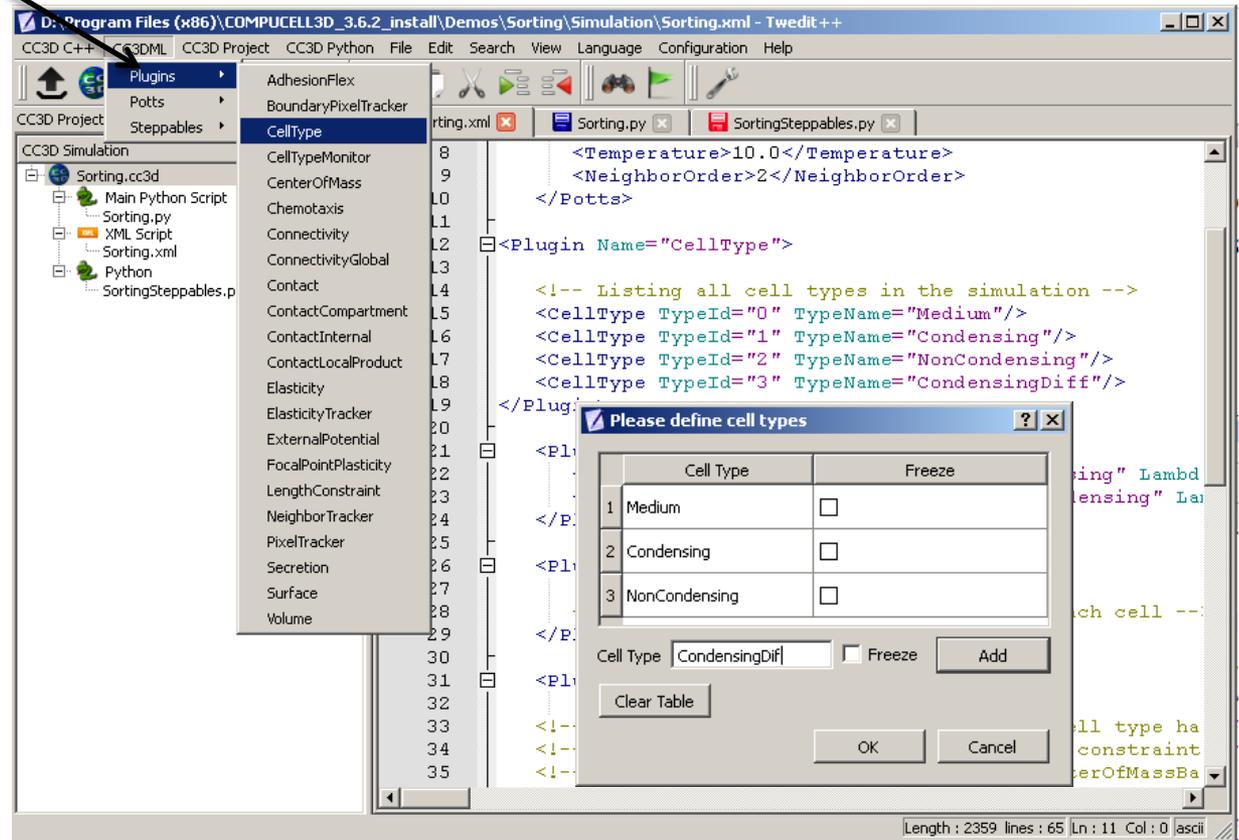
Cell behaviors can be added after completing wizard-based simulation autogeneration. Here we add ExternalPotential plugin which exerts a force on cells. Most CC3D modules are covered. If a plugin is missing from pull down menu it can be added to Twedit pretty easily. Please let us know and we will do it for you.



Adding/removing cell types is most hated operation because it requires modification of many simulation components. Twedit++ makes this operation pretty painless and it does so without using sedatives:

Go to CC3DML->Plugins->CellType and insert new type.

Important: After adding removing cell type you have to change CC3D module definitions which use cell type names



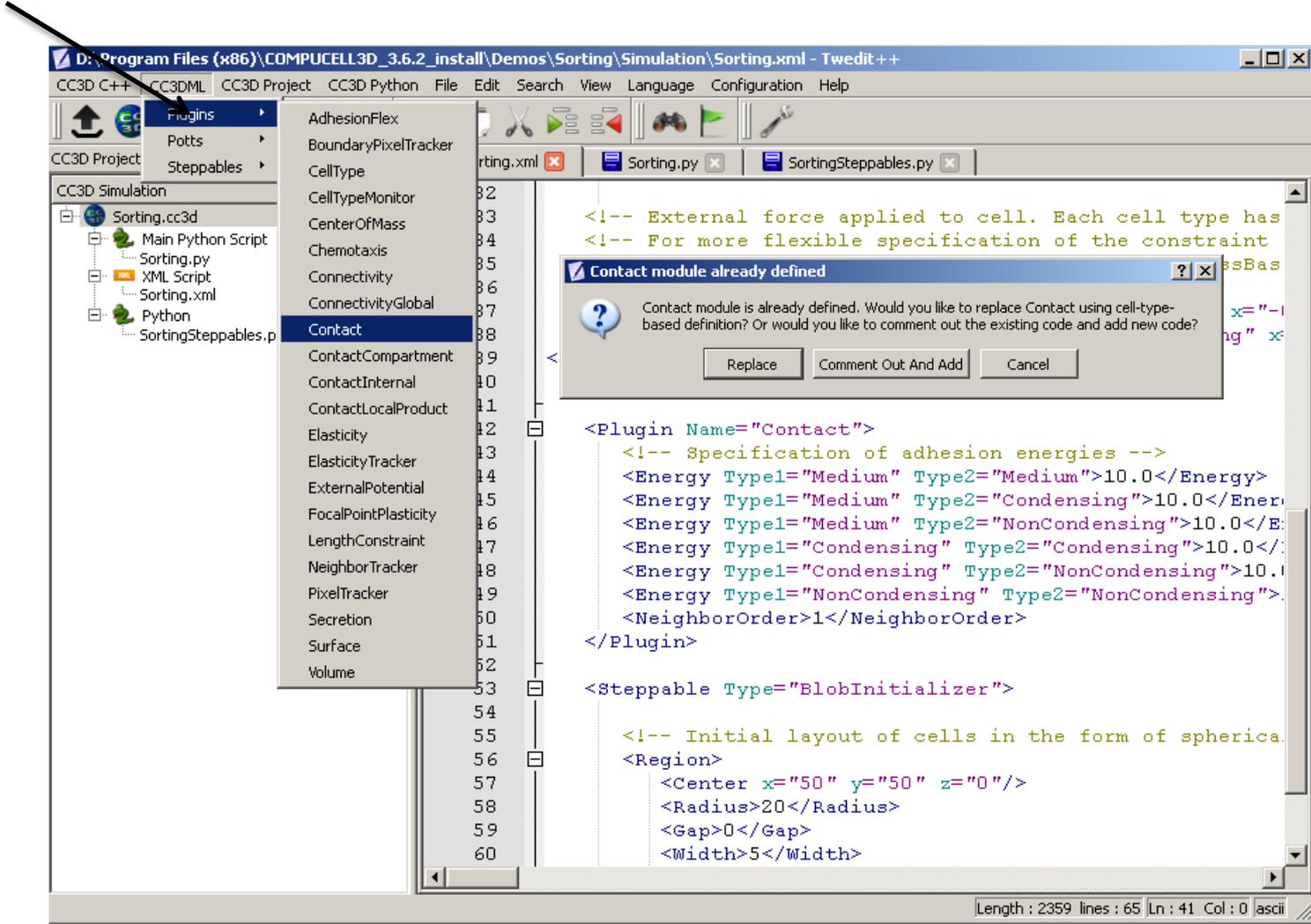
The screenshot shows the Twedit++ interface with the CC3DML Plugins menu open. The 'CellType' option is selected. A dialog box titled 'Please define cell types' is displayed in the foreground. The dialog contains a table with the following data:

Cell Type	Freeze
1 Medium	<input type="checkbox"/>
2 Condensing	<input type="checkbox"/>
3 NonCondensing	<input type="checkbox"/>

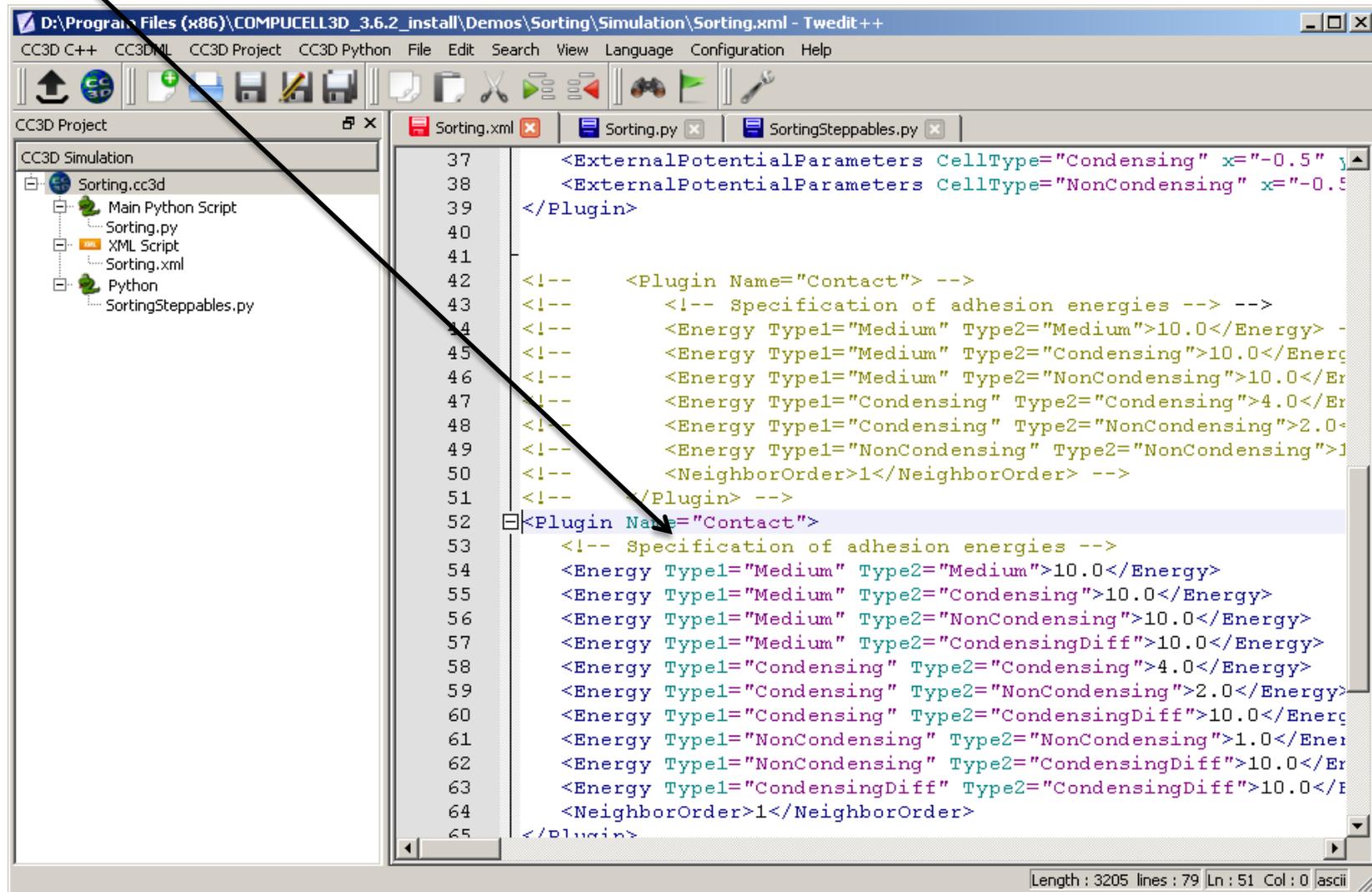
Below the table, there is a text input field containing 'CondensingDif', a 'Freeze' checkbox, and an 'Add' button. There is also a 'Clear Table' button. At the bottom of the dialog are 'OK' and 'Cancel' buttons. The background shows the XML code in the editor, including a <Plugin Name="CellType"> block and a list of cell types: <CellType TypeId="0" TypeName="Medium"/>, <CellType TypeId="1" TypeName="Condensing"/>, <CellType TypeId="2" TypeName="NonCondensing"/>, and <CellType TypeId="3" TypeName="CondensingDif"/>.

Changing plugin definition which list cell types is relatively easy . Most plugins are supported although occasionally you may need to do some typing:

For Contact plugin go to CC3DML->Plugins->Contact and you will get a choice of either replacing or commenting out existing code (if the code for contact is already there)



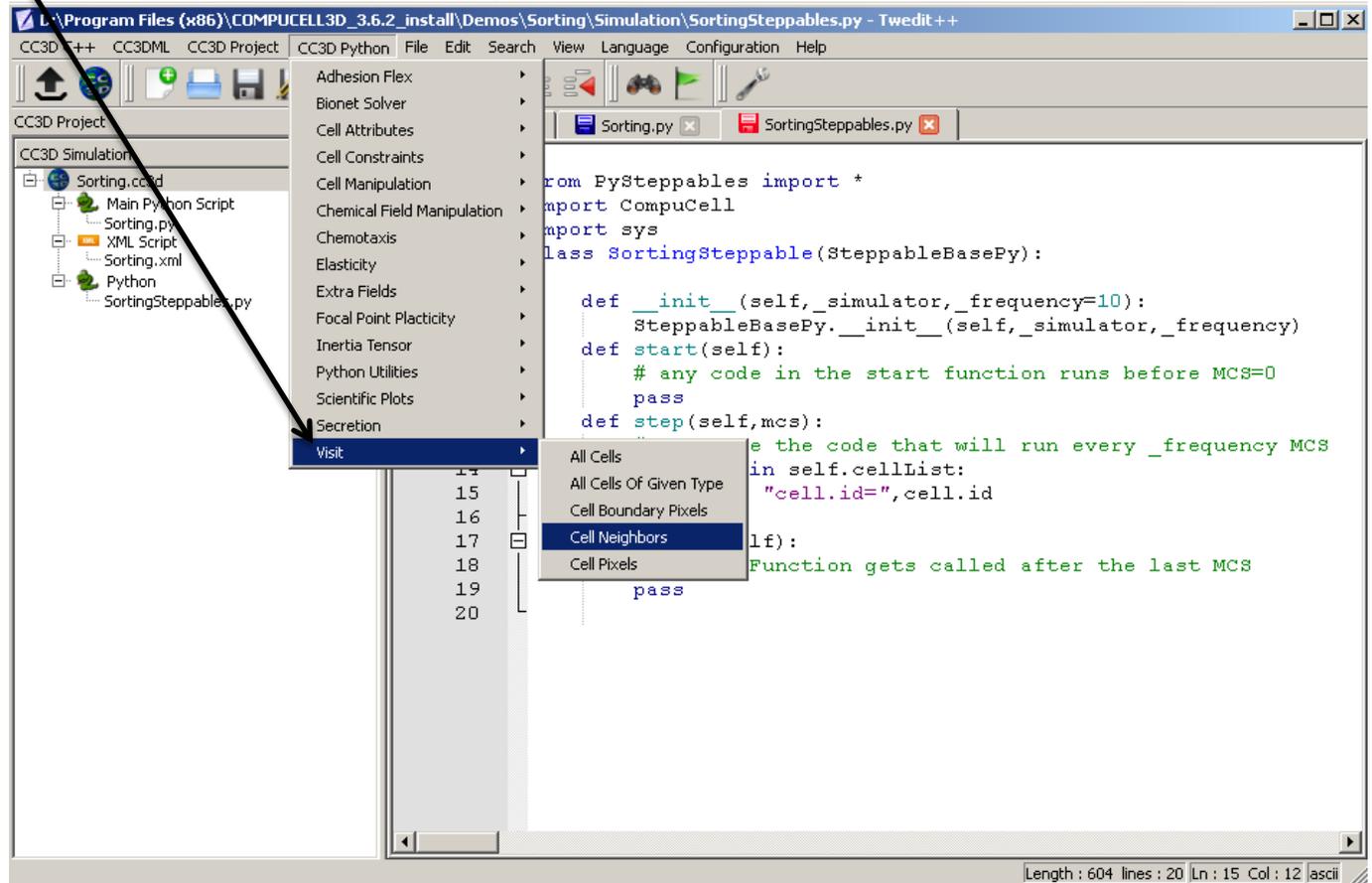
New code for contact includes new types we have defined but contact energy values between types present in the old simulation code are preserved – less typing for you



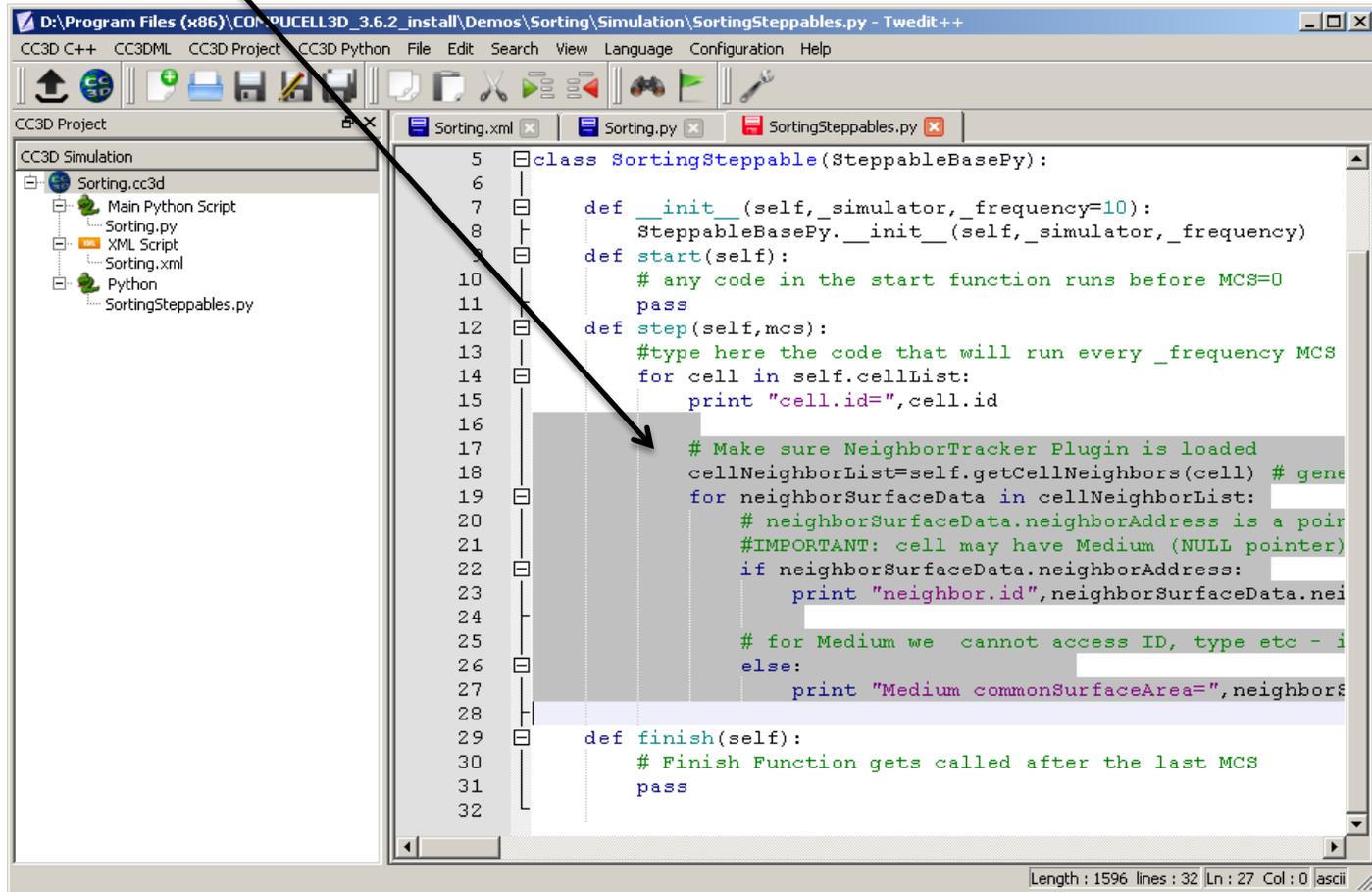
```
37 <ExternalPotentialParameters CellType="Condensing" x="-0.5" y="0.5" z="0.5" />
38 <ExternalPotentialParameters CellType="NonCondensing" x="-0.5" y="0.5" z="0.5" />
39 </Plugin>
40
41
42 <!-- <Plugin Name="Contact"> -->
43 <!-- <!-- Specification of adhesion energies --> -->
44 <!-- <Energy Type1="Medium" Type2="Medium">10.0</Energy> -->
45 <!-- <Energy Type1="Medium" Type2="Condensing">10.0</Energy> -->
46 <!-- <Energy Type1="Medium" Type2="NonCondensing">10.0</Energy> -->
47 <!-- <Energy Type1="Condensing" Type2="Condensing">4.0</Energy> -->
48 <!-- <Energy Type1="Condensing" Type2="NonCondensing">2.0</Energy> -->
49 <!-- <Energy Type1="NonCondensing" Type2="NonCondensing">1.0</Energy> -->
50 <!-- <NeighborOrder>1</NeighborOrder> -->
51 <!-- </Plugin> -->
52 <Plugin Name="Contact">
53 <!-- Specification of adhesion energies -->
54 <Energy Type1="Medium" Type2="Medium">10.0</Energy>
55 <Energy Type1="Medium" Type2="Condensing">10.0</Energy>
56 <Energy Type1="Medium" Type2="NonCondensing">10.0</Energy>
57 <Energy Type1="Medium" Type2="CondensingDiff">10.0</Energy>
58 <Energy Type1="Condensing" Type2="Condensing">4.0</Energy>
59 <Energy Type1="Condensing" Type2="NonCondensing">2.0</Energy>
60 <Energy Type1="Condensing" Type2="CondensingDiff">10.0</Energy>
61 <Energy Type1="NonCondensing" Type2="NonCondensing">1.0</Energy>
62 <Energy Type1="NonCondensing" Type2="CondensingDiff">10.0</Energy>
63 <Energy Type1="CondensingDiff" Type2="CondensingDiff">10.0</Energy>
64 <NeighborOrder>1</NeighborOrder>
65 </Plugin>
```

Length : 3205 lines : 79 Ln : 51 Col : 0 ascii

Python CC3D code assistant inserts code snippets for the most common CC3D tasks. Here we add iteration over cell neighbors



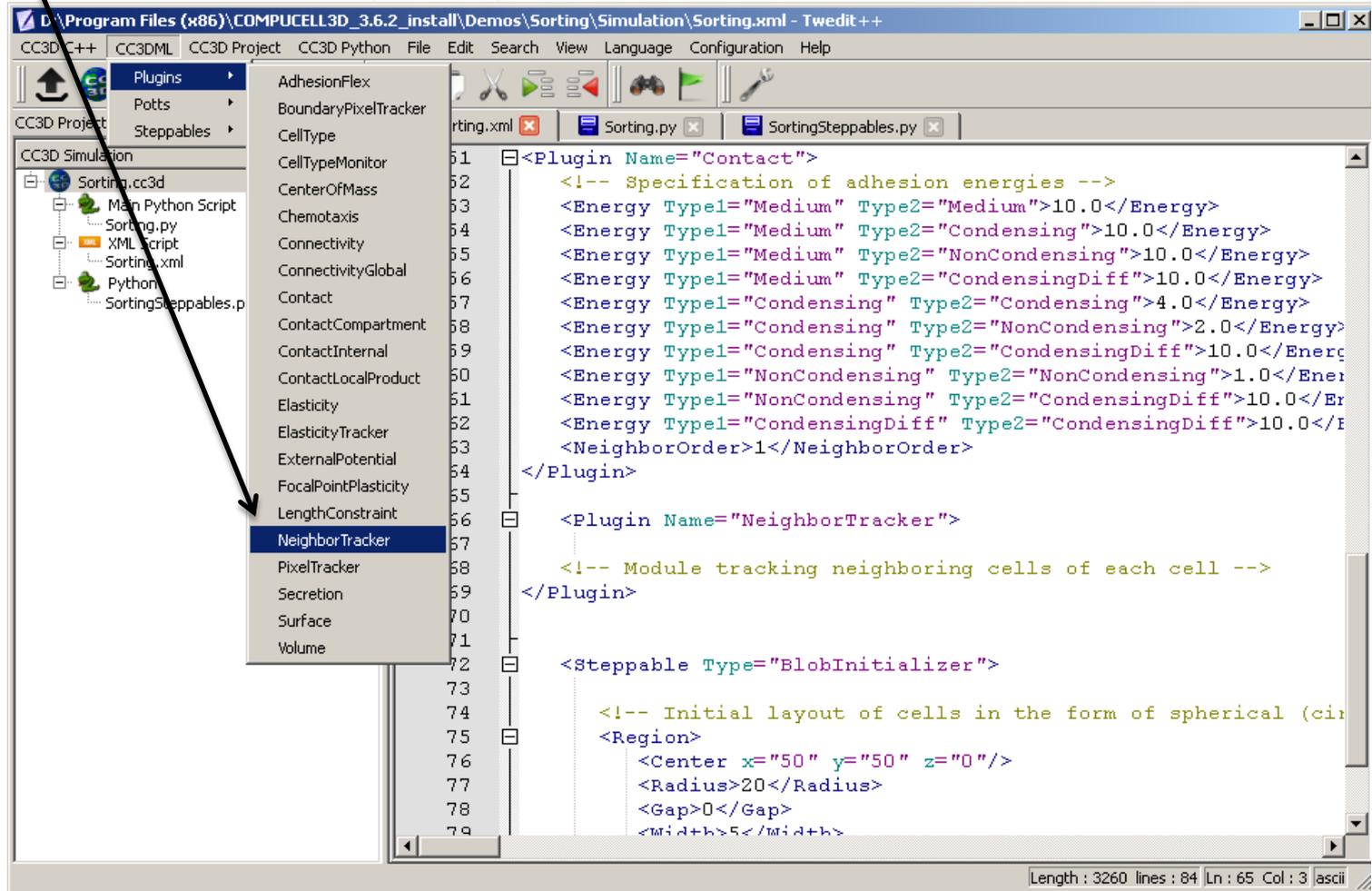
Cell neighbor iteration snippet. We usually have to do small edits to adapt it to our simulation



```
5 class SortingSteppable(SteppableBasePy):
6
7     def __init__(self,_simulator,_frequency=10):
8         SteppableBasePy.__init__(self,_simulator,_frequency)
9
10    def start(self):
11        # any code in the start function runs before MCS=0
12        pass
13
14    def step(self,mcs):
15        #type here the code that will run every _frequency MCS
16        for cell in self.cellList:
17            print "cell.id=",cell.id
18
19        # Make sure NeighborTracker Plugin is loaded
20        cellNeighborList=self.getCellNeighbors(cell) # generate neighbor list
21        for neighborSurfaceData in cellNeighborList:
22            # neighborSurfaceData.neighborAddress is a pointer to the neighbor
23            #IMPORTANT: cell may have Medium (NULL pointer)
24            if neighborSurfaceData.neighborAddress:
25                print "neighbor.id",neighborSurfaceData.neighborAddress
26
27            # for Medium we cannot access ID, type etc - it's a pointer to the medium
28            else:
29                print "Medium commonSurfaceArea=",neighborSurfaceData.commonSurfaceArea
30
31    def finish(self):
32        # Finish Function gets called after the last MCS
33        pass
```

Length : 1596 lines : 32 Ln : 27 Col : 0 asci

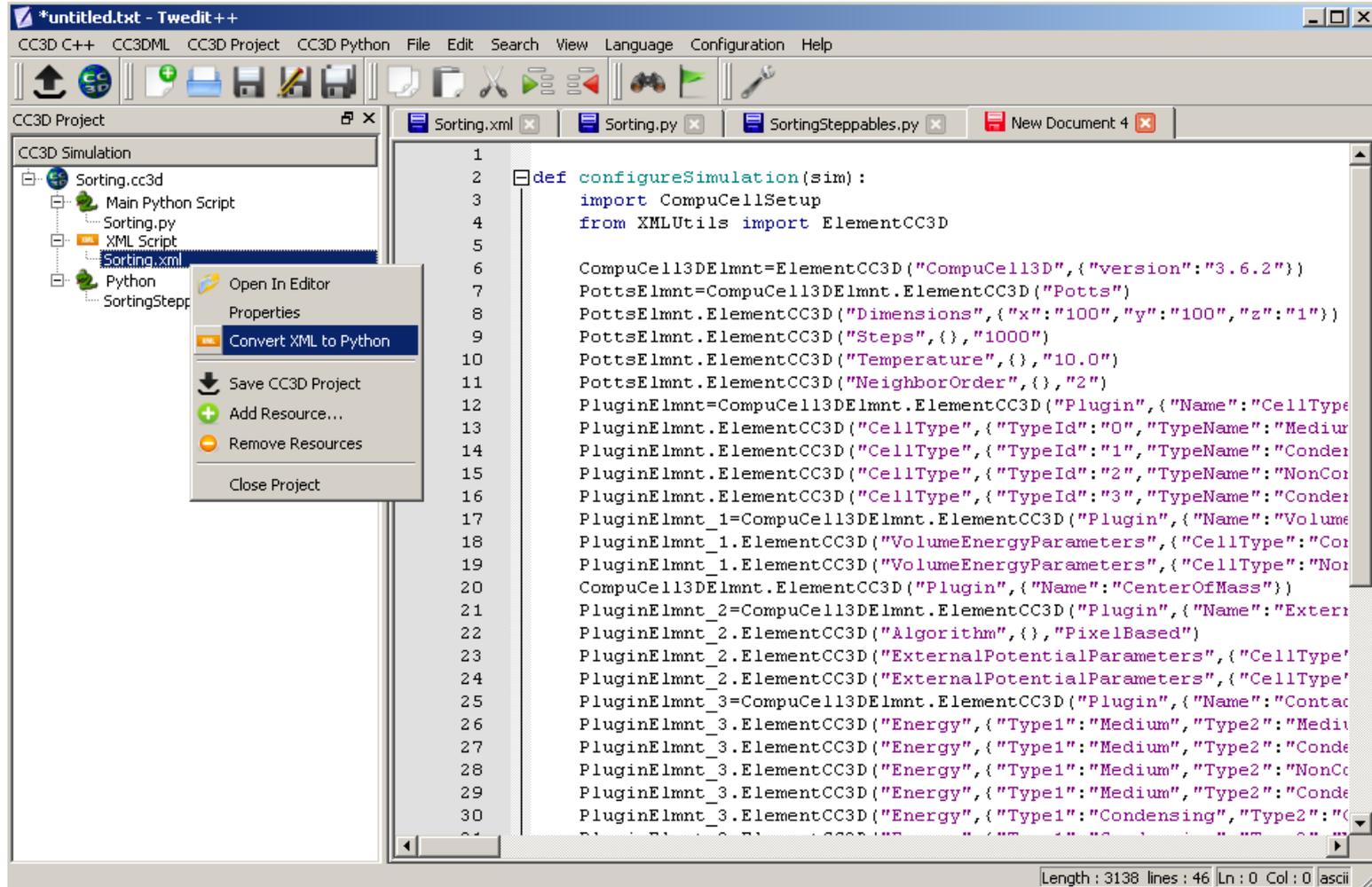
And we also must make sure that we insert NeighborTracker plugin in CC3DML:



One-click conversion of CC3DML to equivalent Python syntax

Right-click (in the CC3D Simulation panel) on the .xml file you wish to convert and choose 'Convert XML to Python'. There are two things worth noticing here:

1. One-click solutions in reality require multiple clicks
2. configureSimulation function gets generated as a new tab. You may need to copy code from here and paste it in other document.



The screenshot shows the CC3D C++ application interface. On the left, the 'CC3D Simulation' panel displays a project tree with 'Sorting.xml' selected. A context menu is open over 'Sorting.xml', with 'Convert XML to Python' highlighted. The main editor window shows the generated Python code for 'Sorting.py'. The code defines a 'configureSimulation' function that imports 'CompuCellSetup' and 'ElementCC3D' from 'XMLUtils'. It then creates a 'CompuCell13DElmnt' object and populates it with various parameters and plugins, including 'PottsElmnt', 'PluginElmnt', and 'VolumeEnergyParameters'.

```
1
2 def configureSimulation(sim):
3     import CompuCellSetup
4     from XMLUtils import ElementCC3D
5
6     CompuCell13DElmnt=ElementCC3D("CompuCell13D",{"version":"3.6.2"})
7     PottsElmnt=CompuCell13DElmnt.ElementCC3D("Potts")
8     PottsElmnt.ElementCC3D("Dimensions",{"x":"100","y":"100","z":"1"})
9     PottsElmnt.ElementCC3D("Steps",(), "1000")
10    PottsElmnt.ElementCC3D("Temperature",(), "10.0")
11    PottsElmnt.ElementCC3D("NeighborOrder",(), "2")
12    PluginElmnt=CompuCell13DElmnt.ElementCC3D("Plugin",{"Name":"CellType
13    PluginElmnt.ElementCC3D("CellType",{"TypeId":"0","TypeName":"Mediur
14    PluginElmnt.ElementCC3D("CellType",{"TypeId":"1","TypeName":"Conder
15    PluginElmnt.ElementCC3D("CellType",{"TypeId":"2","TypeName":"NonCor
16    PluginElmnt.ElementCC3D("CellType",{"TypeId":"3","TypeName":"Conder
17    PluginElmnt_1=CompuCell13DElmnt.ElementCC3D("Plugin",{"Name":"Volume
18    PluginElmnt_1.ElementCC3D("VolumeEnergyParameters",{"CellType":"Co
19    PluginElmnt_1.ElementCC3D("VolumeEnergyParameters",{"CellType":"Nor
20    CompuCell13DElmnt.ElementCC3D("Plugin",{"Name":"CenterOfMass"})
21    PluginElmnt_2=CompuCell13DElmnt.ElementCC3D("Plugin",{"Name":"Extern
22    PluginElmnt_2.ElementCC3D("Algorithm",(), "PixelBased")
23    PluginElmnt_2.ElementCC3D("ExternalPotentialParameters",{"CellType'
24    PluginElmnt_2.ElementCC3D("ExternalPotentialParameters",{"CellType'
25    PluginElmnt_3=CompuCell13DElmnt.ElementCC3D("Plugin",{"Name":"Contac
26    PluginElmnt_3.ElementCC3D("Energy",{"Type1":"Medium","Type2":"Medi
27    PluginElmnt_3.ElementCC3D("Energy",{"Type1":"Medium","Type2":"NonC
28    PluginElmnt_3.ElementCC3D("Energy",{"Type1":"Medium","Type2":"NonC
29    PluginElmnt_3.ElementCC3D("Energy",{"Type1":"Medium","Type2":"Conde
30    PluginElmnt_3.ElementCC3D("Energy",{"Type1":"Condensing","Type2":"C"
```

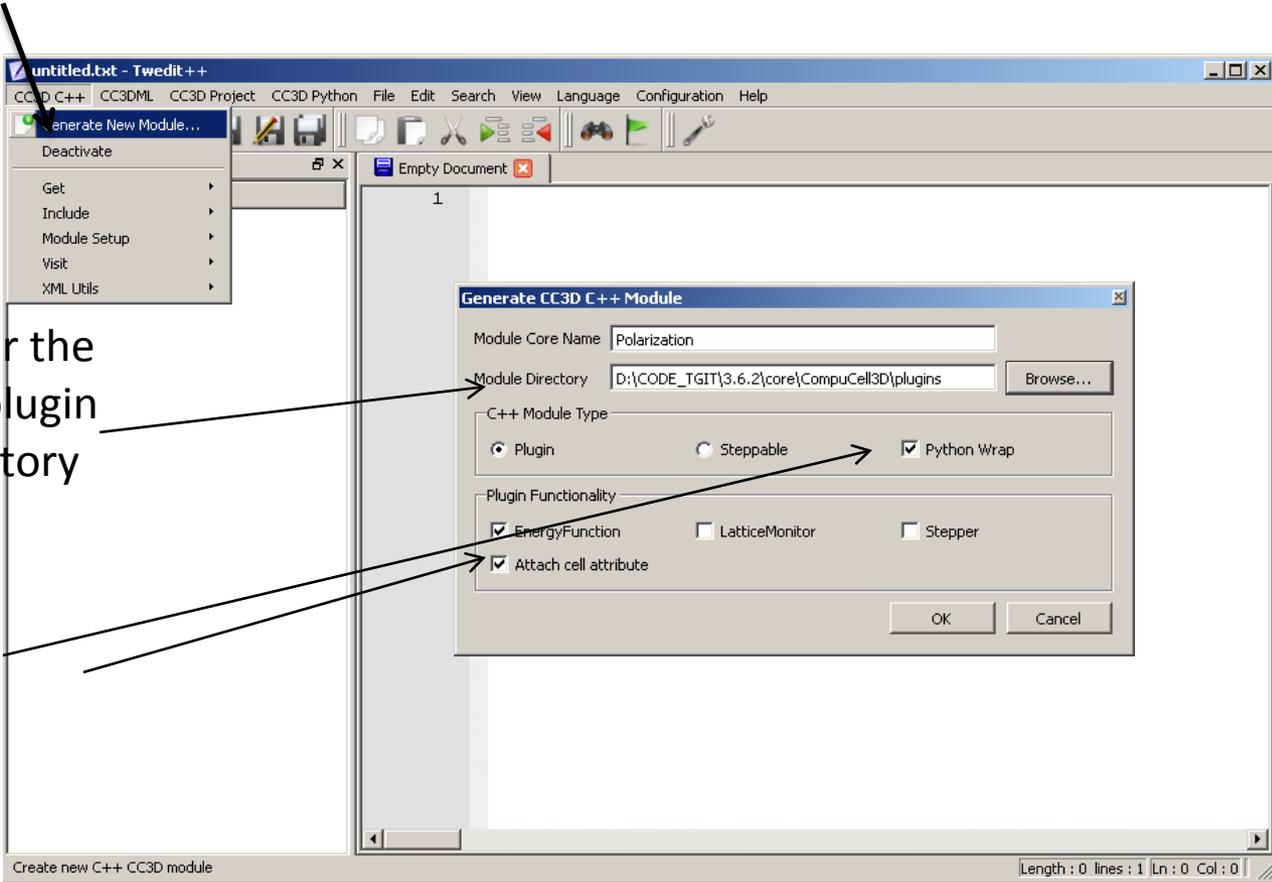
Generating CC3D C++ extension modules using Twedit++

- Writing plugin or steppable code in C++ is often a mechanistic process – at least when you begin. It can be also very tedious
- Up to 10 files need to be generated or modified
- Twedit++ autogenerates a **working** plugin or steppable for you freeing you from error-prone manual process
- Twedit++ has C++ code assistant which is not as comprehensive as Python or CC3DML assistants but we can certainly change it. Your input is greatly appreciated
- C++ Code Generated in Twedit can be then edited in any editor you wish including Twedit++

Generating CC3D C++ extension module

After clicking OK on the Generate CC3D C++ Module dialog box Twedit++ will generate/modify about 10 files and open them as separate tabs.

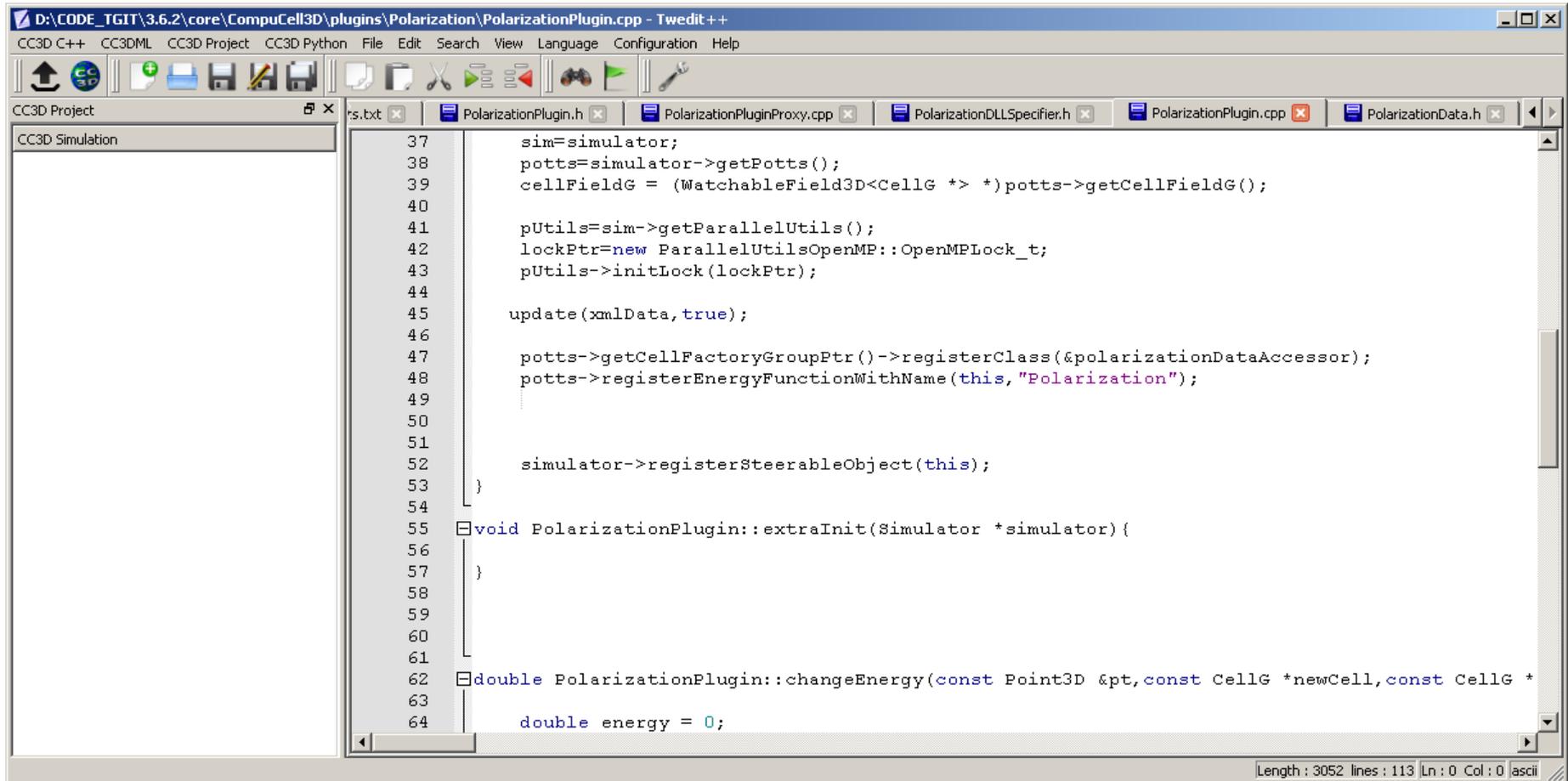
Go to CC3D C++ menu and choose Generate New Module



Pick appropriate directory for the module – here we develop plugin so we pick 'plugins' subdirectory in the CC3D source tree

You can attach extra cell attributes and make your module Python callable

Autogenerated CC3D C++ plugin code



The image shows a screenshot of a C++ IDE window titled "D:\CODE_TGIT\3.6.2\core\CompuCell3D\plugins\Polarization\PolarizationPlugin.cpp - Twedit++". The window contains a menu bar with options like "File", "Edit", "Search", "View", "Language", "Configuration", and "Help". Below the menu bar is a toolbar with various icons for file operations and development. The main editor area displays C++ code for the "PolarizationPlugin.cpp" file. The code includes several methods and initializations, such as "update(xmlData, true);", "registerClass(&polarizationDataAccessor);", and "registerEnergyFunctionWithName(this, \"Polarization\")". The code is line-numbered from 37 to 64. The status bar at the bottom right indicates "Length : 3052 lines : 113 Ln : 0 Col : 0 | ascii".

```
37     sim=simulator;
38     potts=simulator->getPotts();
39     cellFieldG = (WatchableField3D<CellG *> *)potts->getCellFieldG();
40
41     pUtils=sim->getParallelUtils();
42     lockPtr=new ParallelUtilsOpenMP::OpenMPLock_t;
43     pUtils->initLock(lockPtr);
44
45     update(xmlData, true);
46
47     potts->getCellFactoryGroupPtr()->registerClass(&polarizationDataAccessor);
48     potts->registerEnergyFunctionWithName(this, "Polarization");
49     ...
50
51
52     simulator->registerSteerableObject(this);
53 }
54
55 void PolarizationPlugin::extraInit(Simulator *simulator) {
56
57 }
58
59
60
61
62 double PolarizationPlugin::changeEnergy(const Point3D &pt, const CellG *newCell, const CellG *
63
64     double energy = 0;
```