E. ANNEX: PLUGINS' SOURCE CODE

Contents

This is a snapshot of source code available in https://github.com/zanfire/qgis-utils.

E.1 SOURCE CODE FOR DEFINES.PY FILE

```
  1:from qgis.core import QgsVectorLayer, QgsField, QgsMapLayerRegistry
  2:from PyQt4.QtCore import *
  3:
  4:"""
  5:This file contains declaration of fields and layers used by this plugin.
  6:
  7:"""
  8:
  9:"""
 10:Following used fields from input data.
 11:"""
 12:""" Used """
 13:FIELD_CODCAT                    = 'COD_CATAST'
 14:FIELD_VOLUME_HEIGHT             = 'UN_VOL_AV'
 15:FIELD_CADASTRE_TERRAIN_ID       = 'CHIAVE'
 16:FIELD_CADASTRE_USAGE            = 'USO'
 17:""" Used in AssignClassAction.py """
 18:FIELD_SEZ_ISTAT                 = 'SEZ2011'
 19:""" Used to get age in ISTAT CSV file. """
 20:FIELD_CSV_SEZ_AGE               = 'SEZ_AGE'
 21:
 22:
 23:FIELD_HEIGHT                    = 'HEIGHT'
 24:
 25:"""
 26:Field definition for generated layer.
 27:"""
 28:FIELD_ID_CADASTRE = 'ID_CAD'
 29:FIELD_ID_MEM      = 'ID_MEM' #9   identifier of the MEM building,  only buildings that presuma
bly use heating and/or cooling
 30:FIELD_USE         = 'USE' #3   residential (E1) or non-residential (En1) building
 31:FIELD_ID_EPC      = 'ID_EPC' #6   identifier of EPCs data     USE + codice catastale
 32:FIELD_TYPOLOGY    = 'TYPOLOGY' #8   identifier of compactness-age typology (serve compact_r e
age)  A1, A2,
 33:FIELD_ID_ISTAT    = 'ID_ISTAT' #8   identifier of ISTAT zone
 34:FIELD_ID_OMI      = 'ID_OMI' #6   identifier of OMI zone
 35:FIELD_FOOT_AREA       = 'FOOT_AREA' #4   footprint building area area
 36:FIELD_FLOOR_AREA  = 'FLOOR_AREA' #10      (sommatoria un-vol_MEM)
 37:FIELD_VOL_GROSS   = 'VOL_GROSS' #6       (sommatoria un-vol_MEM)
 38:FIELD_DISP_SURF   = 'DISP_SURF' #9   dispersing surface  (sommatoria un-vol_MEM)
 39:FIELD_COMPACT_R   = 'COMPACT_R' #9   S/V (considera tutte le un-vol_MEM)
 40:FIELD_WALL_SURF   = 'WALL_SURF' #9   walls surface   (sommatoria un-vol_MEM)
 41:FIELD_AGE         = 'AGE' #3       ID_EPC o ID_ISTAT   da fare per ID_EPC
 42:FIELD_WIND_R      = 'WIND_R' #6   window ratio (to dispersing surface)    ID_EPC o TYPOLOGY
wind_r da verificare per typology
 43:FIELD_WIND_SURF   = 'WIND_SURF' #9   window surface  disp_surf*wind_r
 44:FIELD_U_ENV       = 'U_ENV' #5   Envelope U-value    ID_EPC o TYPOLOGY
 45:FIELD_U_ROOF      = 'U_ROOF' #6   Roof U-value    ID_EPC o TYPOLOGY
 46:FIELD_U_GROUND    = 'U_GROUND' #8   Ground U-value  ID_EPC o TYPOLOGY
 47:FIELD_U_WIND      = 'U_WIND' #6   Window U-value  ID_EPC o TYPOLOGY
 48:FIELD_EPH         = 'EPH' #3   Primary Energy Heating  ID_EPC o TYPOLOGY
 49:FIELD_ETH         = 'ETH' #3   Thermal Energy Heating  ID_EPC o TYPOLOGY
 50:FIELD_ETC         = 'ETC' #3   Thermal Energy Cooling  ID_EPC o TYPOLOGY
 51:FIELD_EFER        = 'EFER' #4   Energy from RES ID_EPC o TYPOLOGY
 52:FIELD_EPW         = 'EPW' #3   Primary Energy DHW  ID_EPC o TYPOLOGY
 53:FIELD_EPT         = 'EPT' #3   Primary Energy Total    ID_EPC o TYPOLOGY
 54:FIELD_E_HEAT      = 'E_HEAT' #6   overall efficiency of the heating system    ID_EPC o TYPOLOG
Y
 55:FIELD_E_DHW       = 'E_DHW' #5   overall efficiency of the dhw system    ID_EPC o TYPOLOGY
 56:FIELD_E_H_DHW     = 'E_H_DHW' #7   overall efficiency of the heating & dhw system  ID_EPC o TY
POLOGY
 57:FIELD_PV_AREA      = 'PV_AREA' #6   surface of photovoltaic panels  ID_EPC  if not available u
se 0.
 58:FIELD_ST_AREA      = 'ST_AREA' #6   surface of solar thermal panels ID_EPC  if not available u
se 0.
 59:
 60:FIELD_AREA_GROSS = 'AREA_GROSS'
 61:FIELD_VOL_GROSS = 'VOL_GROSS'
 62:FIELD_WALL_SURF = 'WALL_SURF'
 63:FIELD_DISP_SURF = 'DISP_SURF'
 64:FIELD_AREA_R = 'AREA_R'
```

```
 65:FIELD_AREA_NET = 'AREA_NET'
 66:FIELD_VOL_R = 'VOL_R'
 67:FIELD_VOL_NET = 'VOL_NET'
 68:FIELD_H_LEVEL = 'H_LEVEL'
 69:FIELD_N_LEVEL = 'N_LEVEL'
 70:FIELD_FLOOR_AREA = 'FLOOR_AREA'
 71:
 72:"""
 73:Layer definition.
 74:"""
 75:LAYER_BUILDING_FIELD = [
 76:        QgsField(FIELD_ID_CADASTRE, QVariant.String),
 77:        QgsField(FIELD_ID_MEM, QVariant.String),
 78:        QgsField(FIELD_USE, QVariant.String),
 79:        QgsField(FIELD_ID_EPC, QVariant.String),
 80:        QgsField(FIELD_TYPOLOGY, QVariant.String),
 81:        QgsField(FIELD_ID_ISTAT, QVariant.String),
 82:        QgsField(FIELD_ID_OMI, QVariant.String),
 83:        QgsField(FIELD_FOOT_AREA, QVariant.Double),
 84:        QgsField(FIELD_FLOOR_AREA, QVariant.Double),
 85:        QgsField(FIELD_VOL_GROSS, QVariant.Double),
 86:        QgsField(FIELD_VOL_NET, QVariant.Double),
 87:        QgsField(FIELD_DISP_SURF, QVariant.Double),
 88:        QgsField(FIELD_COMPACT_R, QVariant.Double),
 89:        QgsField(FIELD_WALL_SURF, QVariant.Double),
 90:        QgsField(FIELD_AGE, QVariant.String),
 91:        QgsField(FIELD_WIND_R, QVariant.Double),
 92:        QgsField(FIELD_WIND_SURF, QVariant.Double),
 93:        QgsField(FIELD_U_ENV, QVariant.Double),
 94:        QgsField(FIELD_U_ROOF, QVariant.Double),
 95:        QgsField(FIELD_U_GROUND, QVariant.Double),
 96:        QgsField(FIELD_U_WIND, QVariant.Double),
 97:        QgsField(FIELD_EPH, QVariant.Double),
 98:        QgsField(FIELD_ETH, QVariant.Double),
 99:        QgsField(FIELD_ETC, QVariant.Double),
100:        QgsField(FIELD_EFER, QVariant.Double),
101:        QgsField(FIELD_EPW, QVariant.Double),
102:        QgsField(FIELD_EPT, QVariant.Double),
103:        QgsField(FIELD_E_HEAT, QVariant.Double),
104:        QgsField(FIELD_E_DHW, QVariant.Double),
105:        QgsField(FIELD_E_H_DHW, QVariant.Double),
106:        QgsField(FIELD_PV_AREA, QVariant.Double),
107:        QgsField(FIELD_ST_AREA, QVariant.Double)
108:        ]
109:
110:
111:"""
112:Layer definition.
113:"""
114:LAYER_VOLUMES_FIELDS = [
115:        QgsField(FIELD_ID_CADASTRE, QVariant.String),
116:        #un-vol_MEM  identifier of the MEM volumetric unit
117:        QgsField(FIELD_USE, QVariant.String), # residential (E1) or non-residential (En1) buil
ding
118:        QgsField(FIELD_ID_EPC, QVariant.String), # identifier of EPCs data
119:        QgsField(FIELD_TYPOLOGY, QVariant.String),
120:        #TYPOLOGY    identifier of compactness-age typology
121:        #ID_ISTAT    identifier of ISTAT zone
122:        #ID_OMI  identifier of OMI zone
123:        QgsField(FIELD_ID_MEM, QVariant.String),     # identifier of the MEM building belongs
to
124:        QgsField(FIELD_HEIGHT, QVariant.Double),  # height of the volumetric unit
125:        QgsField(FIELD_AREA_GROSS, QVariant.Double),  # gross area
126:        QgsField(FIELD_VOL_GROSS, QVariant.Double),   # gross volume
127:        QgsField(FIELD_WALL_SURF, QVariant.Double),   # walls surface
128:        QgsField(FIELD_DISP_SURF, QVariant.Double),   # dispersing surface
129:        QgsField(FIELD_AREA_R, QVariant.Double),  # net area to gross area ratio
130:        QgsField(FIELD_AREA_NET, QVariant.Double),    # net area
131:        QgsField(FIELD_VOL_R, QVariant.Double),   # net volume to gross volume ratio
132:        QgsField(FIELD_VOL_NET, QVariant.Double), # net volume
133:        QgsField(FIELD_H_LEVEL, QVariant.Double), # # average level's height
134:        QgsField(FIELD_N_LEVEL, QVariant.Int), # # number of levels
```

```
135:        QgsField(FIELD_FLOOR_AREA, QVariant.Double)  # #net floor area
136:        ]
```

E.2 SOURCE CODE FOR GJKO.PY FILE

```
  1:# Import the PyQt and QGIS libraries
  2:from PyQt4.QtCore import *
  3:from PyQt4.QtGui import *
  4:from qgis.core import *
  5:from actions import *
  6:from resources import *
  7:
  8:class Gjko:
  9:    """
 10:    This is the entry point of Gjko-plugin.
 11:    QGIS will load this class and call different interface method on this class.
 12:
 13:    Internally this class load two set of Action. The main action that are the main purpose of
this plugin
 14:    and tools action.
 15:    """
 16:
 17:    """ Menus """
 18:    menu = None
 19:    tool_menu = None
 20:    """ Main actions set. """
 21:    actions = []
 22:    """ Tool actions set. """
 23:    tool_actions = []
 24:
 25:    def __init__(self, iface):
 26:        """
 27:        Constructor of Gjko plugin.
 28:
 29:        :param iface interface to QGIS instance.
 30:        """
 31:
 32:        menuName = "&Municipal Energy Model"
 33:        self.menu = QMenu(iface.mainWindow())
 34:        self.menu.setObjectName("gjkoMenu")
 35:        self.menu.setTitle(menuName)
 36:        menuToolName = "&Municipal Energy Model - tools"
 37:        self.tool_menu = QMenu(iface.mainWindow())
 38:        self.tool_menu.setObjectName("gjkoToolMenu")
 39:        self.tool_menu.setTitle(menuToolName)
 40:
 41:        # Save reference to the QGIS interface
 42:        self.iface = iface
 43:        self.actions.append(SpatialJoinAction(iface, menuName))
 44:        self.actions.append(ComputeCompactRatioAction(iface, menuName))
 45:        self.actions.append(AssignClassAction(iface, menuName))
 46:        self.tool_actions.append(ManualCheckAction(iface, menuToolName))
 47:        self.tool_actions.append(SpatialJoinMaxAreaAction(iface, menuToolName))
 48:
 49:    def initGui(self):
 50:        for a in self.actions:
 51:            a.load()
 52:            self.menu.addAction(a.action)
 53:        for a in self.tool_actions:
 54:            a.load()
 55:            self.tool_menu.addAction(a.action)
 56:
 57:    def unload(self):
 58:         for a in self.actions:
 59:            a.unload()
 60:         for a in self.tool_actions:
 61:            a.unload()
```

E.3 SOURCE CODE FOR ACTIONS/ACTION.PY FILE

```
  1:# Import the PyQt and QGIS libraries
  2:from PyQt4.QtCore import *
  3:from PyQt4.QtGui import *
```

```
 4:from qgis.core import *
 5:from qgis.gui import *
 6:from ..util import pyqt_helper
 7:from ..resources import *
 8:import traceback
 9:import time
10:
11:class Worker(QtCore.QObject):
12:
13:    finished = QtCore.pyqtSignal(object)
14:    error = QtCore.pyqtSignal(Exception, basestring)
15:    progress = QtCore.pyqtSignal(float)
16:    inerror = False
17:
18:    def __init__(self, action):
19:        QtCore.QObject.__init__(self)
20:        self.action = action
21:        self.killed = False
22:
23:    def process(self):
24:        try:
25:            QgsMessageLog.logMessage("Starting processing ...")
26:            self.action.initialize()
27:            #time.sleep(2) # simulate a more time consuming task
28:            #self.progress.emit(55)
29:            #time.sleep(2)
30:            self.action.compute(self.progress)
31:            QgsMessageLog.logMessage("Terminated processing ...")
32:        except Exception, e:
33:            self.inerror = True
34:            self.error.emit(e, traceback.format_exc())
35:        self.finished.emit(None)
36:
37:    def kill(self):
38:        self.killed = True
39:
40:
41:class Action(object):
42:    """
43:    Define the abstract implementation of an action.
44:    """
45:    #dlg = None
46:
47:    def __init__(self, iface, menu_name, name):
48:        self.iface = iface
49:        self.menu = menu_name
50:        self.action = QAction(QIcon(":/plugins/Gjko/icon.png"), name, self.iface.mainWindow())
51:        QObject.connect(self.action, SIGNAL("activated()"), self.run)
52:
53:    def load(self):
54:        #self.iface.addToolBarIcon(self.action)
55:        self.iface.addPluginToMenu(self.menu, self.action)
56:
57:    def unload(self):
58:        self.iface.removePluginMenu(self.menu, self.action)
59:        #self.iface.removeToolBarIcon(self.action)
60:
61:    def create_dialog(self):
62:        return None
63:
64:    def initialize(self):
65:        return
66:
67:    def compute(self, progress):
68:        return
69:
70:    def run(self):
71:        self.dlg = self.create_dialog()
72:        if self.dlg == None:
73:            QgsMessageLog.logMessage("No dialog available!")
74:            # Message an error due to missing implementation.
75:            return
76:        self.dlg.show()
```

```
 77:         result = self.dlg.exec_()
 78:         if result == 1:
 79:             self.start_worker()
 80:             #self.apply()
 81:
 82:
 83:     def start_worker(self):
 84:         # create a new worker instance
 85:         worker = Worker(self)
 86:
 87:         (self.message_bar, progressBar) = pyqt_helper.create_progress_bar(self.iface, "Computi
ng information ...", worker)
 88:         # start the worker in a new thread
 89:         thread = QtCore.QThread(worker)
 90:         worker.moveToThread(thread)
 91:
 92:         worker.finished.connect(self.worker_finished)
 93:         worker.error.connect(self.worker_error)
 94:         worker.progress.connect(progressBar.setValue)
 95:
 96:         thread.started.connect(worker.process)
 97:         thread.start()
 98:         self.thread = thread
 99:         self.worker = worker
100:
101:
102:     def worker_finished(self, ret):
103:         # clean up the worker and thread
104:         ##self.worker.deleteLater()
105:         #self.thread.quit()
106:         #self.thread.wait()
107:         #self.thead.deleteLater()
108:         # remove widget from message bar
109:         self.iface.messageBar().popWidget(self.message_bar)
110:         if not self.worker.inerror:
111:             self.apply()
112:
113:     def worker_error(self, e, exception_string):
114:         QgsMessageLog.logMessage('Worker thread raised an exception:' + exception_string, leve
l=QgsMessageLog.CRITICAL)
115:         self.iface.messageBar().pushMessage('An error was occurred, exception:' + str(e), QgsM
essageBar.CRITICAL)
116:
```

E.4 SOURCE CODE FOR ACTIONS/ASSIGNCLASSACTION.PY FILE

```
  1:# Import the PyQt and QGIS libraries
  2:from PyQt4.QtCore import *
  3:from PyQt4.QtGui import *
  4:from qgis.core import *
  5:import math
  6:
  7:from Action import Action
  8:from ..dialogs import AssignClassDialog
  9:from ..util import layer_helper, reader_csv
 10:from ..DEFINES import *
 11:from ..logic import mem, code_generator
 12:
 13:class AssignClassAction(Action):
 14:     def __init__(self, iface, menu_name):
 15:         super(AssignClassAction, self).__init__(iface, menu_name, "3 - Assign EPC and Typology
")
 16:
 17:     def create_dialog(self):
 18:         return AssignClassDialog()
 19:
 20:
 21:     def initialize(self):
 22:         self.volumes_layer = layer_helper.get_layer(self.dlg.volumes_layer_name())
 23:         self.building_layer = layer_helper.get_layer(self.dlg.building_layer_name())
 24:         self.istat_layer = layer_helper.get_layer(self.dlg.istat_layer_name())
 25:         self.istat_csv = reader_csv.ISTAT(self.dlg.istat_csv_file())
 26:         self.epcs_csv = reader_csv.EPCs(self.dlg.epcs_csv_file())
```

```
 27:         self.typology_csv = reader_csv.EPCs(self.dlg.typology_csv_file())
 28:
 29:     def compute(self, progress):
 30:         self.compute_volumes(progress)
 31:         self.compute_building(progress)
 32:
 33:     def compute_building(self, progress):
 34:         QgsMessageLog.logMessage("Starting compute building ...")
 35:         istat_features = layer_helper.load_features(self.istat_layer)
 36:         building_features = layer_helper.load_features(self.building_layer)
 37:         index = layer_helper.build_spatialindex(istat_features.values())
 38:
 39:         self.updated_building_features = []
 40:         count = 0
 41:         count_max = len(building_features.values())
 42:         for f in building_features.values():
 43:             progress.emit(50 + int(count * (50.0 / count_max)))
 44:             count += 1
 45:             ids = index.intersects(f.geometry().boundingBox())
 46:             # guess the ISTAT code that have biggest area in this feature.
 47:             id_max = -1
 48:             area_max = -1
 49:             for i in ids:
 50:                 common = QgsGeometry(f.geometry().intersection(istat_features[i].geometry()))
 51:                 if common.area() > area_max:
 52:                     id_max = i
 53:                     area_max = common.area()
 54:             if id_max > -1:
 55:                 codistat = str(int(istat_features[id_max][FIELD_SEZ_ISTAT]))
 56:                 f[FIELD_ID_ISTAT] = codistat
 57:                 f[FIELD_AGE] = self.istat_csv.get_element(codistat, FIELD_CSV_SEZ_AGE)
 58:                 f[FIELD_TYPOLOGY] = code_generator.get_code(f[FIELD_USE], f[FIELD_AGE], f[FIEL
D_COMPACT_R])
 59:                 id_mem = f[FIELD_ID_MEM]
 60:                 if id_mem in self.idmem_to_volumes.keys():
 61:                     for vol in self.idmem_to_volumes[id_mem]:
 62:                         vol[FIELD_TYPOLOGY] = f[FIELD_TYPOLOGY]
 63:                         self.assign_volumes_values(vol, self.typology_csv, f[FIELD_TYPOLOGY])
 64:                 else:
 65:                     QgsMessageLog.logMessage("id_mem missing ... " + id_mem)
 66:
 67:                 id_epc = reader_csv.codcat_to_epcs(f[FIELD_USE], f[FIELD_ID_CADASTRE])
 68:                 epcs = self.epcs_csv.get(id_epc)
 69:                 if epcs == None:
 70:                     self.assign_building_values(f, self.typology_csv, f[FIELD_TYPOLOGY])
 71:                 if epcs != None:
 72:                     f[FIELD_ID_EPC] = id_epc
 73:                     epc_age = self.epcs_csv.get_element(id_epc, 'age')
 74:                     if epc_age != None and epc_age != '' and epc_age != f[FIELD_AGE]:
 75:                         f[FIELD_AGE] = epc_age
 76:                         f[FIELD_TYPOLOGY] = code_generator.get_code(f[FIELD_USE], f[FIELD_AGE]
, f[FIELD_COMPACT_R])
 77:
 78:                     self.assign_building_values(f, self.epcs_csv, id_epc)
 79:
 80:                 f[FIELD_FLOOR_AREA] = 0
 81:                 f[FIELD_VOL_NET] = 0
 82:                 for vol in self.idmem_to_volumes[f[FIELD_ID_MEM]]:
 83:                     try:
 84:                         f[FIELD_FLOOR_AREA] += vol[FIELD_FLOOR_AREA]
 85:                         f[FIELD_VOL_NET] += vol[FIELD_VOL_NET]
 86:                     except:
 87:                         QgsMessageLog.logMessage("Same id_mem but different epc (different USO
???)")
 88:                 self.updated_building_features.append(f)
 89:
 90:     def compute_volumes(self, progress):
 91:         volumes_features = layer_helper.load_features(self.volumes_layer)
 92:
 93:         self.idmem_to_volumes = {}
 94:         self.updated_volumes_features = []
 95:         count = 0
 96:         count_max = len(volumes_features.values())
```

```
 97:         for f in volumes_features.values():
 98:             progress.emit(int(count * (50.0 / count_max)))
 99:             count += 1
100:             id_epc = reader_csv.codcat_to_epcs(f[FIELD_USE], f[FIELD_ID_CADASTRE])
101:             epcs = self.epcs_csv.get(id_epc)
102:             if epcs != None:
103:                 f[FIELD_ID_EPC] = id_epc
104:                 self.assign_volumes_values(f, self.epcs_csv, id_epc)
105:             self.updated_volumes_features.append(f)
106:             # Update map with all features not only with EP# Update map with all features not
only with EPCC
107:             id_mem = f[FIELD_ID_MEM]
108:             if id_mem in self.idmem_to_volumes.keys():
109:                 self.idmem_to_volumes[id_mem].append(f)
110:             else:
111:                 self.idmem_to_volumes[id_mem] = [ f ]
112:
113:     def apply(self):
114:         self.building_layer.startEditing()
115:         for f in self.updated_building_features:
116:             self.building_layer.updateFeature(f)
117:         self.building_layer.commitChanges()
118:         self.volumes_layer.startEditing()
119:         for f in self.updated_volumes_features:
120:             self.volumes_layer.updateFeature(f)
121:         self.volumes_layer.commitChanges()
122:
123:         return None
124:
125:     def assign_volumes_values(self, f, csv, id_elem):
126:         try:
127:             f[FIELD_AREA_R] = csv.get_element(id_elem, 'area_r')
128:             f[FIELD_VOL_R] = csv.get_element(id_elem, 'vol_r')
129:             f[FIELD_H_LEVEL] = csv.get_element(id_elem, 'h_level')
130:             f[FIELD_AREA_NET] = float(f[FIELD_AREA_GROSS]) * float(f[FIELD_AREA_R])
131:             f[FIELD_VOL_NET] = float(f[FIELD_VOL_GROSS]) * float(f[FIELD_VOL_R])
132:             n_level = float(f[FIELD_HEIGHT]) / float(f[FIELD_H_LEVEL])
133:             if math.modf(n_level) > 0.8:
134:                 n_level = math.ceil(n_level)
135:             else:
136:                 n_level = math.floor(n_level)
137:             if n_level < 1:
138:                 n_level = 1
139:             f[FIELD_N_LEVEL] = n_level
140:             f[FIELD_FLOOR_AREA] = float(f[FIELD_AREA_NET]) * float(f[FIELD_N_LEVEL])
141:         except:
142:             QgsMessageLog.logMessage("Error for " + id_elem + " no values.")
143:
144:
145:     def assign_building_values(self, f, csv, id_elem):
146:         f[FIELD_WIND_R] = csv.get_element(id_elem, 'wind_r')
147:         f[FIELD_WIND_SURF] = csv.get_element(id_elem, 'wind_surf')
148:         f[FIELD_U_ENV] = csv.get_element(id_elem, 'u_env')
149:         f[FIELD_U_ROOF] = self.epcs_csv.get_element(id_elem, 'u_roof')
150:         f[FIELD_U_GROUND] = self.epcs_csv.get_element(id_elem, 'u_ground')
151:         f[FIELD_U_WIND] = csv.get_element(id_elem, 'u_wind')
152:         f[FIELD_EPH] = csv.get_element(id_elem, 'eph')
153:         f[FIELD_ETH] = csv.get_element(id_elem, 'eth')
154:         f[FIELD_ETC] = csv.get_element(id_elem, 'etc')
155:         f[FIELD_EFER] = csv.get_element(id_elem, 'efer')
156:         f[FIELD_EPW] = csv.get_element(id_elem, 'epw')
157:         f[FIELD_EPT] = csv.get_element(id_elem, 'ept')
158:         f[FIELD_E_HEAT] = csv.get_element(id_elem, 'e_heat')
159:         f[FIELD_E_DHW] = csv.get_element(id_elem, 'e_dhw')
160:         f[FIELD_E_H_DHW] = csv.get_element(id_elem, 'e_h_dhw')
161:         f[FIELD_PV_AREA] = csv.get_element(id_elem, 'fv_area', 0)
162:         f[FIELD_ST_AREA] = csv.get_element(id_elem, 'st_area', 0)
163:
```

E.5 SOURCE CODE FOR ACTIONS/COMPUTECOMPACTRATIOACTION.PY FILE

```
1:# Import the PyQt and QGIS libraries
2:from PyQt4.QtCore import *
```

```
  3:from PyQt4.QtGui import *
  4:from qgis.core import *
  5:import time
  6:
  7:from Action import Action
  8:from ..dialogs import ComputeCompactRatioDialog, ProgressDialog
  9:from ..util import layer_helper
 10:from ..DEFINES import *
 11:from ..logic import mem
 12:
 13:class ComputeCompactRatioAction(Action):
 14:    """
 15:    Compute energy efficency base value.
 16:    """
 17:
 18:    intersection_features = None
 19:
 20:    def __init__(self, iface, menu_name):
 21:        super(ComputeCompactRatioAction, self).__init__(iface, menu_name, "2 - Create energy l
ayers")
 22:
 23:    def create_dialog(self):
 24:        return ComputeCompactRatioDialog()
 25:
 26:    def initialize(self):
 27:        self.input_layer = layer_helper.get_layer(self.dlg.input_layer_name())
 28:        self.volumes_layer = layer_helper.create_layer(self.dlg.volumes_layer_name(), LAYER_VO
LUMES_FIELDS, self.input_layer, 'Polygon', False)
 29:        self.building_layer = layer_helper.create_layer(self.dlg.building_layer_name(), LAYER_
BUILDING_FIELD, self.input_layer, 'Polygon', False)
 30:
 31:    def compute(self, progress):
 32:        """
 33:        Creating final layer trough an intermediate layer.
 34:        """
 35:        t1 = time.clock()
 36:        features = self.input_layer.getFeatures()
 37:        features_id = layer_helper.load_features(self.input_layer)
 38:        index = layer_helper.build_spatialindex(features_id.values())
 39:        map_cadastre_building = {}
 40:        self.volumes_features = self.compute_volumes(progress, features, index, features_id, m
ap_cadastre_building)
 41:        t2 = time.clock()
 42:        self.building_features = self.compute_building(progress, map_cadastre_building)
 43:        if self.dlg.create_intersection_layer_check():
 44:            self.create_intersection_layer(index, features_id)
 45:        t3 = time.clock()
 46:        QgsMessageLog.logMessage("Performance t1 " + str(t2 - t1) + ", t2 " + str(t3 - t2))
 47:
 48:    def compute_volumes(self, progress, features, index, features_id, map_cadastre_building):
 49:        result = []
 50:        # Filling working/temporary layer.
 51:        count = 0
 52:        count_max = len(features_id.values())
 53:        for f in features:
 54:            count += 1
 55:            progress.emit(int(count * (50.0 / count_max)))
 56:            feature = QgsFeature(self.volumes_layer.pendingFields())
 57:            g = f.geometry()
 58:            feature.setGeometry(QgsGeometry(g))
 59:            #feature.setGeometry(QgsGeometry(Layer_helper.copy_geometry(f)))
 60:            feature[FIELD_ID_CADASTRE] = f[FIELD_CODCAT]
 61:            feature[FIELD_USE] = f[FIELD_CADASTRE_USAGE]
 62:            feature[FIELD_HEIGHT] = f[FIELD_VOLUME_HEIGHT]
 63:            feature[FIELD_AREA_GROSS] = g.area()
 64:            feature[FIELD_VOL_GROSS] = feature[FIELD_HEIGHT] * feature[FIELD_AREA_GROSS]
 65:            feature[FIELD_WALL_SURF] = mem.compute_external_wall_surface(index, g, features_id
, feature[FIELD_HEIGHT])
 66:            feature[FIELD_DISP_SURF] = feature[FIELD_AREA_GROSS] * 2 + feature[FIELD_WALL_SURF
]
 67:            #feature[FIELD_AREA_R] =
 68:            #feature[FIELD_AREA_NET] =
 69:            #feature[FIELD_VOL_R] =
```

```
 70:             #feature[FIELD_VOL_NET] =
 71:             #feature[FIELD_H_LEVEL] =
 72:             #feature[FIELD_N_LEVEL] =
 73:             #feature[FIELD_FLOOR_AREA] =
 74:
 75:             result.append(feature)
 76:             # Create map cadastre to features for next steps.
 77:             id_cadastre = feature[FIELD_USE] + '-' + f[FIELD_CODCAT]
 78:             if not id_cadastre in map_cadastre_building.keys():
 79:                 map_cadastre_building[id_cadastre] = [ feature ]
 80:             else:
 81:                 map_cadastre_building[id_cadastre].append(feature)
 82:         return result
 83:
 84:
 85:
 86:     def compute_building(self, progress, map_cadastre_building):
 87:         #features_id = layer_helper.load_features(l)
 88:         #index = layer_helper.build_spatialindex(features_id.values())
 89:
 90:         result = []
 91:         count = 0
 92:         count_max = len(map_cadastre_building.keys())
 93:         for key in map_cadastre_building.keys():
 94:             count += 1
 95:             progress.emit(50 + int(count * (50.0 / count_max)))
 96:             features_temp = [ ]
 97:             idx = 0
 98:             for f in map_cadastre_building[key]:
 99:                 insert = True
100:                 (geom, feature_set) = mem.merge(f, map_cadastre_building[key])
101:                 for fe in features_temp:
102:                     if not geom.disjoint(fe.geometry()) or geom.equals(fe.geometry()) or geom.
contains(fe.geometry()) or geom.within(fe.geometry()):
103:                         insert = False
104:                         break
105:                 if insert:
106:                     feature = QgsFeature(self.building_layer.pendingFields())
107:                     features_temp.append(feature);
108:                     feature.setGeometry(geom)
109:                     id_mem = key + '_' + str(idx)
110:                     feature[FIELD_ID_CADASTRE] = '-'.join(key.split('-')[1:])
111:                     feature[FIELD_ID_MEM] = id_mem
112:                     feature[FIELD_USE] = f[FIELD_USE]
113:
114:                     feature[FIELD_FOOT_AREA] = 0
115:                     feature[FIELD_VOL_GROSS] = 0
116:                     feature[FIELD_DISP_SURF] = 0
117:                     feature[FIELD_WALL_SURF] = 0
118:                     total_disp = 0
119:                     for elem in feature_set:
120:                         elem[FIELD_ID_MEM] = id_mem
121:                         feature[FIELD_FOOT_AREA] += elem[FIELD_AREA_GROSS]
122:                         feature[FIELD_VOL_GROSS] += elem[FIELD_VOL_GROSS]
123:                         feature[FIELD_DISP_SURF] += elem[FIELD_DISP_SURF]
124:                         feature[FIELD_WALL_SURF] += elem[FIELD_WALL_SURF]
125:                     if feature[FIELD_VOL_GROSS] > 0:
126:                         feature[FIELD_COMPACT_R] = feature[FIELD_DISP_SURF] / feature[FIELD_VO
L_GROSS]
127:                     idx += 1
128:             # We have created the final set.
129:             result.extend(features_temp)
130:         return result
131:
132:     def create_intersection_layer(self, index, features):
133:         self.intersection_layer = layer_helper.create_layer("Volumes_intersection", [], self.v
olumes_layer, 'LineString', False)
134:         new_features = []
135:         for f in features.values():
136:             g1 = f.geometry()
137:             ids = index.intersects(g1.boundingBox())
138:             for i in ids:
139:                 g2 = features[i].geometry()
```

```
140:                 if not g1.equals(g2):
141:                     intersection_set = mem.get_intersection_debug(g1, g2)
142:                     for intersection in intersection_set:
143:                         feature = QgsFeature()
144:                         feature.setGeometry(intersection)
145:                         new_features.append(feature)
146:         self.intersection_features = new_features
147: 
148:     def apply(self):
149:         if self.dlg.volumes_layer_path() != '':
150:             self.volumes_layer.startEditing()
151:             self.volumes_layer.dataProvider().addFeatures(self.volumes_features)
152:             self.volumes_layer.commitChanges()
153:             result = layer_helper.save_layer(self.volumes_layer, self.dlg.volumes_layer_path()
)
154:             if result == QgsVectorFileWriter.NoError:
155:                 layer_name = self.dlg.volumes_layer_name()
156:                 self.iface.addVectorLayer(self.dlg.volumes_layer_path(), layer_name, "ogr")
157: 
158:         if self.dlg.building_layer_path() != '':
159:             self.building_layer.startEditing()
160:             self.building_layer.dataProvider().addFeatures(self.building_features)
161:             self.building_layer.commitChanges()
162:             result = layer_helper.save_layer(self.building_layer, self.dlg.building_layer_path
())
163:             if result == QgsVectorFileWriter.NoError:
164:                 layer_name = self.dlg.building_layer_name()
165:                 self.iface.addVectorLayer(self.dlg.building_layer_path(), layer_name, "ogr")
166:         if self.intersection_features != None:
167:             self.intersection_layer.startEditing()
168:             self.intersection_layer.dataProvider().addFeatures(self.intersection_features)
169:             self.intersection_layer.commitChanges()
170:             QgsMapLayerRegistry.instance().addMapLayer(self.intersection_layer)
```

E.6 Source code for actions/CreateLayerAction.py file

```
  1:# Import the PyQt and QGIS libraries
  2:from PyQt4.QtCore import *
  3:from PyQt4.QtGui import *
  4:from qgis.core import *
  5: 
  6:from ..dialogs import CreateLayerDialog
  7:from ..util import layer_helper
  8:from Action import Action
  9:from ..DEFINES import *
 10: 
 11:class CreateLayerAction(Action):
 12:     def __init__(self, iface, menu_name):
 13:         super(CreateLayerAction, self).__init__(iface, menu_name, "Create working Layer")
 14: 
 15:     def run(self):
 16:         dlg = CreateLayerDialog()
 17:         dlg.show()
 18:         result = dlg.exec_()
 19:         # See if OK was pressed
 20:         if result == 1:
 21:             name = dlg.name()
 22:             if len(name) == 0:
 23:                 QMessageBox.critical(None, 'Error', 'Layer name is missing, abort operation.')
 24:                 return
 25:             location = dlg.location()
 26:             if len(location) == 0:
 27:                 QMessageBox.critical(None, 'Error', 'Save folder is missing, abort operation.'
)
 28:                 return
 29: 
 30:             layer = layer_helper.get_layer(name)
 31:             if layer != None:
 32:                 QMessageBox.critical(None, 'Error', 'Layer exists, abort operation.')
 33:                 return
 34: 
 35:             layer = layer_helper.create_layer(name, LAYER_NEIGHBORS_FIELDS)
 36:             if layer != None:
```

```
37:                 layer_helper.save_layer(layer, name, location)
38:             print("Completed.")
```

E.7 SOURCE CODE FOR ACTIONS/MANUALCHECKACTION.PY FILE

```
 1:# Import the PyQt and QGIS libraries
 2:from PyQt4.QtCore import *
 3:from PyQt4.QtGui import *
 4:from qgis.core import *
 5:
 6:from ..dialogs import ManualCheckDialog
 7:from ..util import layer_helper
 8:from Action import Action
 9:from ..DEFINES import *
10:
11:class ManualCheckAction(Action):
12:    def __init__(self, iface, menu_name):
13:        super(ManualCheckAction, self).__init__(iface, menu_name, "Navigate through features..
.")
14:
15:    def run(self):
16:        dlg = ManualCheckDialog()
17:        dlg.show()
18:        result = dlg.exec_()
```

E.8 SOURCE CODE FOR ACTIONS/SPATIALJOINACTION.PY FILE

```
 1:# Import the PyQt and QGIS libraries
 2:from PyQt4.QtCore import *
 3:from PyQt4.QtGui import *
 4:from qgis.core import *
 5:import os
 6:import time
 7:
 8:from Action import Action
 9:from ..dialogs import SpatialJoinDialog
10:from ..util import layer_helper, reader_csv
11:from ..DEFINES import *
12:from ..logic import mem, code_generator
13:
14:class SpatialJoinAction(Action):
15:    def __init__(self, iface, menu_name):
16:        super(SpatialJoinAction, self).__init__(iface, menu_name, "1 - Assign ID_CAD")
17:
18:    def create_dialog(self):
19:        return SpatialJoinDialog()
20:
21:    def initialize(self):
22:        self.volumes_layer = layer_helper.get_layer(self.dlg.volumes_layer_name())
23:        self.cadastre_layer = layer_helper.get_layer(self.dlg.cadastre_layer_name())
24:        self.cadastre_terrain_layer = layer_helper.get_layer(self.dlg.cadastre_terrain_layer_n
ame())
25:        self.attributes = []
26:        self.fields = QgsFields()
27:        for attr in self.volumes_layer.pendingFields():
28:            field = QgsField(attr.name(), attr.type())
29:            self.attributes.append(field)
30:            self.fields.append(field)
31:        field = QgsField(FIELD_CODCAT, QVariant.String)
32:        self.attributes.append(field)
33:        self.fields.append(field)
34:
35:    def compute(self, progress):
36:        cadastre_features = layer_helper.load_features(self.cadastre_layer)
37:        index = layer_helper.build_spatialindex(cadastre_features.values())
38:        cadastre_terrain_features = layer_helper.load_features(self.cadastre_terrain_layer)
39:        index_cadastre_terrain = layer_helper.build_spatialindex(cadastre_terrain_features.val
ues())
40:
41:        self.new_features = []
42:        features = layer_helper.load_features(self.volumes_layer)
43:        count_max = len(features)
44:        count = 0
```

```
 45:         for f in features.values():
 46:             count += 1
 47:             progress.emit(int(count * (100.0 / count_max)))
 48:             feature = QgsFeature(QgsFields(self.fields))
 49:             idf = layer_helper.get_intersection_max_area(index, f, cadastre_features)
 50:             add = False
 51:             if idf >= 0:
 52:                 add = True
 53:                 feature[FIELD_CODCAT] = cadastre_features[idf][FIELD_CODCAT]
 54:             # Try with the cadastre terrain.
 55:             if not add:
 56:                 idf = layer_helper.get_intersection_max_area(index_cadastre_terrain, f, cadast
re_terrain_features)
 57:                 if idf >= 0:
 58:                     add = True
 59:                     feature[FIELD_CODCAT] = cadastre_terrain_features[idf][FIELD_CADASTRE_TERR
AIN_ID]
 60:             # Final we add this feature
 61:             if add:
 62:                 feature.setGeometry(QgsGeometry(f.geometry()))
 63:                 for attr in self.volumes_layer.pendingFields():
 64:                     feature[attr.name()] = f[attr.name()]
 65:                 self.new_features.append(feature)
 66:
 67:     def apply(self):
 68:         self.output_layer = layer_helper.create_layer(os.path.splitext(os.path.basename(self.d
lg.location()))[0], self.attributes, self.volumes_layer)
 69:         self.output_layer.startEditing()
 70:         self.output_layer.dataProvider().addFeatures(self.new_features)
 71:         self.output_layer.commitChanges()
 72:
 73:         if self.dlg.location() != '':
 74:             result = layer_helper.save_layer(self.output_layer, self.dlg.location())
 75:             if result == QgsVectorFileWriter.NoError:
 76:                 layer_name = self.output_layer.name()
 77:                 QgsMapLayerRegistry.instance().removeMapLayer(self.output_layer.id())
 78:                 self.output_layer = self.iface.addVectorLayer(self.dlg.location(), layer_name,
"ogr")
 79:             else:
 80:                 QgsMessageLog.logMessage("Failed to save layer, error: " + str(result))
```

E.9 SOURCE CODE FOR ACTIONS/SPATIALJOINMAXAREAACTION.PY FILE

```
  1:# Import the PyQt and QGIS libraries
  2:from PyQt4.QtCore import *
  3:from PyQt4.QtGui import *
  4:from qgis.core import *
  5:import os
  6:import time
  7:
  8:from Action import Action
  9:from ..dialogs import SpatialJoinMaxAreaDialog
 10:from ..util import layer_helper, reader_csv
 11:from ..DEFINES import *
 12:from ..logic import mem, code_generator
 13:
 14:class SpatialJoinMaxAreaAction(Action):
 15:     def __init__(self, iface, menu_name):
 16:         super(SpatialJoinMaxAreaAction, self).__init__(iface, menu_name, "Spatial join max are
a...")
 17:
 18:     def create_dialog(self):
 19:         return SpatialJoinMaxAreaDialog()
 20:
 21:     def initialize(self):
 22:         self.layer1 = layer_helper.get_layer(self.dlg.layer1_layer_name())
 23:         self.layer2 = layer_helper.get_layer(self.dlg.layer2_layer_name())
 24:         self.max_area_field = self.dlg.field_name()
 25:         self.attributes = []
 26:         self.fields = QgsFields()
 27:         for attr in self.layer1.pendingFields():
 28:             field = QgsField(attr.name(), attr.type())
 29:             self.attributes.append(field)
```

```
 30:             self.fields.append(field)
 31:         for attr in self.layer2.pendingFields():
 32:             if attr.name() == self.max_area_field:
 33:                 field = QgsField(attr.name(), attr.type())
 34:                 self.attributes.append(field)
 35:                 self.fields.append(field)
 36:                 break
 37:
 38:     def compute(self, progress):
 39:         features1 = layer_helper.load_features(self.layer1)
 40:         features2 = layer_helper.load_features(self.layer2)
 41:         index = layer_helper.build_spatialindex(features2.values())
 42:         #index2 = layer_helper.build_spatialindex(features2.values())
 43:
 44:         self.new_features = []
 45:         features = layer_helper.load_features(self.layer1)
 46:         count_max = len(features)
 47:         count = 0
 48:         for f in features.values():
 49:             count += 1
 50:             progress.emit(int(count * (100.0 / count_max)))
 51:             feature = QgsFeature(QgsFields(self.fields))
 52:             ids = index.intersects(f.geometry().boundingBox())
 53:             add = False
 54:             if len(ids) == 0:
 55:                 add = True
 56:             elif len(ids) == 1:
 57:                 add = True
 58:                 if not f.geometry().disjoint(features2[ids[0]].geometry()):
 59:                     feature[self.max_area_field] = features2[ids[0]][self.max_area_field]
 60:             else:
 61:                 id_max = -1
 62:                 area_max = -1
 63:                 for i in ids:
 64:                     if not f.geometry().disjoint(features2[i].geometry()):
 65:                         common = QgsGeometry(f.geometry().intersection(features2[i].geometry()
))
 66:                         if common.area() > area_max:
 67:                             id_max = i
 68:                             area_max = common.area()
 69:                 if id_max > -1:
 70:                     add = True
 71:                     feature[self.max_area_field] = features2[id_max][self.max_area_field]
 72:             # Final we add this feature
 73:             if add:
 74:                 feature.setGeometry(QgsGeometry(f.geometry()))
 75:                 # Determinate cadastre ID.
 76:                 for attr in self.layer1.pendingFields():
 77:                     feature[attr.name()] = f[attr.name()]
 78:                 self.new_features.append(feature)
 79:
 80:     def apply(self):
 81:         self.output_layer = layer_helper.create_layer(os.path.splitext(os.path.basename(self.d
lg.location()))[0], self.attributes, self.layer1)
 82:         self.output_layer.startEditing()
 83:         self.output_layer.dataProvider().addFeatures(self.new_features)
 84:         self.output_layer.commitChanges()
 85:
 86:         if self.dlg.location() != '':
 87:             result = layer_helper.save_layer(self.output_layer, self.dlg.location())
 88:             if result == QgsVectorFileWriter.NoError:
 89:                 layer_name = self.output_layer.name()
 90:                 QgsMapLayerRegistry.instance().removeMapLayer(self.output_layer.id())
 91:                 self.output_layer = self.iface.addVectorLayer(self.dlg.location(), layer_name,
"ogr")
 92:             else:
 93:                 QgsMessageLog.logMessage("Failed to save layer, error: " + str(result))
```

E.10 SOURCE CODE FOR DIALOGS/ASSIGNCLASSDIALOG.PY FILE

```
  1:from PyQt4 import QtCore, QtGui
  2:from qgis.utils import iface
  3:from Ui_AssignClass import Ui_Dialog
```

```
 4:import os
 5:
 6:def helper_selectcombo(combo, text):
 7:    index = combo.findText(text, QtCore.Qt.MatchContains)
 8:    if index >= 0:
 9:        combo.setCurrentIndex(index)
10:
11:class AssignClassDialog(QtGui.QDialog):
12:
13:    def __init__(self):
14:        QtGui.QDialog.__init__(self)
15:        self.ui = Ui_Dialog()
16:        self.ui.setupUi(self)
17:        layers = iface.legendInterface().layers()
18:        for l in layers:
19:            name = l.name()
20:            self.ui.volumesCombo.addItem(name)
21:            self.ui.buildingCombo.addItem(name)
22:            self.ui.istatCombo.addItem(name)
23:        # Guess data for convenience.
24:        helper_selectcombo(self.ui.volumesCombo, "volume")
25:        helper_selectcombo(self.ui.buildingCombo, "build")
26:        helper_selectcombo(self.ui.istatCombo, "istat")
27:
28:        QtCore.QObject.connect(self.ui.epcOpenButton, QtCore.SIGNAL('clicked()'), self.epc_ope
n_file)
29:        QtCore.QObject.connect(self.ui.typologyOpenButton, QtCore.SIGNAL('clicked()'), self.ty
pology_open_file)
30:        QtCore.QObject.connect(self.ui.istatOpenButton, QtCore.SIGNAL('clicked()'), self.istat
_open_file)
31:
32:
33:    def volumes_layer_name(self):
34:       return str(self.ui.volumesCombo.currentText())
35:
36:    def building_layer_name(self):
37:       return str(self.ui.buildingCombo.currentText())
38:
39:    def epcs_csv_file(self):
40:        return self.ui.epcEdit.text()
41:
42:    def typology_csv_file(self):
43:        return self.ui.typologyEdit.text()
44:
45:    def istat_csv_file(self):
46:        return self.ui.istatEdit.text()
47:
48:    def istat_layer_name(self):
49:        return str(self.ui.istatCombo.currentText())
50:
51:    def epc_open_file(self):
52:        f = self.open_file()
53:        self.ui.epcEdit.setText(f)
54:
55:    def typology_open_file(self):
56:        f = self.open_file()
57:        self.ui.typologyEdit.setText(f)
58:
59:    def istat_open_file(self):
60:        f = self.open_file()
61:        self.ui.istatEdit.setText(f)
62:
63:    def open_file(self):
64:        location = QtGui.QFileDialog.getOpenFileName(None, 'Open CSV file:', os.getenv('HOME')
, 'CSV (*.csv);; All files (*)')
65:        return location
66:
```

E.11 Source code for dialogs/ComputeCompactRatioDialog.py file

```
 1:from PyQt4 import QtCore, QtGui
 2:from qgis.utils import iface
 3:from Ui_ComputeCompactRatio import Ui_Dialog
```

```
 4:import os
 5:
 6:class ComputeCompactRatioDialog(QtGui.QDialog):
 7:
 8:    def __init__(self):
 9:        QtGui.QDialog.__init__(self)
10:        self.ui = Ui_Dialog()
11:        self.ui.setupUi(self)
12:        self.ui.volumesLayerPath.setText(os.path.join(os.getenv('HOME'), "Volumes.shp"))
13:        self.ui.buildingLayerPath.setText(os.path.join(os.getenv('HOME'), "Building.shp"))
14:        QtCore.QObject.connect(self.ui.locationButton1, QtCore.SIGNAL('clicked()'), self.save_
location_dialog1)
15:        QtCore.QObject.connect(self.ui.locationButton2, QtCore.SIGNAL('clicked()'), self.save_
location_dialog2)
16:        layers = iface.legendInterface().layers()
17:        for l in layers:
18:            self.ui.volumesCombo.addItem(l.name())
19:
20:    def save_location_dialog1(self):
21:        location = QtGui.QFileDialog.getSaveFileName(None, 'Shapefile file:', self.ui.volumesL
ayerPath.text(), 'Shp (*.shp);; All files (*)')
22:        if len(location) > 0:
23:            self.ui.volumesLayerPath.setText(location)
24:
25:    def save_location_dialog2(self):
26:        location = QtGui.QFileDialog.getSaveFileName(None, 'Shapefile file:', self.ui.building
LayerPath.text(), 'Shp (*.shp);; All files (*)')
27:        if len(location) > 0:
28:            self.ui.buildingLayerPath.setText(location)
29:
30:    def building_layer_path(self):
31:        return self.ui.buildingLayerPath.text()
32:
33:    def building_layer_name(self):
34:        return os.path.splitext(os.path.basename(self.ui.buildingLayerPath.text()))[0]
35:
36:    def volumes_layer_path(self):
37:        return self.ui.volumesLayerPath.text()
38:
39:    def volumes_layer_name(self):
40:        return os.path.splitext(os.path.basename(self.ui.volumesLayerPath.text()))[0]
41:
42:    def input_layer_name(self):
43:        return str(self.ui.volumesCombo.currentText())
44:
45:    def create_intersection_layer_check(self):
46:         return self.ui.intersectionLayerCheckBox.isChecked()
47:
```

E.12 SOURCE CODE FOR DIALOGS/CREATELAYERDIALOG.PY FILE

```
 1:from PyQt4 import QtCore, QtGui
 2:from Ui_CreateLayer import Ui_Dialog
 3:
 4:class CreateLayerDialog(QtGui.QDialog):
 5:    def __init__(self):
 6:        QtGui.QDialog.__init__(self)
 7:        self.ui = Ui_Dialog()
 8:        self.ui.setupUi(self)
 9:        QtCore.QObject.connect(self.ui.locationButton, QtCore.SIGNAL('clicked()'), self.openLo
cation)
10:
11:    def openLocation(self):
12:        location = QtGui.QFileDialog.getExistingDirectory(None, 'Select a folder:', '', QtGui.
QFileDialog.ShowDirsOnly)
13:        self.ui.saveFolder.setText(location)
14:
15:    def name(self):
16:        return self.ui.layerName.text()
17:
18:    def location(self):
19:        return self.ui.saveFolder.text()
20:
```

E.13 SOURCE CODE FOR DIALOGS/GJKODIALOG.PY FILE

```
1:from PyQt4 import QtCore, QtGui
2:from Ui_Gjko import Ui_Gjko
3:# create the dialog for Gjko
4:class GjkoDialog(QtGui.QDialog):
5:  def __init__(self):
6:    QtGui.QDialog.__init__(self)
7:    # Set up the user interface from Designer.
8:    self.ui = Ui_Gjko ()
9:    self.ui.setupUi(self)
```

E.14 SOURCE CODE FOR DIALOGS/MANUALCHECKDIALOG.PY FILE

```
1:from qgis.utils import iface
2:from PyQt4 import QtCore, QtGui
3:from Ui_ManualCheck import Ui_Dialog
4:from ..util import layer_helper
5:
6:class ManualCheckDialog(QtGui.QDialog):
7:    def __init__(self):
8:        QtGui.QDialog.__init__(self, None, QtCore.Qt.WindowStaysOnTopHint)
9:        self.ui = Ui_Dialog()
10:       self.ui.setupUi(self)
11:       #self.ui.Dialog.setWindowFlags(QtCore.Qt.WindowStaysOnTopHint)
12:       # Fill data
13:       QtCore.QObject.connect(self.ui.previousButton, QtCore.SIGNAL('clicked()'), self.onPrev
)
14:       QtCore.QObject.connect(self.ui.currentButton, QtCore.SIGNAL('clicked()'), self.onCurre
nt)
15:       QtCore.QObject.connect(self.ui.nextButton, QtCore.SIGNAL('clicked()'), self.onNext)
16:       QtCore.QObject.connect(self.ui.layersCombo, QtCore.SIGNAL('currentIndexChanged(QString
)'), self.onLayerChanged)
17:       QtCore.QObject.connect(self.ui.currentIndexText, QtCore.SIGNAL('textEdited(QString)'),
self.onIndexChanged)
18:
19:       layers = iface.legendInterface().layers()
20:       for l in layers:
21:           self.ui.layersCombo.addItem(l.name())
22:
23:       self.layerName = None
24:
25:   def onNext(self):
26:       if self.layerName == None:
27:           self.initLayerNavigation()
28:       if self.currentFeatureIdx >= (len(self.features) - 1):
29:           return
30:       self.currentFeatureIdx += 1
31:       self.ui.currentIndexText.setText(str(self.currentFeatureIdx + 1))
32:       layer_helper.show_features(self.layer, [ self.features[self.currentFeatureIdx]])
33:
34:   def onCurrent(self):
35:       if self.layerName == None:
36:           return
37:       if self.currentFeatureIdx < 0:
38:           return
39:       layer_helper.show_features(self.layer, [ self.features[self.currentFeatureIdx]])
40:       self.ui.currentIndexText.setText(str(self.currentFeatureIdx + 1))
41:
42:   def onPrev(self):
43:       if self.layerName == None:
44:           return
45:       if self.currentFeatureIdx <= 0:
46:           return
47:       self.currentFeatureIdx -= 1
48:       layer_helper.show_features(self.layer, [ self.features[self.currentFeatureIdx]])
49:       self.ui.currentIndexText.setText(str(self.currentFeatureIdx + 1))
50:
51:   def onLayerChanged(self, name):
52:       self.initLayerNavigation()
53:       self.onCurrent()
54:
55:   def onIndexChanged(self, value):
```

```
56:         if len(value) == 0:
57:             return
58:         if not str(value).isdigit():
59:             return
60:         intvalue = int(value) - 1
61:         if intvalue >= (len(self.features) - 1):
62:             return
63:
64:         self.currentFeatureIdx = intvalue
65:         self.onCurrent()
66:
67:     def initLayerNavigation(self):
68:         self.layerName = str(self.ui.layersCombo.currentText())
69:         self.layer = layer_helper.get_layer(self.layerName)
70:         iface.setActiveLayer(self.layer)
71:         self.features = []
72:         for f in self.layer.getFeatures():
73:             self.features.append(f)
74:         self.ui.maxLabel.setText(" of " + str(len(self.features)))
75:         self.currentFeatureIdx = 0
```

E.15 Source code for dialogs/ProgressDialog.py file

```
1:from PyQt4 import QtCore, QtGui
2:from qgis.utils import iface
3:from Ui_Progress import Ui_Dialog
4:
5:class ProgressDialog(QtGui.QDialog):
6:
7:    def __init__(self):
8:        QtGui.QDialog.__init__(self)
9:        self.ui = Ui_Dialog()
10:       self.ui.setupUi(self)
11:
12:   def set_max(self, max):
13:       self.max = max
14:       self.ui.label.setText('0 / ' + str(max))
15:       self.ui.progressBat.setMaximum(max)
16:
17:   def set_current(self, current):
18:       self.ui.labelsetText(current + ' / ' + str(self.max))
19:       self.ui.progressBar.setValue(current)
20:
```

E.16 Source code for dialogs/SpatialJoinDialog.py file

```
1:from PyQt4 import QtCore, QtGui
2:from qgis.utils import iface
3:from Ui_SpatialJoinCadastre import Ui_Dialog
4:import os
5:
6:class SpatialJoinDialog(QtGui.QDialog):
7:
8:    def __init__(self):
9:        QtGui.QDialog.__init__(self)
10:       self.ui = Ui_Dialog()
11:       self.ui.setupUi(self)
12:       self.ui.layerName.setText(os.path.join(os.getenv('HOME'), "SpatialJoin.shp"))
13:       QtCore.QObject.connect(self.ui.locationButton, QtCore.SIGNAL('clicked()'), self.save_l
ocation_dialog)
14:       layers = iface.legendInterface().layers()
15:       for l in layers:
16:           self.ui.volumesCombo.addItem(l.name())
17:           self.ui.cadastreCombo.addItem(l.name())
18:           self.ui.cadastreTerrainCombo.addItem(l.name())
19:
20:   def save_location_dialog(self):
21:       location = QtGui.QFileDialog.getSaveFileName(None, 'Shapefile file:', self.ui.layerNam
e.text(), 'Shp (*.shp);; All files (*)')
22:       if len(location) > 0:
23:           self.ui.layerName.setText(location)
24:
25:   def location(self):
```

```
        return self.ui.layerName.text()

    def volumes_layer_name(self):
        return str(self.ui.volumesCombo.currentText())

    def cadastre_layer_name(self):
        return str(self.ui.cadastreCombo.currentText())

    def cadastre_terrain_layer_name(self):
        return str(self.ui.cadastreTerrainCombo.currentText())

```

E.17 SOURCE CODE FOR DIALOGS/SPATIALJOINMAXAREADIALOG.PY FILE

```
from PyQt4 import QtCore, QtGui
from qgis.utils import iface
from Ui_SpatialJoinMaxArea import Ui_Dialog
from ..util import layer_helper
import os

class SpatialJoinMaxAreaDialog(QtGui.QDialog):

    def __init__(self):
        QtGui.QDialog.__init__(self)
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.ui.layerName.setText(os.path.join(os.getenv('HOME'), "SpatialJoinMaxArea.shp"))
        QtCore.QObject.connect(self.ui.locationButton, QtCore.SIGNAL('clicked()'), self.save_location_dialog)
        QtCore.QObject.connect(self.ui.layer2Combo, QtCore.SIGNAL('currentIndexChanged(QString)'), self.on_layer_changed)
        layers = iface.legendInterface().layers()
        for l in layers:
            self.ui.layer1Combo.addItem(l.name())
            self.ui.layer2Combo.addItem(l.name())
        #self.ui.cadastreTerrainCombo.addItem(l.name())

    def save_location_dialog(self):
        location = QtGui.QFileDialog.getSaveFileName(None, 'Shapefile file:', self.ui.layerName.text(), 'Shp (*.shp);; All files (*)')
        if len(location) > 0:
            self.ui.layerName.setText(location)

    def location(self):
        return self.ui.layerName.text()

    def layer1_layer_name(self):
        return str(self.ui.layer1Combo.currentText())

    def layer2_layer_name(self):
        return str(self.ui.layer2Combo.currentText())

    def field_name(self):
        return str(self.ui.fieldCombo.currentText())

    def on_layer_changed(self):
        self.ui.fieldCombo.clear();
        layer = layer_helper.get_layer(self.layer2_layer_name())
        for attr in layer.pendingFields():
            self.ui.fieldCombo.addItem(attr.name())
```

E.18 SOURCE CODE FOR LOGIC/CODE_GENERATOR.PY FILE

```
from qgis.core import *

"""

ctess ratio             construction age
 <0.3    1       <=1945  A
0.3-0.4    2       1946-1960   B
0.4-0.5   3       1961-1980   C
0.5-0.6  4       1981-1990   D
0.6-0.7     5       1991-2005   E
```

```
11:0.7-0.8    6       >=2006  F
12:0.8     7
13:
14:For residential buildings
15:<0.3    0.3-0.4     0.4-0.5     0.5-0.6     0.6-0.7     0.7-0.8     >=0.8
16:<=1945     E1-A.1  E1-A.2  E1-A.3  E1-A.4  E1-A.5  E1-A.6  E1-A.7
17:1946-1960     E1-B.1  E1-B.2  E1-B.3  E1-B.4  E1-B.5  E1-B.6  E1-B.7
18:1961-1980    E1-C.1  E1-C.2  E1-C.3  E1-C.4  E1-C.5  E1-C.6  E1-C.7
19:1981-1990   E1-D.1  E1-D.2  E1-D.3  E1-D.4  E1-D.5  E1-D.6  E1-D.7
20:1991-2005  E1-E.1  E1-E.2  E1-E.3  E1-E.4  E1-E.5  E1-E.6  E1-E.7
21:>=2006    E1-F.1  E1-F.2  E1-F.3  E1-F.4  E1-F.5  E1-F.6  E1-F.7
22:
23:For non residential buildings
24:<0.3    0.3-0.4     0.4-0.5     0.5-0.6     0.6-0.7     0.7-0.8     >=0.8
25:<=1945     En1-A.1     En1-A.2     En1-A.3     En1-A.4     En1-A.5     En1-A.6     En1-A.7
26:1946-1960     En1-B.1     En1-B.2     En1-B.3     En1-B.4     En1-B.5     En1-B.6     En1-B.7
27:1961-1980    En1-C.1     En1-C.2     En1-C.3     En1-C.4     En1-C.5     En1-C.6     En1-C.7
28:1981-1990   En1-D.1     En1-D.2     En1-D.3     En1-D.4     En1-D.5     En1-D.6     En1-D.7
29:1991-2005  En1-E.1     En1-E.2     En1-E.3     En1-E.4     En1-E.5     En1-E.6     En1-E.7
30:>=2006    En1-F.1     En1-F.2     En1-F.3     En1-F.4     En1-F.5     En1-F.6     En1-F.7""
31:"""
32:
33:def get_code(prefix, epoch_str, sv):
34:    construction_age = ''
35:    # convert epoch str in first value
36:    epoch = 0
37:    if epoch_str == None or epoch_str == '':
38:        epoch = 0
39:    elif epoch_str.startswith('<=') or epoch_str.startswith('>='):
40:        epoch = int(epoch_str[2:6])
41:    else:
42:        epoch = int(epoch_str[:4])
43:
44:    if epoch == 0: construction_age = 'X'
45:    elif epoch <= 1945: construction_age = 'A'
46:    elif epoch in range(1946, 1960): construction_age = 'B'
47:    elif epoch in range(1961, 1980): construction_age = 'C'
48:    elif epoch in range(1981, 1990): construction_age = 'D'
49:    elif epoch in range(1991, 2005): construction_age = 'E'
50:    else: construction_age = 'F'
51:
52:    compactness = ''
53:    if sv <= 0.3: compactness = '1'
54:    elif sv > 0.3 and sv <= 0.4: compactness = '2'
55:    elif sv > 0.4 and sv <= 0.5: compactness = '3'
56:    elif sv > 0.5 and sv <= 0.6: compactness = '4'
57:    elif sv > 0.6 and sv <= 0.7: compactness = '5'
58:    elif sv > 0.7 and sv <= 0.8: compactness = '6'
59:    else: compactness = '7'
60:    return prefix + '-' + construction_age + '-' + compactness
61:
62:
63:def get_code_for_residential_building(epoch, sv):
64:    return get_code('E1', epoch, sv)
65:
66:
67:def get_code_for_non_residential_building(epoch, sv):
68:    return get_code('En1', epoch, sv)
69:
```

E.19 SOURCE CODE FOR LOGIC/MEM.PY FILE

```
1:from qgis.utils import iface
2:from qgis.core import QgsVectorLayer, QgsField, QgsMapLayerRegistry, QgsGeometry
3:from PyQt4.QtCore import *
4:from PyQt4.QtGui import *
5:from ..DEFINES import *
6:from ..util import layer_helper
7:import pdb
8:
9:def merge(feature, features_input):
10:    """
11:    Merge geometries from features list that touch given feature.
```

```
    """
    count = 0
    features_output = [ feature ]
    geom = QgsGeometry(feature.geometry())
    #geom = QgsGeometry.fromPolygon(feature.geometry())
    while True:
        breakloop = True
        for f in features_input:
            if f in features_output:
                continue
            g = f.geometry()
            #if geom.equals(g):
            #    continue
            if geom.contains(g):
                continue
            if geom.within(g):
                geom = QgsGeometry(g)
                features_output.append(f)
                breakloop = False
                break
            if not geom.disjoint(g) or geom.touches(g) or geom.overlaps(g):
                geom = QgsGeometry(geom.combine(g))
                features_output.append(f)
                breakloop = False
                break
        if breakloop:
            break
    return (geom, features_output)

def get_intersection(g1, g2):
    """
    This function returns a set of QgsGeometry. Each geometry is the intersection of external ring of given geometry.
    """

    #if len(p1) == 0 or len(p2) == 0:
    #    pyqtRemoveInputHook()
    #    pdb.set_trace()
    result = []
    p1 = g1.asPolygon()
    p2 = g2.asPolygon()
    if len(p1) > 0 and len(p2) > 0:
        r1 = QgsGeometry.fromPolyline(p1[0])
        r2 = QgsGeometry.fromPolyline(p2[0])
        if not r1.equals(r2) and r1.intersects(r2):
            i = r1.intersection(r2)
            result.append(QgsGeometry(i))
    return result

def compute_external_wall_surface(index, geometry, features, height):
    """
    Returns the external wall surface.
    """
    ids = index.intersects(geometry.boundingBox())
    common_part = 0
    for i in ids:
        g = features[i].geometry()
        if not g.equals(geometry) and not geometry.disjoint(g):
            intersection_set = get_intersection(g, geometry)
            h = features[i][FIELD_VOLUME_HEIGHT]
            h2 = height
            if h2 <= h:
                h = h2
            for intersection in intersection_set:
                common_part += intersection.length() * h
    return (geometry.length() * height) - common_part
```

E.20 SOURCE CODE FOR UTIL/LAYER_HELPER.PY FILE

```
from qgis.utils import iface
from qgis.core import QgsVectorLayer, QgsField, QgsMapLayerRegistry, QgsVectorFileWriter, QgsSpatialIndex, QgsGeometry, QgsMessageLog
from PyQt4.QtCore import *
```

```
  4:from PyQt4.QtGui import *
  5:
  6:import os
  7:
  8:def create_layer(name, attributes, baselayer = None, layertype = 'Polygon', addregistry = True
):
  9:    layerattr = ''
 10:    if baselayer != None:
 11:        layerattr = '?crs=' + baselayer.crs().authid()
 12:        layerattr += '&index=yes'
 13:    layer = QgsVectorLayer(layertype + layerattr, name, "memory") # memory
 14:    pr = layer.dataProvider()
 15:    layer.startEditing()
 16:    pr.addAttributes(attributes)
 17:    layer.updateFields()
 18:    layer.commitChanges()
 19:    if addregistry:
 20:        QgsMapLayerRegistry.instance().addMapLayer(layer)
 21:    return layer
 22:
 23:def save_layer(layer, location):
 24:    error = QgsVectorFileWriter.writeAsVectorFormat(layer, location, "CP1250", layer.dataProvi
der().crs(), "ESRI Shapefile")
 25:    #error = QgsVectorFileWriter.writeAsShapefile(layer, location, "CP1250")
 26:    return error
 27:
 28:def get_layer(name):
 29:    layers = iface.legendInterface().layers()
 30:    for layer in layers:
 31:        if layer.name() == name:
 32:            return layer
 33:    return None
 34:
 35:def load_features(layer):
 36:    return {f.id(): f for f in layer.getFeatures()}
 37:
 38:def load_features_with_id(id, layer):
 39:    return {f[id]: f for f in layer.getFeatures()}
 40:
 41:def build_spatialindex(features):
 42:    index = QgsSpatialIndex()
 43:    for f in features:
 44:        index.insertFeature(f)
 45:    return index
 46:
 47:def copy_geometry(feature):
 48:    return QgsGeometry(feature.geometry())
 49:    #polygon = feature.geometry().asQPolygonF()
 50:    #return QgsGeometry.fromQPolygonF(polygon)
 51:
 52:# Move in UI helper
 53:
 54:def show_features(layer, features):
 55:    selection = []
 56:    for x in features:
 57:        selection.append(x.id())
 58:    layer.setSelectedFeatures(selection)
 59:    iface.actionZoomToSelected().trigger()
 60:
 61:
 62:def get_intersection_max_area(index, f, features):
 63:    g = f.geometry()
 64:    ids = index.intersects(g.boundingBox())
 65:    #QgsMessageLog.logMessage("Intersection returns " + str(len(ids)) + " IDs.")
 66:    if len(ids) == 1:
 67:        if not g.disjoint(features[ids[0]].geometry()):
 68:            return ids[0]
 69:    elif len(ids) > 1:
 70:        id_max = -1
 71:        area_max = -1
 72:        for i in ids:
 73:            if not g.disjoint(features[i].geometry()):
 74:                common = QgsGeometry(g.intersection(features[i].geometry()))
```

```
75:                 if common.area() > area_max:
76:                     id_max = i
77:                     area_max = common.area()
78:         if id_max > -1:
79:             return id_max
80:     return -1
```

E.21 SOURCE CODE FOR UTIL/PYQT_HELPER.PY FILE

```
1:from qgis.core import *
2:from PyQt4 import QtCore, QtGui
3:import traceback
4:import time
5:
6:def create_progress_bar(iface, message, worker):
7:     """
8:     This function retrun a dialog with a progress par.
9:     """
10:     messageBar = iface.messageBar().createMessage(message)
11:     progressBar = QtGui.QProgressBar()
12:     progressBar.setAlignment(QtCore.Qt.AlignLeft|QtCore.Qt.AlignVCenter)
13:     #cancelButton = QtGui.QPushButton()
14:     #cancelButton.setText('Cancel')
15:     #cancelButton.clicked.connect(worker.kill)
16:     messageBar.layout().addWidget(progressBar)
17:     #messageBar.layout().addWidget(cancelButton)
18:     iface.messageBar().pushWidget(messageBar, iface.messageBar().INFO)
19:     return (messageBar, progressBar)
```

E.22 SOURCE CODE FOR UTIL/READER_CSV.PY FILE

```
1:import csv
2:import sys
3:
4:class BaseReader:
5:     """
6:     Base reader for CSV.
7:     """
8:
9:     filename = None
10:     valid = False
11:     header = []
12:     table = {}
13:
14:     def load(self, filename):
15:         try:
16:             first = True
17:             with open(filename, 'rU') as csvfile:
18:                 result = csv.reader(csvfile, csv.excel, delimiter=',', quotechar='\\')
19:                 for row in result:
20:                     if first:
21:                         self.handle_header(row)
22:                         first = False
23:                     else:
24:                         self.handle_row(row)
25:                 self.filename = filename
26:                 self.valid = True
27:         except AttributeError as e:
28:             print('file %s, %s' % (filename, e))
29:         except csv.Error as e:
30:             print('file %s, %s' % (filename, e))
31:         except:
32:             print('Exception: ' + str(sys.exc_info()[0]))
33:             self.valid = False
34:
35:     def get(self, code):
36:         if not self.valid:
37:             return None
38:         try:
39:             return self.table[code]
40:         except:
41:             return None
42:
```

```
43:     def get_element(self, code, header, default = None):
44:         if not self.valid:
45:             return default
46:         try:
47:             idx = -1
48:             cur = 0
49:             lowheader = header.lower()
50:             for h in self.header:
51:                 if lowheader == h.lower():
52:                     idx = cur
53:                     break
54:                 cur += 1
55:             return self.table[code][idx]
56:         except:
57:             return default
58:
59:     def handle_header(self, row):
60:         return False
61:
62:     def handle_row(self, row):
63:         return False
64:
65:
66:class ISTAT(BaseReader):
67:     def __init__(self, filename):
68:         #super(ISTAT, self).__init__()
69:         self.load(filename)
70:
71:     def handle_header(self, row):
72:         self.header = row
73:         return True
74:
75:     def handle_row(self, row):
76:         if len(row) >= 2:
77:             self.table[str(row[0])] = row
78:             return True
79:         else:
80:             return False
81:
82:def codcat_to_epcs(type_usage, cadastre):
83:     tokens = cadastre.split('|')
84:     if len(tokens) >= 3:
85:         ret = type_usage + '-' + '-'.join(tokens[2:])
86:         return ret
87:     else:
88:         return ''
89:
90:class EPCs(BaseReader):
91:     def __init__(self, filename):
92:         #super(EPCs, self).__init__()
93:         self.load(filename)
94:
95:     def handle_header(self, row):
96:         self.header = row
97:         return True
98:
99:     def handle_row(self, row):
100:         if len(row) >= 2:
101:             self.table[str(row[0])] = row
102:             return True
103:         else:
104:             return False
```