# -\*- coding: utf-8 -\*-  
*"""  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 SaveAttributes  
 A QGIS plugin  
 this plugin saves the attribute of selected vector layer  
 Generated by Plugin Builder: http://g-sherman.github.io/Qgis-Plugin-Builder/  
 -------------------  
 begin : 2024-04-02  
 git sha : $Format:%H$  
 copyright : (C) 2024 by Dounia  
 email : gj@spatial.com  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* \*  
 \* This program is free software; you can redistribute it and/or modify \*  
 \* it under the terms of the GNU General Public License as published by \*  
 \* the Free Software Foundation; either version 2 of the License, or \*  
 \* (at your option) any later version. \*  
 \* \*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
"""*import sys  
from qgis.PyQt.QtCore import QSettings, QTranslator, QCoreApplication  
from qgis.PyQt.QtGui import QIcon  
from qgis.PyQt.QtWidgets import QAction, QPushButton  
from qgis.core import QgsProject  
  
# Initialize Qt resources from file resources.py  
from .resources import \*  
# Import the code for the dialog  
from .save\_attributes\_dialog import SaveAttributesDialog  
import os.path  
  
  
class SaveAttributes:  
 *"""QGIS Plugin Implementation."""* def \_\_init\_\_(self, iface):  
 *"""Constructor.  
  
 :param iface: An interface instance that will be passed to this class  
 which provides the hook by which you can manipulate the QGIS  
 application at run time.  
 :type iface: QgsInterface  
 """* # Save reference to the QGIS interface  
 self.iface = iface  
 # initialize plugin directory  
 self.plugin\_dir = os.path.dirname(\_\_file\_\_)  
 # initialize locale  
 locale = QSettings().value('locale/userLocale')[0:2]  
 locale\_path = os.path.join(  
 self.plugin\_dir,  
 'i18n',  
 'SaveAttributes\_{}.qm'.format(locale))  
  
 if os.path.exists(locale\_path):  
 self.translator = QTranslator()  
 self.translator.load(locale\_path)  
 QCoreApplication.installTranslator(self.translator)  
  
 # Declare instance attributes  
 self.actions = []  
 self.menu = self.tr(u'&SaveAttributes')  
  
 # Check if plugin was started the first time in current QGIS session  
 # Must be set in initGui() to survive plugin reloads  
 self.first\_start = None  
  
 # noinspection PyMethodMayBeStatic  
 def tr(self, message):  
 *"""Get the translation for a string using Qt translation API.  
  
 We implement this ourselves since we do not inherit QObject.  
  
 :param message: String for translation.  
 :type message: str, QString  
  
 :returns: Translated version of message.  
 :rtype: QString  
 """* # noinspection PyTypeChecker,PyArgumentList,PyCallByClass  
 return QCoreApplication.translate('SaveAttributes', message)  
  
 def add\_action(  
 self,  
 icon\_path,  
 text,  
 callback,  
 enabled\_flag=True,  
 add\_to\_menu=True,  
 add\_to\_toolbar=True,  
 status\_tip=None,  
 whats\_this=None,  
 parent=None):  
 *"""Add a toolbar icon to the toolbar.  
  
 :param icon\_path: Path to the icon for this action. Can be a resource  
 path (e.g. ':/plugins/foo/bar.png') or a normal file system path.  
 :type icon\_path: str  
  
 :param text: Text that should be shown in menu items for this action.  
 :type text: str  
  
 :param callback: Function to be called when the action is triggered.  
 :type callback: function  
  
 :param enabled\_flag: A flag indicating if the action should be enabled  
 by default. Defaults to True.  
 :type enabled\_flag: bool  
  
 :param add\_to\_menu: Flag indicating whether the action should also  
 be added to the menu. Defaults to True.  
 :type add\_to\_menu: bool  
  
 :param add\_to\_toolbar: Flag indicating whether the action should also  
 be added to the toolbar. Defaults to True.  
 :type add\_to\_toolbar: bool  
  
 :param status\_tip: Optional text to show in a popup when mouse pointer  
 hovers over the action.  
 :type status\_tip: str  
  
 :param parent: Parent widget for the new action. Defaults None.  
 :type parent: QWidget  
  
 :param whats\_this: Optional text to show in the status bar when the  
 mouse pointer hovers over the action.  
  
 :returns: The action that was created. Note that the action is also  
 added to self.actions list.  
 :rtype: QAction  
 """* icon = QIcon(icon\_path)  
 action = QAction(icon, text, parent)  
 action.triggered.connect(callback)  
 action.setEnabled(enabled\_flag)  
  
 if status\_tip is not None:  
 action.setStatusTip(status\_tip)  
  
 if whats\_this is not None:  
 action.setWhatsThis(whats\_this)  
  
 if add\_to\_toolbar:  
 # Adds plugin icon to Plugins toolbar  
 self.iface.addToolBarIcon(action)  
  
 if add\_to\_menu:  
 self.iface.addPluginToVectorMenu(  
 self.menu,  
 action)  
  
 self.actions.append(action)  
  
 return action  
  
 def initGui(self):  
 *"""Create the menu entries and toolbar icons inside the QGIS GUI."""* icon\_path = ':/plugins/save\_attributes/icon.png'  
 self.add\_action(  
 icon\_path,  
 text=self.tr(u''),  
 callback=self.run,  
 parent=self.iface.mainWindow())  
  
 # will be set False in run()  
 self.first\_start = True  
  
 def unload(self):  
 *"""Removes the plugin menu item and icon from QGIS GUI."""* for action in self.actions:  
 self.iface.removePluginVectorMenu(  
 self.tr(u'&SaveAttributes'),  
 action)  
 self.iface.removeToolBarIcon(action)  
  
 def tester(self):  
 print("Hello")  
  
 def dounia(self):  
 print("Bonjour")  
  
 def onLayerChanged(self, index):  
 *"""Gère le changement de couche et met à jour le comboBox des champs"""* combo\_box\_fields = self.dlg.comboBox\_2  
 combo\_box\_fields.clear() # Réinitialiser le combo box des champs  
  
 layer\_name = self.dlg.comboBox.currentText() # Obtenir le nom de la couche sélectionnée  
 layer = QgsProject.instance().mapLayersByName(layer\_name)[0]  
  
 # Ajouter les noms des champs de la couche au combo box  
 if layer is not None:  
 for field in layer.fields():  
 combo\_box\_fields.addItem(field.name())  
  
 def add\_text\_to\_edit(self):  
 selected\_text = self.dlg.comboBox\_2.currentText()  
 self.dlg.textEdit.append(selected\_text)  
  
 def run(self):  
 if self.first\_start:  
 self.first\_start = False  
 self.dlg = SaveAttributesDialog()  
  
 # Configuration de l'interface graphique  
 self.dlg.pushButton.setText("Ajouter")  
 self.dlg.pushButton.clicked.connect(self.add\_text\_to\_edit)  
  
 # Obtenir une référence au combo box de la couche  
 combo\_box = self.dlg.comboBox  
 combo\_box.clear()  
  
 layers = QgsProject.instance().mapLayers().values()  
 for layer in layers:  
 combo\_box.addItem(layer.name())  
  
 # Lier le gestionnaire d'événements à comboBox  
 combo\_box.currentIndexChanged.connect(self.onLayerChanged)  
  
 # Obtenir une référence au combo box des champs  
 combo\_box\_fields = self.dlg.comboBox\_2  
 combo\_box\_fields.clear() # Nettoyer le combo box des champs  
  
 self.dlg.show()  
  
 result = self.dlg.exec\_()  
 if result:  
 pass