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

|  |
| --- |
| 1.1.2019 |

Оглавление

[Введение 1](#_Toc16871812)

[Приложение «Excel\_to\_Json» 1](#_Toc16871813)

[Описание назначения приложения 1](#_Toc16871814)

[Особенности использования 3](#_Toc16871815)

[Техническая часть 11](#_Toc16871816)

[Структура проекта 11](#_Toc16871817)

[Файл main.cpp 11](#_Toc16871818)

[Файл SettingsConverter.cpp 11](#_Toc16871819)

[Файл SettingsConverter.h 11](#_Toc16871820)

[Файл main.cpp 12](#_Toc16871821)

[***cnv.* calculateNotEmptyRowsCount *();*** 14](#_Toc16871822)

[***cnv.* calculateNotEmptyColumnsCount *();*** 14](#_Toc16871823)

[***cnv.* convert *();*** 14](#_Toc16871824)

[Файл SettingsConverter.cpp 15](#_Toc16871825)

[Файл SettingsConverter.h 22](#_Toc16871826)

# 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 .

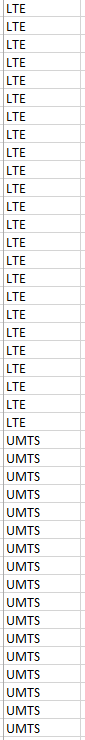
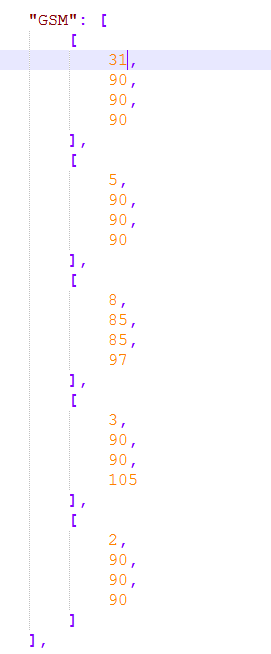
 

Рисунок 2. Обработанные данные в формате .json

Рисунок 1. Данные таблицы из файла формата .xlsx

### Особенности использования

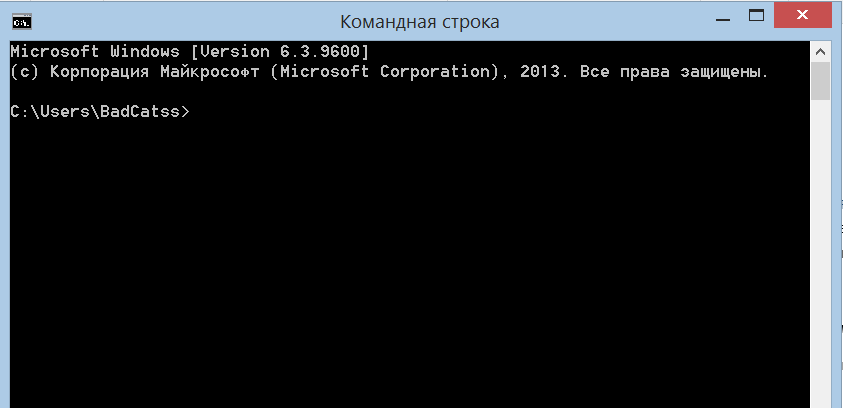
Приложение не имеет графического интерфейса, а предоставляет к использованию – интерфейс командной строки, так же иногда называемый - «консоль», в среде операционной системы

(рис. 3).

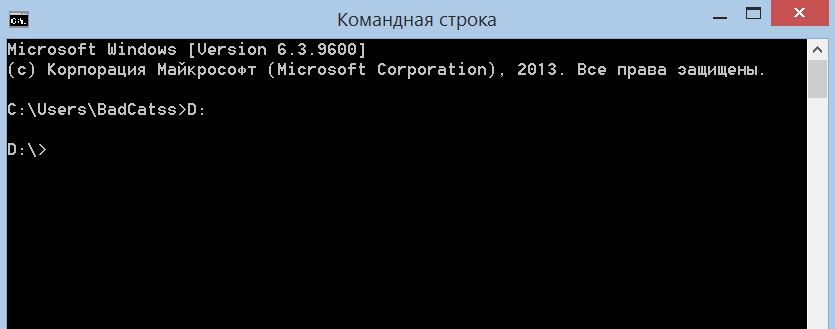
Приложение имеет два режима роботы:

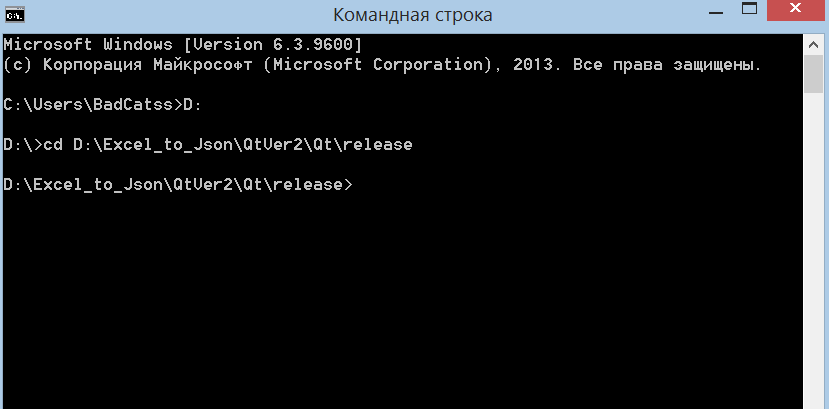
1) Запуск приложения из командной строки с передачей параметра при запуске

1. Откройте команду директорию

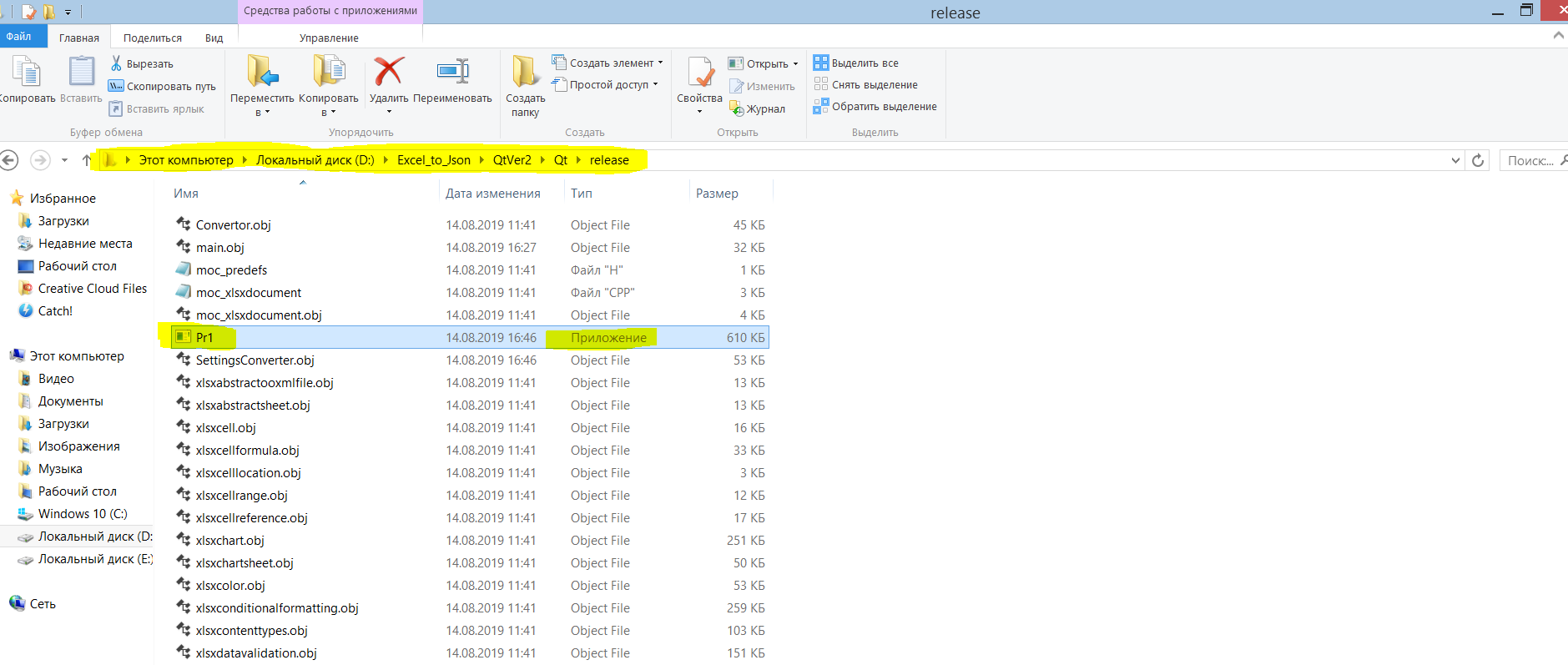


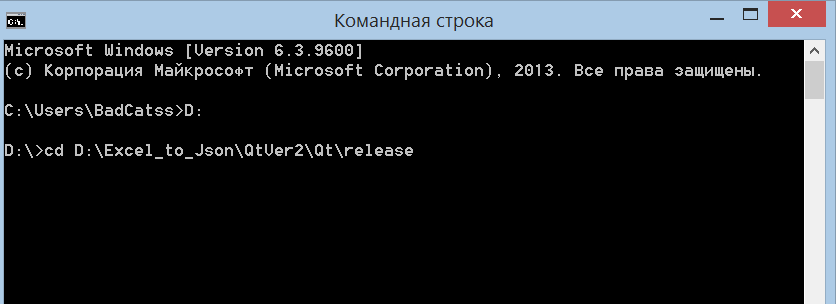
2. Перейдите в директорию release проекта

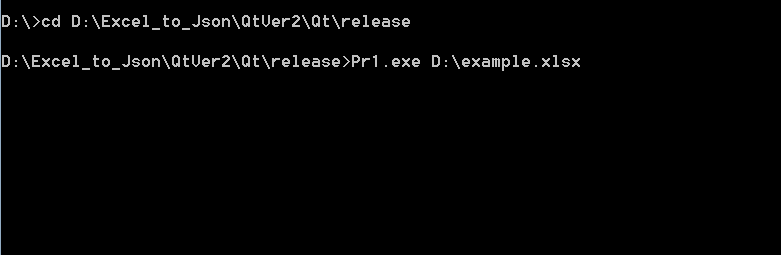


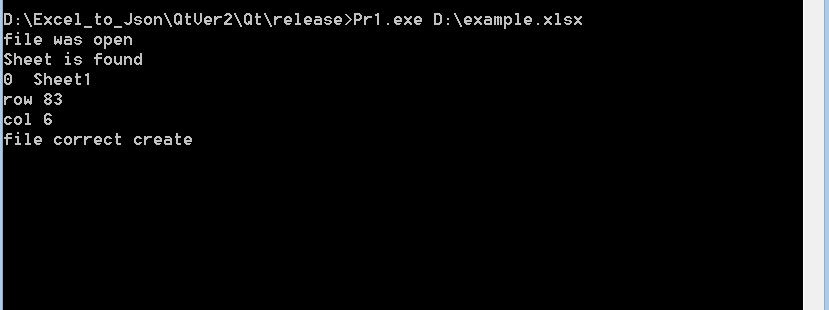


Введите имя файла приложения в консоль, указав так же расширение .exe и добавив через пробел дополнительный параметр в виде пути к файлу, который необходимо конвертировать









2) Непосредственный запуск приложения с последующим вводом пути к файлу, который будет конвертирован.

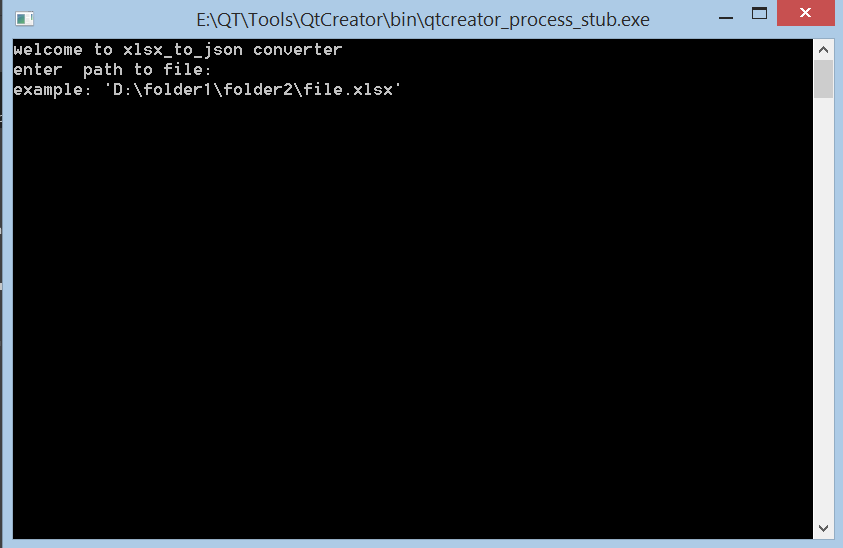


Рисунок 3. Пример работы приложения

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:

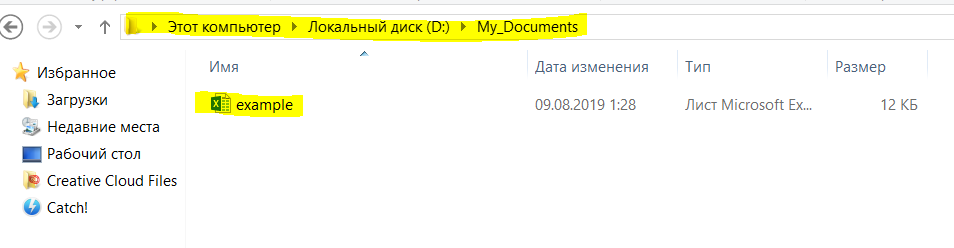
**example**

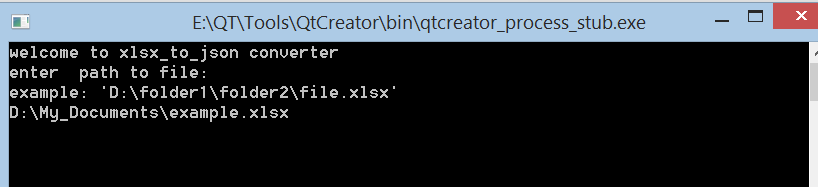
Следовательно, абсолютный путь к файлу будет следующим:

**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)**

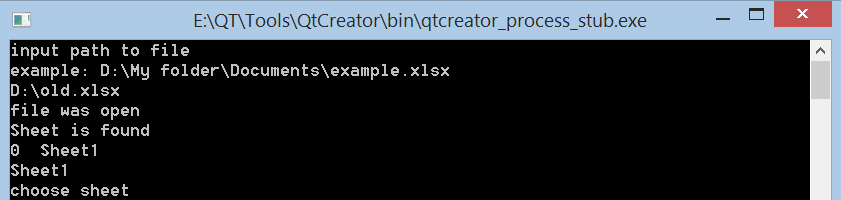




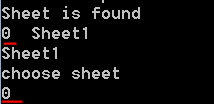
Введите путь к фалу и нажмите клавишу “enter”.

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.



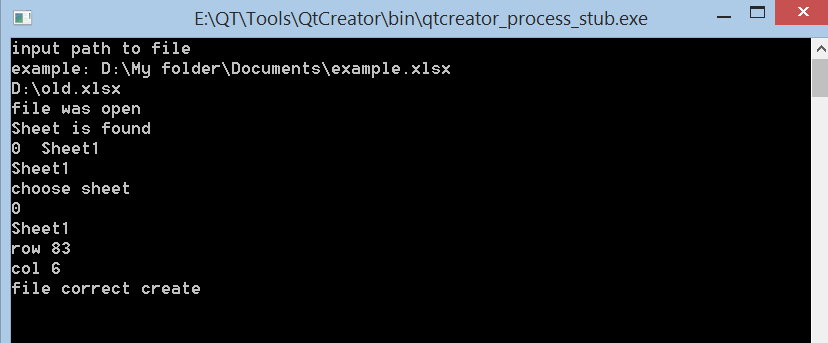
При открытии страницы – будет выведенно имя выбранной страницы и количество строк и столбцов в файле, которые заполнены данными(не пусты)



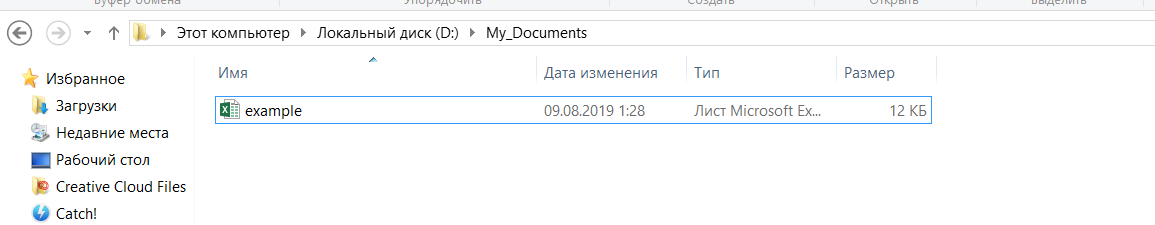
При корректном создании .json файла – будет выведенно соответствующее сообщение



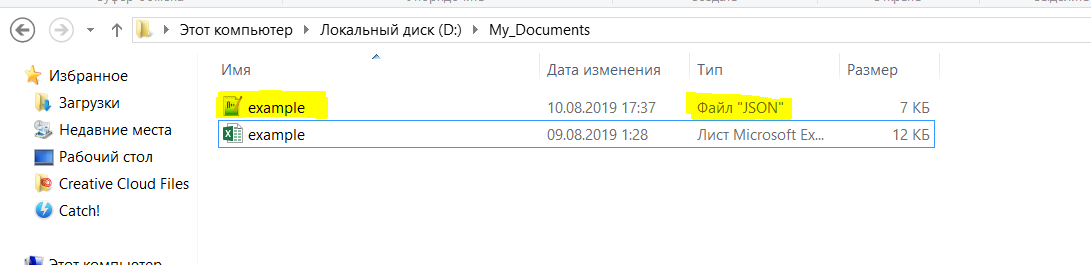
Вид всех сообщений диалога:



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 SettingsConverter.cpp

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

### Файл SettingsConverter.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 <QCoreApplication>

#include "SettingsConverter.h"

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

{

QCoreApplication a(*argc*, argv);

if (argc ==2) {

SettingsConverter cnv(QCoreApplication::arguments().at(1));

if (cnv.openBook())

{

cout << "file was open" << endl;

auto sheetList = cnv.getSheetsList();

cnv.setActivetWorkSheet(sheetList[0]);

if (cnv.convert()) {

a.exec();

a.exit(0);

}

else {

cnv.printErrorMesseges();

exit(-1);

}

} //argc ==2 // if(cnv.openBook())

else {

cnv.printErrorMesseges();

exit(-1);

}

} // if(argc == 2)

else {

string userInput;

cout << "input path to file" << endl;

cout << "example: D:\\My folder\\Documents\\example.xlsx" << endl;

getline(*cin*,*userInput*);

SettingsConverter cnv(QString::fromStdString(userInput));

if (cnv.openBook()) {

cout << "file was open" << endl;

auto sheetList = cnv.getSheetsList();

for (auto listElement : sheetList)

{

cout << listElement.toStdString() << endl;

}

int userInputTry = 0;

while (userInputTry == 0) {

cout << "choose sheet" << endl;

int chooseNumber;

cin >> chooseNumber;

if (chooseNumber >= 0 && chooseNumber <= sheetList.size()) {

userInputTry++;

cnv.setActivetWorkSheet(sheetList[chooseNumber]);

cout << sheetList[chooseNumber].toStdString() << endl;

if (cnv.convert()) {

a.exit(0);

}

else {

cnv.printErrorMesseges();

a.exit(-1);

}

}

else {

cout << "incorrect list number" << endl;

}

}

} //if(argc != 2) // if(cnv.openBook())

else {

cnv.printErrorMesseges();

exit(-1);

}

}

}

if (argc ==2) - проверка режима работы приложения.

getline(*cin*,*userInput*); - ввод пользователем строки к файлу.

SettingsConverter cnv(QString::fromStdString(userInput)); - создание объекта класса отвечающего за конвертирование.

if (cnv.openBook()) – проверка корректности введенного пути и возможности открытия файла.

***фрагмент кода, в main.cpp:***

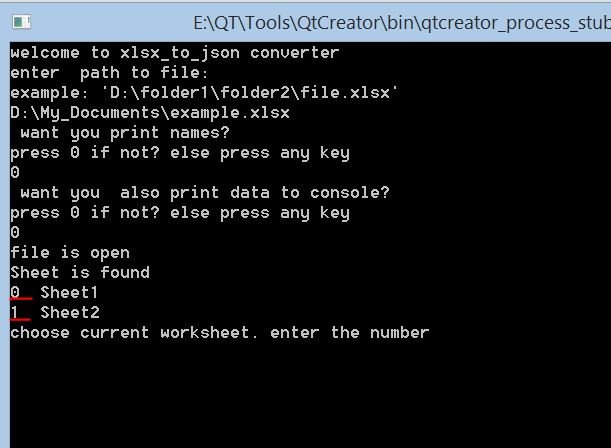
cnv.printErrorMesseges(); - вывод возможных ошибок возникших при конвертировании

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

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

}

- вывод найденых листов в файле.



### ***cnv.* calculateNotEmptyRowsCount *();***

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

### ***cnv.* calculateNotEmptyColumnsCount *();***

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

### ***cnv.* convert *();***

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

## Файл SettingsConverter.cpp

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

#include <QCoreApplication>

#include <QCoreApplication>

#include <QDebug>

#include "xlsxchartsheet.h"

#include "xlsxcellrange.h"

#include "xlsxchart.h"

#include "xlsxrichstring.h"

#include "xlsxworkbook.h"

#include "SettingsConverter.h"

using namespace QXlsx;

using namespace std;

void SettingsConverter:: **addErrorToList**(QString errorMessege)

{

this->errorValues.push\_back(errorMessege);

}

void SettingsConverter:: **printErrorMesseges**()

{

for (auto errorMessege : this->errorValues) {

cout << errorMessege.toStdString() << endl;

}

}

//book and sheet setup

bool SettingsConverter::**openBook**()

{

string tmpString = filePath.toStdString();

int startSearchPosition = tmpString.find(".");

if (tmpString.find("xlsx",startSearchPosition) != std::string::npos) {

xlsxR = new QXlsx::Document(filePath);

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

this->savePath = tmpString;

}

else {

addErrorToList("incorrect format");

return false;

}

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

{

return true;

}

else {

addErrorToList("cant open file");

return false;

}

}

void SettingsConverter::**calculateNotEmptyRowsCount**( )

{

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 SettingsConverter::**calculateNotEmptyColumnsCount**( )

{

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;

}

void SettingsConverter::**setActivetWorkSheet**(const QString& chosenSheet)

{

this->activeSheet = this->xlsxR->sheet(chosenSheet);

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

}

QStringList SettingsConverter::**getSheetsList**()

{

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

QStringList sheetList;

QTextStream qtout(stdout);

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

{

sheetList.push\_back(sheet);

cout << sheetCount << " ";

qtout << sheet << endl;

sheetCount++;

}

return sheetList;

}

//pharse

SettingsConverter::**SettingsConverter**(const QString& param)

{

QFileInfo info(param);

filePath = info.absoluteFilePath();

}

bool SettingsConverter:: **convert**()

{

this->calculateNotEmptyRowsCount();

this->calculateNotEmptyColumnsCount();

this->createJsonObject();

if ( !this->readXlsxFile() || !this->createJsonObject() || !this->writeJsonFile()) {

this->addErrorToList("convert error");

return false;

}

else {

return true;

}

}

bool SettingsConverter:: **readXlsxFile**()

{

bool valueIsCorrect = true;

for ( int r = 2; r < maxRows; ++r) {

QJsonArray arrayOfValue;

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

if (this->xlsxR->cellAt(r,1)->readValue().toString() == "UMTS" && c == maxCols - 1 ) {

arrayOfValue.append((this->xlsxR->cellAt(r,c)->readValue().toJsonValue()));

arrayOfValue.append(30);

arrayOfValue.append(30);

}

else {

arrayOfValue.append((this->xlsxR->cellAt(r,c)->readValue().toJsonValue()));

}

}

valuesMap[this->xlsxR->cellAt(r,1)->readValue().toString()].append(arrayOfValue);

}

for (auto currentArray : valuesMap) {

for (auto currentValue : currentArray) {

if (currentValue.isNull()) {

valueIsCorrect = false;

}

}

}

if (valueIsCorrect && valuesMap.size() != 0) {

return true;

}

else {

this->addErrorToList("read error");

return false;

}

}

bool SettingsConverter::**createJsonObject**()

{

for (auto currentKey : valuesMap.keys()) {

this->valueJsonObject[currentKey] = valuesMap.take(currentKey);

}

if (this->valueJsonObject.size() != 0) {

return true;

}

else {

this->addErrorToList("JsonObject create error");

return false;

}

}

bool SettingsConverter::**writeJsonFile**()

{

QString saveFileName = QString::fromStdString(this->savePath + ".json");

// Создаём объект файла и открываем его на запись

QFile jsonFile(saveFileName);

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

{

addErrorToList("write error");

return false;

}

// Записываем текущий объект Json в файл

jsonFile.write(QJsonDocument(this->valueJsonObject).toJson(QJsonDocument::Indented));

jsonFile.*close*(); // Закрываем файл

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

return true;

}

### ***void SettingsConverter:: addErrorToList(QString errorMessege)***

|  |  |
| --- | --- |
| ***Принимаемое значение*** | QString |
| ***Возвращаемое значение*** | void |

void SettingsConverter:: **addErrorToList**(QString errorMessege)

{

this->errorValues.push\_back(errorMessege);

}

* Метод, отвечающий за добавление возможных возникших ошибок при чтении/записи в единый список, который будет выведен, в таком случае.

### ***void SettingsConverter:: printErrorMesseges()***

|  |  |
| --- | --- |
| ***Принимаемое значение*** | Нет (void) |
| ***Возвращаемое значение*** | void |

void SettingsConverter:: **printErrorMesseges**()

{

for (auto errorMessege : this->errorValues) {

cout << errorMessege.toStdString() << endl;

}

} – метод, отвечающий за вывод списка возникших ошибок.

***( Важно уточнить, что формулировка:***

|  |  |
| --- | --- |
| ***Принимаемое значение*** | Нет (void) |

***- строго справедливо только для c++ . )***

### ***bool SettingsConverter::openBook()***

|  |  |
| --- | --- |
| ***Принимаемое значение*** | Нет (void) |
| ***Возвращаемое значение*** | bool |

bool SettingsConverter::**openBook**()

{

string tmpString = filePath.toStdString();

int startSearchPosition = tmpString.find(".");

if (tmpString.find("xlsx",startSearchPosition) != std::string::npos) {

xlsxR = new QXlsx::Document(filePath);

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

this->savePath = tmpString;

}

else {

addErrorToList("incorrect format");

return false;

}

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

{

return true;

}

else {

addErrorToList("cant open file");

return false;

}

} - метод, проверяющий корректность введенного пути (точнее, корректности указанного формата файла:  
if (tmpString.find("xlsx",startSearchPosition) != std::string::npos)) , а так же возможности открытия указанного файла.

### ***void SettingsConverter::calculateNotEmptyRowsCount( )***

|  |  |
| --- | --- |
| ***Принимаемое значение*** | Нет (void) |
| ***Возвращаемое значение*** | ***void*** |

void SettingsConverter::**calculateNotEmptyRowsCount**( )

{

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;

} – метод, который считает количество заполненных (не пустых) строк.

### ***void SettingsConverter::calculateNotEmptyColumnsCount( )***

|  |  |
| --- | --- |
| ***Принимаемое значение*** | Нет (void) |
| ***Возвращаемое значение*** | ***void*** |

void SettingsConverter::**calculateNotEmptyColumnsCount**( )

{

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;

} – метод, который считает количество заполненных (не пустых) столбцов.

### ***void SettingsConverter::setActivetWorkSheet(const QString& chosenSheet)***

|  |  |
| --- | --- |
| ***Принимаемое значение*** | ***const QString&*** |
| ***Возвращаемое значение*** | ***void*** |

void SettingsConverter::**setActivetWorkSheet**(const QString& chosenSheet)

{

this->activeSheet = this->xlsxR->sheet(chosenSheet);

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

} – устанавливает выбранную пользователем страницу, как активную для чтения.

### ***QStringList SettingsConverter::getSheetsList()***

|  |  |
| --- | --- |
| ***Принимаемое значение*** | Нет (void) |
| ***Возвращаемое значение*** | ***QStringList*** |

QStringList SettingsConverter::**getSheetsList**()

{

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

QStringList sheetList;

QTextStream qtout(stdout);

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

{

sheetList.push\_back(sheet);

cout << sheetCount << " ";

qtout << sheet << endl;

sheetCount++;

}

return sheetList;

}

- получает и выводит список найденных в файле страниц.

### ***SettingsConverter::SettingsConverter(const QString& param)***

|  |  |
| --- | --- |
| ***Принимаемое значение*** | ***const QString&*** |
| ***Возвращаемое значение*** | ***нет*** |

SettingsConverter::**SettingsConverter**(const QString& param)

{

QFileInfo info(param);

filePath = info.absoluteFilePath();

} – конструктор класса **SettingsConverter**

### ***bool SettingsConverter:: convert()***

|  |  |
| --- | --- |
| ***Принимаемое значение*** | Нет (void) |
| ***Возвращаемое значение*** | ***bool*** |

bool SettingsConverter:: **convert**()

{

this->calculateNotEmptyRowsCount();

this->calculateNotEmptyColumnsCount();

this->createJsonObject();

if ( !this->readXlsxFile() || !this->createJsonObject() || !this->writeJsonFile()) {

this->addErrorToList("convert error");

return false;

}

else {

return true;

}

}

- публичный метод, вызывающий все, необходимые для конвертации, приватные методы.

### ***bool SettingsConverter:: readXlsxFile()***

|  |  |
| --- | --- |
| ***Принимаемое значение*** | Нет (void) |
| ***Возвращаемое значение*** | ***bool*** |

bool SettingsConverter:: **readXlsxFile**()

{

bool valueIsCorrect = true;

for ( int r = 2; r < maxRows; ++r) {

QJsonArray arrayOfValue;

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

if (this->xlsxR->cellAt(r,1)->readValue().toString() == "UMTS" && c == maxCols - 1 ) {

arrayOfValue.append((this->xlsxR->cellAt(r,c)->readValue().toJsonValue()));

arrayOfValue.append(30);

arrayOfValue.append(30);

}

else {

arrayOfValue.append((this->xlsxR->cellAt(r,c)->readValue().toJsonValue()));

}

}

valuesMap[this->xlsxR->cellAt(r,1)->readValue().toString()].append(arrayOfValue);

}

for (auto currentArray : valuesMap) {

for (auto currentValue : currentArray) {

if (currentValue.isNull()) {

valueIsCorrect = false;

}

}

}

if (valueIsCorrect && valuesMap.size() != 0) {

return true;

}

else {

this->addErrorToList("read error");

return false;

}

} - метод, выполняющий чтение данных из файла.

### ***bool SettingsConverter::createJsonObject()***

|  |  |
| --- | --- |
| ***Принимаемое значение*** | Нет (void) |
| ***Возвращаемое значение*** | ***bool*** |

bool SettingsConverter::**createJsonObject**()

{

for (auto currentKey : valuesMap.keys()) {

this->valueJsonObject[currentKey] = valuesMap.take(currentKey);

}

if (this->valueJsonObject.size() != 0) {

return true;

}

else {

this->addErrorToList("JsonObject create error");

return false;

}

} - метод, создающий Json-объект содержащий прочитанные данные.

### ***bool SettingsConverter::writeJsonFile()***

|  |  |
| --- | --- |
| ***Принимаемое значение*** | Нет (void) |
| ***Возвращаемое значение*** | ***bool*** |

bool SettingsConverter::**writeJsonFile**()

{

QString saveFileName = QString::fromStdString(this->savePath + ".json");

// Создаём объект файла и открываем его на запись

QFile jsonFile(saveFileName);

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

{

addErrorToList("write error");

return false;

}

// Записываем текущий объект Json в файл

jsonFile.write(QJsonDocument(this->valueJsonObject).toJson(QJsonDocument::Indented));

jsonFile.*close*(); // Закрываем файл

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

return true;

}

- запись созданного Json-объекта в файл.

## Файл SettingsConverter.h

#pragma once

#include <string>

#include <iostream>

#include <iomanip>

#include <map>

#include <QtCore>

#include <QJsonArray>

#include "xlsxdocument.h"

using namespace std;

class **SettingsConverter**

{

size\_t maxRows = 0;

size\_t maxCols = 0;

QXlsx::Document\* xlsxR = nullptr;

QXlsx::AbstractSheet\* activeSheet = nullptr;

QString filePath = "";

string savePath = "";

int sheetCount = 0;

QMap<QString, QJsonArray> valuesMap;

QVector<QString> errorValues;

QJsonObject valueJsonObject;

void **calculateNotEmptyRowsCount**();

void **calculateNotEmptyColumnsCount**();

void **addErrorToList**(QString error);

bool **readXlsxFile**();

bool **createJsonObject**();

bool **writeJsonFile**();

public:

bool **convert**();

bool **openBook**();

void **setActivetWorkSheet**(const QString&);

void **printErrorMesseges**();

QStringList **getSheetsList**();

**SettingsConverter**(const QString& p);

};

size\_t maxRows = 0; - количество найденных непустых строк.

size\_t maxCols = 0; - количество найденных непустых стобцов.

QXlsx::Document\* xlsxR = nullptr; - указатель на объект типа Document – библиотеки QXlsx, отвечающий за работу с файлом.

QXlsx::AbstractSheet\* activeSheet = nullptr; - указатель на объект типа AbstractSheet - библиотеки QXlsx, отвечающий за работу с текущей, активной станицей.

QString filePath = ""; - путь к файлу (включая имя и .xlsx файла)

string savePath = ""; - путь к файлу (без расширения файла)

int sheetCount = 0; - количество найденных страниц в файле.

QMap<QString, QJsonArray> valuesMap; - считанные значения.

QVector<QString> errorValues; - «список» возникших ошибок

QJsonObject valueJsonObject; - Json-объект записываемый в создаваемый файл.