**20200816**

import os

import fnmatch

import sys

import sqlite3

import xml.etree.cElementTree as ET

from PyQt5 import QtSql

from PyQt5.QtCore import pyqtSlot

from PyQt5.QtGui import QFont

from PyQt5.QtWidgets import (QWidget, QLabel, QApplication, QPushButton, QMainWindow)

BASE\_DIR = os.getcwd()

class MyParser(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

# Инициализация программы

self.initUI()

# def createDB(self):

# """

# Соединение с базой данных SQLite

# """

# db = QtSql.QSqlDatabase.addDatabase('QSQLITE')

# db.setDatabaseName('comps.db')

# # Проверка подключения к базе

# if db.open():

# # Работаем

# pass

# else:

# # Ошибка

# print(db.lastError().text())

# # db.open()

# # db.close()

def initUI(self):

# Меню

menu = self.menuBar()

# Настройки шрифта текста.

font = QFont()

font.setFamily("Arial")

font.setPointSize(18)

# Создадим надпись

file\_lbl = QLabel(BASE\_DIR, self)

file\_lbl.setFont(font)

file\_lbl.move(50, 50)

# Создаем кнопку "OK".

ok\_btn = QPushButton('ОК', self)

ok\_btn.resize(100, 25)

ok\_btn.move(650, 300)

# Создаем кнопку "Выход".

quit\_btn = QPushButton('Выход', self)

quit\_btn.resize(100, 25)

quit\_btn.move(650, 350)

# Каждый клик кнопки вызывает метод "makeRequest"

ok\_btn.clicked.connect(self.makeRequest)

# Выход

quit\_btn.clicked.connect(QApplication.instance().quit)

self.setFixedSize(800, 400)

self.setWindowTitle('Парсер XML')

self.show()

@pyqtSlot()

def makeRequest(self):

"""

После нажатия на "ОК" выполняется запрос к API с выбранными данными.

"""

dir\_xml = os.path.join(BASE\_DIR, 'xml')

files = fnmatch.filter(os.listdir(dir\_xml), "\*.XML") # только XML

# file\_xml = os.path.join(dir\_xml, 'ECONOM.XML')

# parser\_xml(file\_xml)

for num\_files, file in enumerate(files):

file\_xml = os.path.join(dir\_xml, file)

parser\_xml(file\_xml)

print(num\_files)

def parser\_xml(file\_xml):

"""

Прсер xml-файла (программа HWINFO)

"""

# Подключение к базе

db = QtSql.QSqlDatabase.addDatabase('QSQLITE')

db.setDatabaseName('comps.db')

# Проверка подключения к базе

if db.open():

# Работаем

pass

else:

# Ошибка

print(db.lastError().text())

# comp\_dict = {} # словарь узлов

comp\_info = () # кортеж словатей

# tree = ET.parse(os.path.join(os.getcwd(), 'SERVERIT.XML'))

tree = ET.parse(file\_xml)

root = tree.getroot() # корневой элемент

# print(root) # HWINFO

# print(len(root)) # 1

for child in root:

print(child.tag + ' :') # COMPUTER

# node\_dict = {}

# print(len(child)) # 11

# параметры компьютера

for cn, sub\_child in enumerate(child):

if (sub\_child.tag != 'SubNodes') and (sub\_child.tag == 'Property'):

print(f'{sub\_child[0].text} => {sub\_child[1].text}')

# узлы компьютера

# if cn == 10:

if sub\_child.tag == 'SubNodes':

# список параметров

# comp\_dict[child.tag] = node\_dict

# comp\_info += (comp\_dict,)

# comp\_dict = {}

# node\_dict = {}

print(' {0} - {1} :'.format(sub\_child.tag, len('sub\_child') + 1)) # количество узлов

for node in sub\_child:

# Процессор

if node.tag == 'CPU':

for cp, par in enumerate(node):

# параметры процессора

if cp == 0:

print(f' {par.text} :')

key\_dict = par.text # запомним имя основного ключа

if (cp > 1) and (cp < 5):

print(f' {par[0].text} => {par[1].text}')

# node\_dict[par[0].text] = par[1].text

# список параметров

# comp\_dict[key\_dict] = node\_dict

# comp\_info += (comp\_dict,)

# comp\_dict = {}

# node\_dict = {}

# Материнская плата

if node.tag == 'MOBO':

# print(f' {node.tag} :')

for cp, par in enumerate(node):

# параметры материнской платы

if cp == 0:

print(f' {par.text} :')

key\_dict = par.text # запомним имя основного ключа

if (cp > 1) and (cp < 5):

print(f' {par[0].text} => {par[1].text}')

# список параметров

# comp\_dict[child.tag] = node\_dict

# Память

if node.tag == 'MEMORY':

# print(f' {node.tag} :')

num\_par = [3, 5, 6, 8] # список номеров параметров

for cp, par in enumerate(node):

if cp == 0:

print(f' {par.text} :')

# параметры памяти

if cp in num\_par:

print(f' {par[0].text} => {par[1].text}')

# модули

if par.tag == 'SubNode':

print(' Memory module :')

print(f' {par[0].text}')

# список параметров

# comp\_dict[child.tag] = node\_dict

# Видео

if node.tag == 'VIDEO':

# print(f' {node.tag} :')

for cp, par in enumerate(node):

if cp == 0:

print(f' {par.text} :')

if par.tag == 'SubNode':

print(f' {par[0].text}')

# список параметров

# comp\_dict[child.tag] = node\_dict

# Монитор

if node.tag == 'MONITOR':

# print(f' {node.tag} :')

for cp, par in enumerate(node):

if cp == 0:

print(f' {par.text} :') # Monitor :

if par.tag == 'SubNode':

num\_par = [1, 2, 3, 4] # список номеров параметров

for np, p in enumerate(par):

# параметры монитора

if np in num\_par:

print(f' {p[0].text} => {p[1].text}')

# список параметров

# comp\_dict[child.tag] = node\_dict

# Диски

if node.tag == 'DRIVES':

# print(f' {node.tag} :')

for cp, par in enumerate(node):

if cp == 0:

print(f' {par.text} :') # Drives :

if par.tag == 'SubNode':

for np, p in enumerate(par):

if p.tag == 'SubNode':

print(f' {p[0].text}')

# print(len(p))

for sub\_np, sub\_p in enumerate(p):

num\_par = [1, 3, 4, 5, 7, 8, 13] # список номеров параметров

if sub\_np in num\_par:

print(f' {sub\_p[0].text} => {sub\_p[1].text}')

# список параметров

# comp\_dict[child.tag] = node\_dict

# Звук

if node.tag == 'SOUND':

# print(f' {node.tag} :')

for cp, par in enumerate(node):

if cp == 0:

print(f' {par.text} :')

if par.tag == 'SubNode':

print(f' {par[0].text}')

# список параметров

# comp\_dict[child.tag] = node\_dict

# Сеть

if node.tag == 'NETWORK':

for cp, par in enumerate(node):

if cp == 0:

print(f' {par.text} :') # Network :

if par.tag == 'SubNode':

num\_par = [1, 3, 4] # список номеров параметров

for np, p in enumerate(par):

# параметры сетевой карты

if np in num\_par:

if p[0].text == 'MAC Address':

print(f' {p[0].text} => {p[1].text.replace("-", ":")}')

else:

print(f' {p[0].text} => {p[1].text}')

# список параметров

# comp\_dict[child.tag] = node\_dict

return comp\_info

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

window = MyParser()

sys.exit(app.exec\_())