|  |
| --- |
|  |
| Сопроводительный документ |
| https://github.com/BadCatsss/Excel\_to\_Json/tree/Qt |

|  |
| --- |
| 1.1.2019 |

Оглавление

[Введение 2](#_Toc16362435)

[Приложение «Excel\_to\_Json» 2](#_Toc16362436)

[Описание назначения приложения 2](#_Toc16362437)

[Особенности использования 4](#_Toc16362438)

[Техническая часть 10](#_Toc16362439)

[Структура проекта 10](#_Toc16362440)

[Файл main.cpp 10](#_Toc16362441)

[Файл Converter.cpp 10](#_Toc16362442)

[Файл Convertor.h 10](#_Toc16362443)

[Файл main.cpp 11](#_Toc16362444)

[f\_path 12](#_Toc16362445)

[string f\_std\_str\_path 12](#_Toc16362446)

[Convertor cnv(f\_path,PrintDataFlag,PrintNameFlag); 12](#_Toc16362447)

[***bool PrintDataFlag*** 12](#_Toc16362448)

[***bool PrintNameFlag*** 12](#_Toc16362449)

[cnv.OpenBook(); 12](#_Toc16362450)

[***cnv.GetActivetWorkSheet();*** 13](#_Toc16362451)

[***cnv.SetMaxRows();*** 13](#_Toc16362452)

[***cnv.SetMaxColumns();*** 13](#_Toc16362453)

[***cnv.Generate();*** 13](#_Toc16362454)

[Файл Converter.cpp 14](#_Toc16362455)

[***Convertor::Convertor(QString f\_str\_path, bool f1, bool f2)*** 19](#_Toc16362456)

[***QString Convertor:: ParsePath(QString path)*** 20](#_Toc16362457)

[***void Convertor::OpenBook()*** 20](#_Toc16362458)

[***void Convertor::SetMaxRows( )*** 20](#_Toc16362459)

[***void Convertor::SetMaxColumns( )*** 20](#_Toc16362460)

[***int Convertor:: GetSheetCount() const*** 21](#_Toc16362461)

[***void Convertor::SetActivetWorkSheet(QString p)*** 21](#_Toc16362462)

[***QXlsx::AbstractSheet\* Convertor:: GetActivetWorkSheet() const*** 21](#_Toc16362463)

[***QXlsx::Document\* Convertor:: GetBook() const*** 21](#_Toc16362464)

[***QStringList Convertor::OpenWorkSheet()*** 22](#_Toc16362465)

[***void Convertor::Generate()*** 22](#_Toc16362466)

[***void Convertor::GetDigitalStandartName()*** 22](#_Toc16362467)

[***void Convertor::GetData()*** 23](#_Toc16362468)

[***void Convertor::PrintNames()*** 25](#_Toc16362469)

[***void Convertor::PrintData()*** 25](#_Toc16362470)

[***void Convertor::PrintBlocksToFile()*** 26](#_Toc16362471)

[Файл Converter.h 28](#_Toc16362472)

[**size\_t maxRows;** 29](#_Toc16362473)

[**size\_t maxCols;** 29](#_Toc16362474)

[**bool PrintDataFlag** 29](#_Toc16362475)

# Введение

## Приложение «Excel\_to\_Json»

### Описание назначения приложения

Приложение «Excel\_to\_Json» - предназначено для простой конвертации (прим. – подразумевается копирование данных из файла одного формата – в создаваемый файл - другого) файлов с расширением и структурой **.xlsx** – в файлы формата **.json** . Приложение производит операцию парсинга ячеек тыблицы в формте **.xlsx** – создавая «блоки одноименных данных» (рис. 1), в которые вкладываются блоки меньшего размера - по 4 значения и записывая полученный результат в соответствуящем виде(рис. 2) в создаваемый файл с расширением .json .

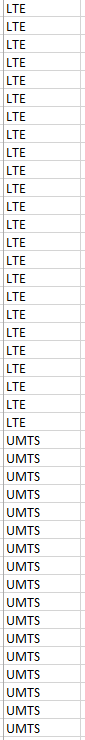
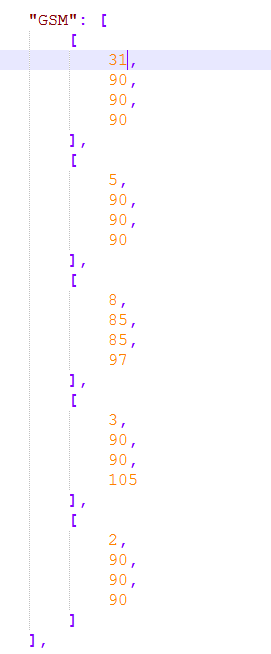
 

Рисунок 2. Обработанные данные в формате .json

Рисунок . Данные таблицы из файла формата .xlsx

### Особенности использования

Приложение не имеет графического интерфейса, а предоставляет к использованию – интерфейс командной строки, так же иногда называемый - «консоль», в среде операционной системы

(рис. 3).

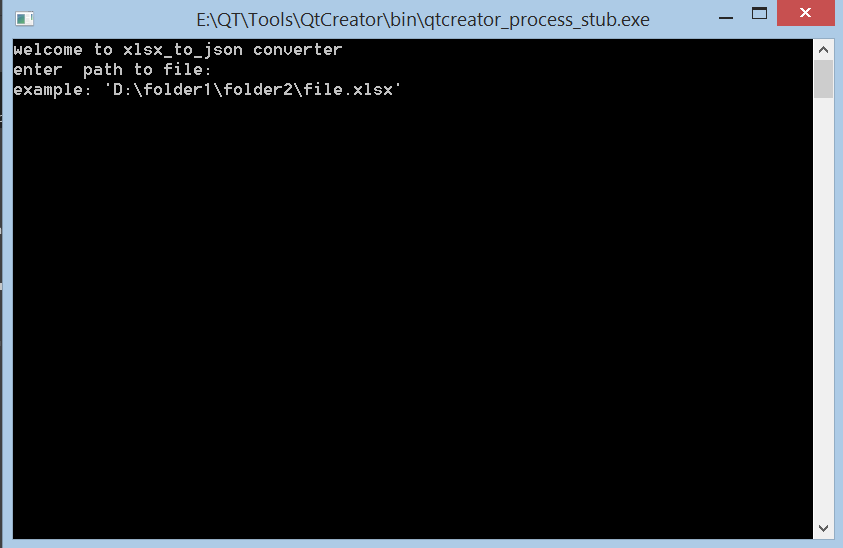


Рисунок 3. Пример работы приложения

Приложение получает на ввод от пользователя – абсолютный путь к файлу, который включает в себя сам путь и имя файла.

Пример абсолютного пути, вводимого пользователем:

**D:\User document\folder\example.xlsx**

Пример октрытия файла.  
Допустим, что файл находится по следующему пути:   
**D:\My\_Documents**

И имеет имя:

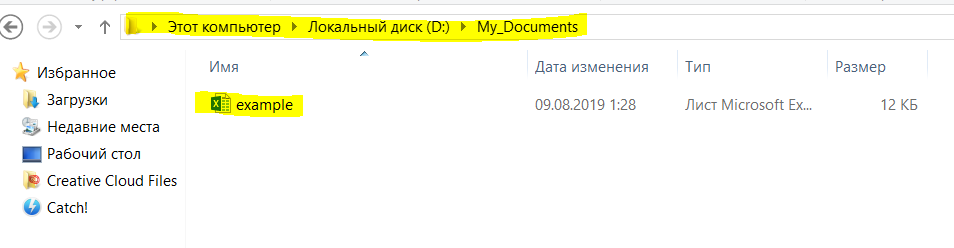
**example**

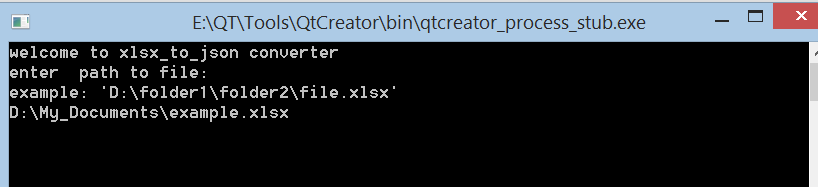
Следовательно, абсолютный путь к файлу будет следующим:

**D:\My\_Documents\** **example.xlsx**

**(Обратите внимание, что в данном примере – имя каталога, в котором находится исходный**  **файл – содержит символ « \_ » - т.к , в данном примере, этот символ указан вместо пробела в названии директории, в противном случае – указывается обычный символ «пробел»)**

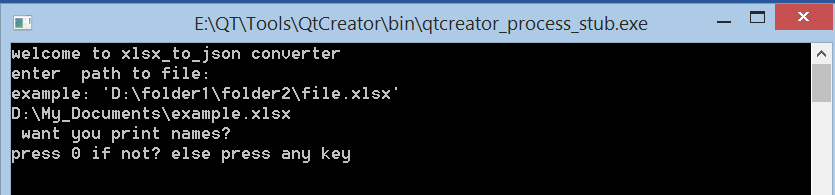
**(Так же, обязательно добавлять к имени файла .xlsx при указании имени файла)**





Введите путь к фалу и нажмите клавишу “enter”.

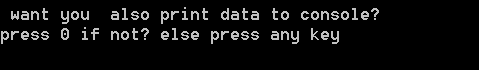
Вы увидите следующее диалоговое сообщение:



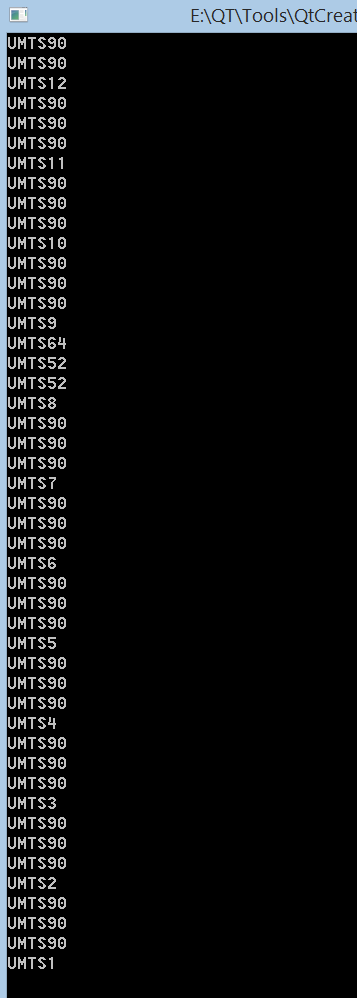
При утвердительном варианте ответа – в консоль будут выведены уникальные имена для строк с данными с .xlsx файла



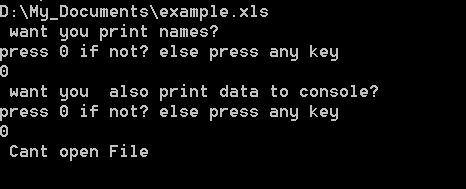
Второе диалоговое сообшение:



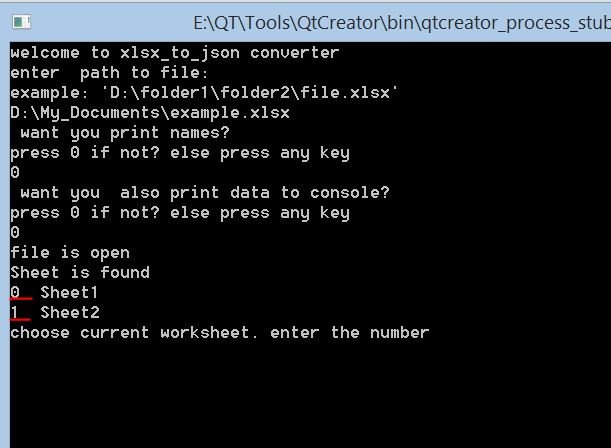
При утвердительном варианте ответа – в консоль будут выведены имена всех строк, в месте с данными с .xlsx файла



При некорректно указаном имени или пути файла – будет выведенно сообщение - **“Cant open file”**

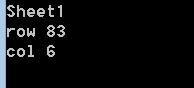


При корректном открытии файла – будет выведен список содержащихся страниц и будет предложено выбрать необходимую страницу в ,xlsx документе, с которой будет произведено чтение данных:



- введите номер необходимой страницы.

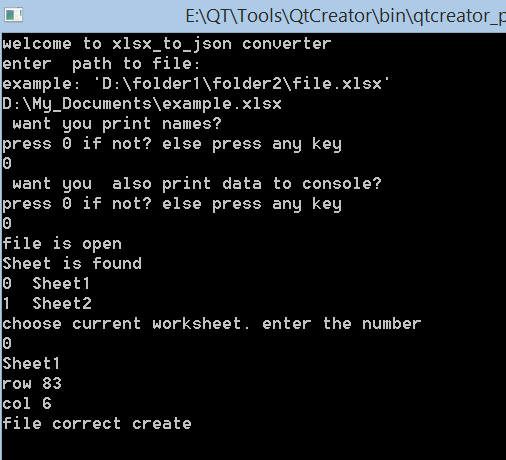
При открытии страницы – будет выведенно имя выбранной страницы и количество строк и столбцов в файле, которые заполнены данными(не пусты)



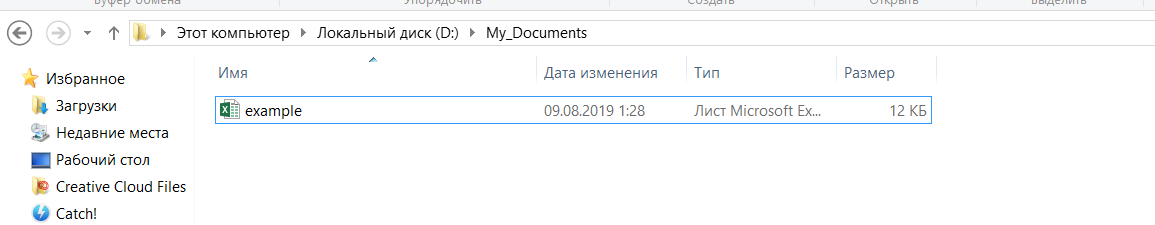
При корректном создании .json файла – будет выведенно соответствующее сообщение



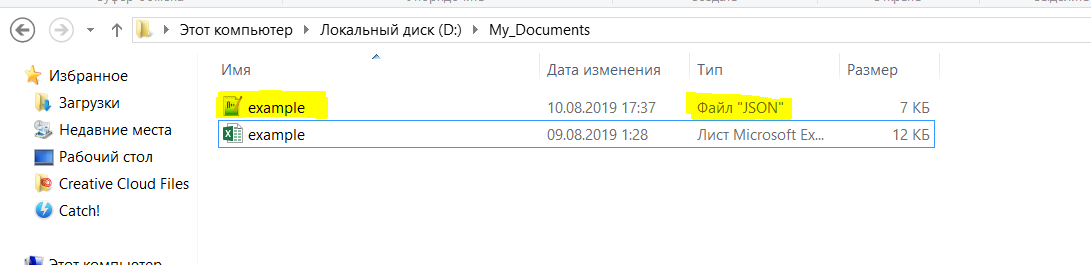
Вид всех сообщений диалога:



Каталог до работы приложения:



Каталог после работы приложения:



# Техническая часть

## Структура проекта

### Файл main.cpp

Файл main.cpp – подставляет собой стандартный файл реализации в языке программирования c++. Обычно, согласно наименованию, содержит точку входа в программу.

### Файл Converter.cpp

Файл Converter.cpp – подставляет собой стандартный файл реализации в языке программирования c++. Содержит реализацию функций членов-класса “Convertor”.

### Файл Convertor.h

Файл Converter.h – подставляет собой стандартный файл-заголовок в языке программирования c++. Содержит объявления функций членов-класса “Convertor”, реалезуемых в .cpp файлах.

## Файл main.cpp

Содержимое файла main.cpp – проекта «Excel\_to\_Json»:

#include "Convertor.h"

int **main**(int argc, char \*argv[])

{

QCoreApplication a(*argc*, argv);

QTextStream qtin(stdin);

QTextStream qtout(stdout);

QString f\_path;

string f\_std\_str\_path;

QString sh\_name;

bool PrintDataFlag=true,PrintNameFlag=true;

int ch\_1,ch\_2;

qtout<<"welcome to xlsx\_to\_json converter"<<endl;

qtout << "enter path to file:" << endl;

qtout << "example: 'D:\\folder1\\folder2\\file.xlsx'" << endl;

getline(*cin*,*f\_std\_str\_path*) ;

f\_path=Convertor::ParsePath( QString::fromStdString( f\_std\_str\_path));

qtout<<" want you print names?"<<endl;

qtout<<"press 0 if not? else press any key"<<endl;

qtin>>ch\_1;

PrintNameFlag=ch\_1;

qtout<<" want you also print data to console?"<<endl;

qtout<<"press 0 if not? else press any key"<<endl;

qtin>>ch\_2;

PrintDataFlag=ch\_2;

Convertor cnv(f\_path,PrintDataFlag,PrintNameFlag);

cnv.OpenBook();

QStringList sh\_list= cnv.OpenWorkSheet();

int ch\_number=0;

cout<<"choose current worksheet. enter the number"<<endl;

cin>>ch\_number;

if (ch\_number>=0 && ch\_number<= cnv.GetSheetCount() ) {

cnv.SetActivetWorkSheet(sh\_list[ch\_number]);

}

cnv.GetActivetWorkSheet();

cnv.SetMaxRows();

cnv.SetMaxColumns();

cnv.Generate();

cout << endl;

return a.exec();

}

### ***QString f\_path***

QString f\_path - переменная типа QString - для сохранения и задания пути к файлу в объекте класса Convertor. Передается в конструктор класса, где инициализирует внутреннюю private переменную.

### ***string f\_std\_str\_path***

string f\_std\_str\_path – используется с целью использования std::getline, который позволяет получать строку – включая пробелы. Затем, используется для инициализации f\_path.

### ***Convertor cnv(f\_path,PrintDataFlag,PrintNameFlag);***

Convertor cnv(f\_path,PrintDataFlag,PrintNameFlag); - создание объекта класса Convertor.

### ***bool PrintDataFlag***

bool PrintDataFlag – переменная, которая указывает, необходимо ли выводить данные в консоль.

### ***bool PrintNameFlag***

bool PrintNameFlag – переменная, которая указывает, необходимо ли выводить найденные уникальные имена в консоль.

### ***cnv.OpenBook();***

cnv.OpenBook(); - вызов метода, проверяющего результат открытия файла (получилось ли открыть файл)

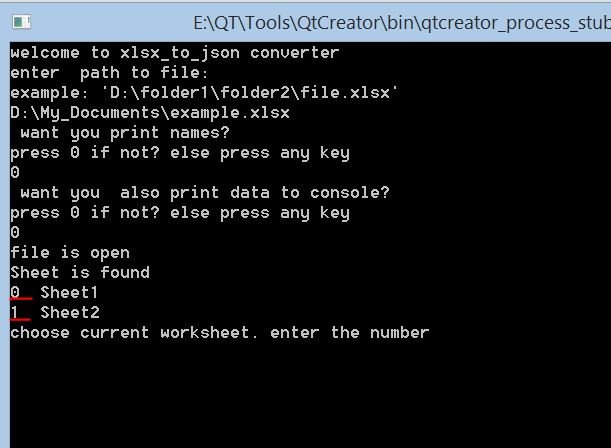
***фрагмент кода, в main.cpp:***

if (ch\_number>=0 && ch\_number<= cnv.GetSheetCount() ) {

cnv.SetActivetWorkSheet(sh\_list[ch\_number]);

}

- вывод найденых листов в файле.



### ***cnv.GetActivetWorkSheet();***

cnv.GetActivetWorkSheet(); - установка активной страницы для работы, выбранной пользователем.

### ***cnv.SetMaxRows();***

cnv.SetMaxRows(); - получение количества заполненных (не пустых) строк на странице.

### ***cnv.SetMaxColumns();***

cnv.SetMaxColumns();- получение количества заполненных (не пустых) столбцов на странице

### ***cnv.Generate();***

cnv.Generate(); - общий метод, вызывающий все private методы для парсинга.

## Файл Converter.cpp

Содержимое файла Converter.cpp – проекта «Excel\_to\_Json»:

#include <QCoreApplication>

#include <QCoreApplication>

#include "xlsxdocument.h"

#include "xlsxchartsheet.h"

#include "xlsxcellrange.h"

#include "xlsxchart.h"

#include "xlsxrichstring.h"

#include "xlsxworkbook.h"

#include <QDebug>

#include "Convertor.h"

using namespace QXlsx;

using namespace std;

//initialize

Convertor::**Convertor**(QString f\_str\_path, bool f1, bool f2) :PrintDataFlag(f1), PrintNameFlag(f2),sheet\_count(0),f\_str\_path(f\_str\_path)

{

xlsxR= new QXlsx::Document(f\_str\_path);

string s=f\_str\_path.toStdString();

s.erase(s.begin()+s.find("."),s.end());

this->f\_str\_name=QString::fromStdString( s );

}

QString Convertor:: **ParsePath**(QString path)

{

// QTextStream s(stdout);

// s<< QFileInfo (path).absolutePath()<<endl;

return QFileInfo (path).absoluteFilePath();

}

//book and sheet setup

void Convertor::**OpenBook**()

{

if (this->xlsxR->load())

{

cout << "file is open" << endl;

}

else {

cout<<" Cant open File"<<endl;

exit(-1);

}

}

void Convertor::**SetMaxRows**( )

{

int row = 1;

Cell\* cell = this->xlsxR->cellAt(row, 1); // get cell pointer.

while (cell!=nullptr) {

row++;

cell = this->xlsxR->cellAt(row, 1);

}

this->maxRows=row;

cout<<"row "<<this->maxRows<<endl;

}

void Convertor::**SetMaxColumns**( )

{

int col = 1;

Cell\* cell = this->xlsxR->cellAt(1, col); // get cell pointer.

while (cell!=nullptr) {

col++;

cell = this->xlsxR->cellAt(1, col);

}

this->maxCols=col;

cout<<"col "<<this->maxCols<<endl;

}

int Convertor:: **GetSheetCount**() const

{

return this->sheet\_count;

}

void Convertor::**SetActivetWorkSheet**(QString p)

{

this->ActiveSheet= this->xlsxR->sheet(p);

this->xlsxR->selectSheet(this->ActiveSheet->sheetName());

}

QXlsx::AbstractSheet\* Convertor:: **GetActivetWorkSheet**() const

{

QTextStream qtout(stdout);

qtout<<this->ActiveSheet->sheetName()<<endl;

return this->ActiveSheet;

}

QXlsx::Document\* Convertor:: **GetBook**() const

{

return this->xlsxR;

}

QStringList Convertor::**OpenWorkSheet**()

{

cout<<"Sheet is found"<<endl;

QStringList list\_;

QTextStream qtout(stdout);

this->sheet\_count;

for (auto i: this->xlsxR->sheetNames())

{

list\_.push\_back(i);

cout<<sheet\_count<<" ";

qtout<<i<<endl;

sheet\_count++;

}

return list\_;

}

//pharse

void Convertor::**Generate**()

{

this->GetDigitalStandartName();

this->GetData();

this->CreateBlockByName();

}

void Convertor::**GetDigitalStandartName**()

{

for (size\_t r = 2; r < maxRows; r++)

{

Cell\* cell = this->GetBook()->cellAt(r, 1);

Names.emplace(cell->value());

}

if (this->PrintNameFlag)

{

PrintNames();

}

};

void Convertor::**GetData**()

{

int r = 2;

auto Names\_it = Names.begin();

vector<vector<int>> Datablock;

QTextStream qtout(stdout);

for (; r < maxRows; r++)

{

for (int c = 2; c <maxCols; c++)

{

// при 0 - падает

Data.insert(this->xlsxR->cellAt(r, 1)->value().toString(), this->xlsxR->cellAt(r, c)->value().toInt());

}

}

if (PrintDataFlag)

{

PrintData();

}

};

void Convertor::**CreateBlockByName**()

{

DataBlock.resize((Names.size()));

auto it = Names.begin();

QMultiMap<QString, int > ::iterator Data\_it=Data.begin();

for (size\_t i = 0; i < Names.size(); i++)

{

DataBlock[i].reserve(1000);

}

int i = 0;

for (; Data\_it!=Data.end();Data\_it++)

{

if (Data\_it.key() != \*it && i < Names.size() - 1)

{

DataBlock[i].shrink\_to\_fit();

i++;

it++;

/\*cout << k.first;\*/

}

DataBlock[i].push\_back(Data\_it.value());

}

PrintBlocksToFile();

}

//out

//to console

void Convertor::**PrintNames**()

{

QTextStream qtout(stdout);

for (auto it : Names)

{

qtout << it.toString() << endl;

}

}

void Convertor::**PrintData**()

{ QTextStream qtout(stdout);

QMultiMap<QString, int > ::iterator it=Data.begin();

for (;it!=Data.end();it++)

{

qtout<<it.key();

qtout<<it.value()<<endl;

}

}

//to file

void Convertor::**PrintBlocksToFile**()

{

auto it\_Names = Names.begin();

auto it\_Data\_Block=DataBlock.begin();

// Текущий json объект, с которым производится работа

QJsonObject m\_currentJsonObject;

QJsonArray listArray;

// Создаём объект текста

int c=0;

int r=0;

int current\_size;

QJsonArray textsArray = m\_currentJsonObject[it\_Names->toString()].toArray();

for (;it\_Data\_Block!=DataBlock.end();it\_Data\_Block++,c++) {

r=DataBlock[c].size();

for(vector<int>::iterator it2 = (\*it\_Data\_Block).begin() ; it2 != (\*it\_Data\_Block).end(),r-4>=0; ++it2 ,r -= 4){

vector<int> tmp\_v;

int tmp=r-1;

for (int var = 0; var <4 ; var++) {

if (tmp-var>=0) {

textsArray.push\_back(DataBlock[c][tmp-var]);

}

;

}

listArray.push\_back(textsArray);

current\_size =textsArray.size();

for (int i = 0; i <current\_size; i++) {

for (int var = 0; var <current\_size; var++) {

textsArray.removeAt(var);

}

m\_currentJsonObject[it\_Names->toString()] = listArray;

}

//

}

it\_Names++;

int size\_=listArray.size();

for (int i = 0; i <size\_; i++) {

listArray.pop\_back();

}

// Добавляем объект текста в массив

// Сохраняем массив обратно в текущий объект

}

QString saveFileName=QString::fromStdString(this->f\_str\_name.toStdString()+".json");

// Создаём объект файла и открываем его на запись

QFile jsonFile(saveFileName);

if (!jsonFile.*open*(QIODevice::WriteOnly))

{

cout<<"error"<<endl;

return;

}

// Записываем текущий объект Json в файл

jsonFile.write(QJsonDocument(m\_currentJsonObject).toJson(QJsonDocument::Indented));

jsonFile.*close*(); // Закрываем файл

cout<<"file correct create"<<endl;

}

### ***Convertor::Convertor(QString f\_str\_path, bool f1, bool f2)***

Convertor::**Convertor**(QString f\_str\_path, bool f1, bool f2) :PrintDataFlag(f1), PrintNameFlag(f2),sheet\_count(0),f\_str\_path(f\_str\_path)

{

xlsxR= new QXlsx::Document(f\_str\_path);

string s=f\_str\_path.toStdString();

s.erase(s.begin()+s.find("."),s.end());

this->f\_str\_name=QString::fromStdString( s );

}

- конструктор класса Convertor

xlsxR - объект библиотеки QXlsx для работы с .xlsx файлами

string s=f\_str\_path.toStdString();

s.erase(s.begin()+s.find("."),s.end());

this->f\_str\_name=QString::fromStdString( s );

- получение имени файла без расширения **.xlsx** – т.е. **D:\My\_Documents\** **example**

### ***QString Convertor:: ParsePath(QString path)***

QString Convertor:: **ParsePath**(QString path)

{

// QTextStream s(stdout);

// s<< QFileInfo (path).absolutePath()<<endl;

return QFileInfo (path).absoluteFilePath();

}

- преобразование введенного пользователем файла в приемлемый для парсинга формат.

(см. <https://doc.qt.io/archives/qt-4.8/qfileinfo.html>)

### ***void Convertor::OpenBook()***

void Convertor::**OpenBook**()

{

if (this->xlsxR->load())

{

cout << "file is open" << endl;

}

else {

cout<<" Cant open File"<<endl;

exit(-1);

}

}

- метод, проверяющий результат открытия файла (получилось ли открыть файл)

load() – внутрений метод бмблиотеки QXlsx

### ***void Convertor::SetMaxRows( )***

void Convertor::**SetMaxRows**( )

{

int row = 1;

Cell\* cell = this->xlsxR->cellAt(row, 1); // get cell pointer.

while (cell!=nullptr) {

row++;

cell = this->xlsxR->cellAt(row, 1);

}

this->maxRows=row;

cout<<"row "<<this->maxRows<<endl;

}

- получение количества заполненных (не пустых) строк на странице

(**Важно: внутрений метод бмблиотеки QXlsx – cellAt(int row, int column) – принимает индексы не с 0, а с 1)**

### ***void Convertor::SetMaxColumns( )***

void Convertor::**SetMaxColumns**( ) – аналогично void Convertor::**SetMaxRows**( )

### ***int Convertor:: GetSheetCount() const***

int Convertor:: **GetSheetCount**() const

{

return this->sheet\_count;

}

- возвращает количество найденных страниц в документе.   
Используется в :

if (ch\_number>=0 && ch\_number<= cnv.GetSheetCount() ) {

cnv.SetActivetWorkSheet(sh\_list[ch\_number]);

}

- в main.cpp

### ***void Convertor::SetActivetWorkSheet(QString p)***

void Convertor::**SetActivetWorkSheet**(QString p)

{

this->ActiveSheet= this->xlsxR->sheet(p);

this->xlsxR->selectSheet(this->ActiveSheet->sheetName());

}

* Установка выбранной пользователем страницы, как активной для чтения (страницы с которой будет производиться чтение)

### ***QXlsx::AbstractSheet\* Convertor:: GetActivetWorkSheet() const***

QXlsx::AbstractSheet\* Convertor:: **GetActivetWorkSheet**() const

{

QTextStream qtout(stdout);

qtout<<this->ActiveSheet->sheetName()<<endl;

return this->ActiveSheet;

}

- выводит информацию о выбранное странице – имя страницы, позволяет получить ссылку на объект страницы .

### ***QXlsx::Document\* Convertor:: GetBook() const***

QXlsx::Document\* Convertor:: **GetBook**() const

{

return this->xlsxR;

}

- public метод, позволяющий получить ссылку на private QXlsx объект, для работы с .xlsx файлом.

### ***QStringList Convertor::OpenWorkSheet()***

QStringList Convertor::**OpenWorkSheet**()

{

cout<<"Sheet is found"<<endl;

QStringList list\_;

QTextStream qtout(stdout);

this->sheet\_count;

for (auto i: this->xlsxR->sheetNames())

{

list\_.push\_back(i);

cout<<sheet\_count<<" ";

qtout<<i<<endl;

sheet\_count++;

}

return list\_;

}

- открытие ранее, установленной в void Convertor::**SetActivetWorkSheet**(QString p) – странице.

### ***void Convertor::Generate()***

void Convertor::**Generate**()

{

this->GetDigitalStandartName();

this->GetData();

this->CreateBlockByName();

}

- - общий метод, вызывающий все private методы для парсинга.

### ***void Convertor::GetDigitalStandartName()***

void Convertor::**GetDigitalStandartName**()

{

for (size\_t r = 2; r < maxRows; r++)

{

Cell\* cell = this->GetBook()->cellAt(r, 1);

Names.emplace(cell->value());

}

if (this->PrintNameFlag)

{

PrintNames();

}

};

- получение имен из ячеек и добавление их в set Names с целью «отсеивания» одинаковых значений.

### ***void Convertor::GetData()***

void Convertor::**GetData**()

{

int r = 2;

auto Names\_it = Names.begin();

vector<vector<int>> Datablock;

QTextStream qtout(stdout);

for (; r < maxRows; r++)

{

for (int c = 2; c <maxCols; c++)

{

// при 0 - падает

Data.insert(this->xlsxR->cellAt(r, 1)->value().toString(), this->xlsxR->cellAt(r, c)->value().toInt());

}

}

if (PrintDataFlag)

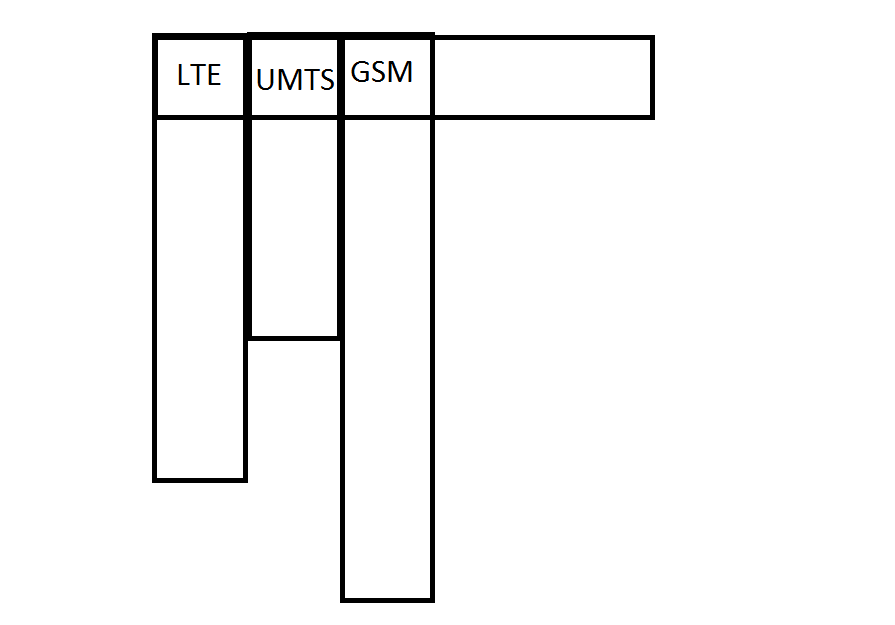
{

PrintData();

}

};

- создание двухмерного вектора, где каждый столбец соответствует уникальному имени



### ***void Convertor::PrintNames()***

void Convertor::**PrintNames**()

{

QTextStream qtout(stdout);

for (auto it : Names)

{

qtout << it.toString() << endl;

}

}

### ***void Convertor::PrintData()***

void Convertor::**PrintData**()

{ QTextStream qtout(stdout);

QMultiMap<QString, int > ::iterator it=Data.begin();

for (;it!=Data.end();it++)

{

qtout<<it.key();

qtout<<it.value()<<endl;

}

}

- вывод данных в консоль.

### ***void Convertor::PrintBlocksToFile()***

void Convertor::**PrintBlocksToFile**()

{

auto it\_Names = Names.begin();

auto it\_Data\_Block=DataBlock.begin();

// Текущий json объект, с которым производится работа

QJsonObject m\_currentJsonObject;

QJsonArray listArray;

// Создаём объект текста

int c=0;

int r=0;

int current\_size;

QJsonArray textsArray = m\_currentJsonObject[it\_Names->toString()].toArray();

for (;it\_Data\_Block!=DataBlock.end();it\_Data\_Block++,c++) {

r=DataBlock[c].size();

for(vector<int>::iterator it2 = (\*it\_Data\_Block).begin() ; it2 != (\*it\_Data\_Block).end(),r-4>=0; ++it2 ,r -= 4){

vector<int> tmp\_v;

int tmp=r-1;

for (int var = 0; var <4 ; var++) {

if (tmp-var>=0) {

textsArray.push\_back(DataBlock[c][tmp-var]);

}

;

}

listArray.push\_back(textsArray);

current\_size =textsArray.size();

for (int i = 0; i <current\_size; i++) {

for (int var = 0; var <current\_size; var++) {

textsArray.removeAt(var);

}

m\_currentJsonObject[it\_Names->toString()] = listArray;

}

//

}

it\_Names++;

int size\_=listArray.size();

for (int i = 0; i <size\_; i++) {

listArray.pop\_back();

}

// Добавляем объект текста в массив

// Сохраняем массив обратно в текущий объект

}

QString saveFileName=QString::fromStdString(this->f\_str\_name.toStdString()+".json");

// Создаём объект файла и открываем его на запись

QFile jsonFile(saveFileName);

if (!jsonFile.*open*(QIODevice::WriteOnly))

{

cout<<"error"<<endl;

return;

}

// Записываем текущий объект Json в файл

jsonFile.write(QJsonDocument(m\_currentJsonObject).toJson(QJsonDocument::Indented));

jsonFile.*close*(); // Закрываем файл

cout<<"file correct create"<<endl;

}

Разбиение целостного массива – на подмассивы по 4 элемента и вывод их в файл с помощью QJsonObject.

-

## Файл Converter.h

#pragma once

#include <string>

#include <set>

#include <iostream>

#include <iomanip>

#include <vector>

#include <map>

#include <QtCore>

#include "xlsxdocument.h"

using namespace std;

class **Convertor**

{

private:

/////parsing/////

void **GetDigitalStandartName**();

void **GetData**();

void **CreateBlockByName**();

void **PrintBlocksToFile**();

///////////////////////////

QString f\_str\_name;

QString f\_str\_path;

size\_t maxRows;

size\_t maxCols;

bool PrintDataFlag,PrintNameFlag;

set<QVariant> Names;

QMultiMap<QString, int > Data;

vector<vector<int>> DataBlock;

QXlsx::Document\* xlsxR;

QXlsx::AbstractSheet\* ActiveSheet;

int sheet\_count;

void **PrintNames**();

void **PrintData**();

public:

static QString **ParsePath**(QString path);

void **OpenBook**();

QXlsx::Document\* **GetBook**() const;

QStringList **OpenWorkSheet**();

void **SetActivetWorkSheet**(QString);

QXlsx::AbstractSheet\* **GetActivetWorkSheet**() const;

int **GetSheetCount**() const;

void **SetMaxRows**();

void **SetMaxColumns**();

void **Generate**();

**Convertor**(QString f\_str\_path, bool f1, bool f2) ;

};

#### QString f\_str\_name; - имя файла

#### *QString f\_str\_path;* - путь к файлу

#### **size\_t maxRows;** - максимальное количество строк

#### **size\_t maxCols;** - максимальное количество столбцов

#### **bool PrintDataFlag**,PrintNameFlag; - флаги указывающие на необходимость вывода в консоль

#### set<QVariant> Names; - множество уникальных имен.

#### QMultiMap<QString, int > Data; - multimap - пара имя-значение

#### vector<vector<int>> DataBlock; - все данные для уникального имени.

#### QXlsx::Document\* xlsxR; - объект QXlsx для работы с .xlsx файлом

#### QXlsx::AbstractSheet\* ActiveSheet; - текщая выбранная страница

#### int sheet\_count; - всего страниц