Static File Analysis Tool

Technical Manual

Brian Tobin

C00216353

Table of Contents

Introduction

This manual details the technologies and versions used to develop the application, some instructions to install and use the application, and finally, all of the code for the application.

Development Technologies and Tools

Technologies

Qt version: 5.14.2

C++ compiler: mingw73_64

Tools

QT Creator 4.11.1

Git version 2.25.0.windows.1

Installation

The application can be downloaded from github: https://github.com/briantobin256/Static-File-Analysis-Tool/releases. The binaries provided have no dependencies, they should run on any 64-bit Windows machine directly.

System Usage

The application home page tells the user to open and follow the built in checklist. This goes into detail about what the user should be doing and where to start.

Code

StaticFileAnalysisToll.pro

```
QT       += core gui charts

win32:RC_ICONS += icon.ico

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets


CONFIG += c++11


# The following define makes your compiler emit warnings if you use

# any Qt feature that has been marked deprecated (the exact warnings

# depend on your compiler). Please consult the documentation of the

# deprecated API in order to know how to port your code away from it.

DEFINES += QT_DEPRECATED_WARNINGS


# You can also make your code fail to compile if it uses deprecated APIs.

# In order to do so, uncomment the following line.

# You can also select to disable deprecated APIs only up to a certain version of Qt.

#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0


SOURCES += \

    customdialog.cpp \

    main.cpp \
```

```
    mainwindow.cpp

HEADERS += \
    customdialog.h \
    mainwindow.h

FORMS += \
    customdialog.ui \
    mainwindow.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

DISTFILES +=
```

MainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QLabel>
```

```
#include "customdialog.h"
#include <QTextBrowser>
#include <QDebug>
#include <QFileDialog>
#include <iostream>
#include <fstream>
#include <QWheelEvent>
#include <QVBoxLayout>
#include <QCheckBox>
#include <QCryptographicHash>
#include <QTableWidgetItem>
#include <QListWidgetItem>
#include <QClipboard>
#include <QMessageBox>
#include <QCoreApplication>
#include <QCloseEvent>
#include <math.h>
#include <QtCharts>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE
```

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

    char *rawData;
    bool fileOpened;
    int fileSize;
    bool checklistOpened;
    bool packed;

    QString basicWindowName;
    QString extendedWindowName;
    QString fileName;
    QString directory;
    QString fileHash;
    QString backupLoc;
```

```
// string things
QMap<int, QString> stringLocationMap;
QMap<int, bool> savedStringMap;
QMap<int, int> savedStringLocationMap;
QMap<int, int> hexLocationMap;
QMap<int, int> swapStringMap;
QStringList strings;
QStringList savedStrings;
QStringList unsortedStrings;
QString previousSearchString;
int stringCount;
int stringOffset;
int maxDisplayStrings;
int totalStringSize;
int totalSavedStringSize;
int searchStringIndex;
int stringLength;
bool sorting;
bool removedStrings;
bool stringsSorted;
bool firstStringsRefresh;
```

```
// DLL things
QString dllNames;
QStringList dllFunctionNames;
QString DLLTitle;
QString FunctionTitle;



//hex things
int hexDisplayRows;
int hexDisplayCols;
int previousPosition;
int byteDisplaySize;
int cursorLocation;
bool savingEdit;
bool nextPage;
bool secondTime;
bool editing;
bool refreshing;
bool dataChanged;
QMap<int, char> originalDataMap;
QMap<int, bool> changedDataMap;
```

```
// initial builds
bool packChecked;
bool stringsBuilt;
bool dllsBuilt;
bool entropyGraphBuilt;



// PE things
bool PE;
int imagebase;
int codeStartLoc, codeEndLoc, codeVirtualAddress;
int rdataStartLoc, rdataRVA;
int idataStartLoc, idataRVA;
int IDTLoc, IDTSize;
int IATLoc;
int codeEntryPoint, baseOfCode;
int dataStartLoc, dataVirtualAddress;



// disassembly
bool disassemblyBuilt;
int maxDisplayInstructions;
QStringList disassembly;
```

```
    QMap<int, int>opTypeMap;
    QMap<int, QString>opcodeMap;
    QMap<QString, int> locOffsetMap;
    int codeStartProcedure;
    QStack<int> jumpStack;

    bool reseting;

private slots:
    void on_actionCheck_if_Packed_triggered();

    void on_actionGenerate_Hash_triggered();

    void on_actionCreate_Backup_triggered();

    void on_actionPack_triggered();

    void on_actionUnpack_triggered();

    void on_actionFind_Strings_triggered();

    void on_actionChecklistMain_triggered();
```

```
void on_actionSaved_Strings_triggered();

void on_actionHex_triggered();

void on_actionOpen_triggered();

void on_hexScrollBar_valueChanged();

void on_actionDLL_s_triggered();

void on_stringsScrollBar_valueChanged();

void on_stringSearchButton_clicked();

void on_actionUndo_All_Changes_triggered();

void on_actionExit_triggered();

void on_actionSave_triggered();

void on_stringList_itemDoubleClicked(QListWidgetItem *item);

void on_actionEntropy_Graph_triggered();
```

```
void on_actionDisassembly_triggered();

void on_disassemblyScrollBar_valueChanged();

void on_disassemblyBrowser_anchorClicked(const QUrl &arg1);

void on_hexByteDisplay_cursorPositionChanged();

void on_hexByteDisplay_textChanged();

void showContextMenu(const QPoint &point);

void copyHighlightedItemsText();

void checkHighlighted();

void uncheckHighlighted();

void removeSelected();

void highlightAll();
```

```
    void sortStrings();

    void outputStrings();

    void stringToHexLocation();

    void on_savedStringSearchButton_clicked();

    void on_DLLTitleBrowser_anchorClicked(const QUrl &arg1);

    void on_DLLFunctionTitleBrowser_anchorClicked(const QUrl &arg1);

    void on_disassemblyStartLocationButton_clicked();

    void on_disassemblySearchButton_clicked();

    void on_disassemblyJumpBackButton_clicked();

private:
    Ui::MainWindow *ui;
    CustomDialog *dialogBox;

    // general ui things
```

```
void refreshWindow();

void resetChecks();

void closeEvent(QCloseEvent *event);

void wheelEvent(QWheelEvent *event);

void showChecklist();

void hideChecklist();

void popChecklist();

void buildChecklist();


// hashing

QString generateHash(char *data, int size);

QString generateFileHash(QString fileName);


// file

void open(QFile *f);

void saveChanges();

void undoChanges();

void getPEinformation();


// packing

bool isPacked();

bool pack();

bool unpack();
```

```
    double getEntropy();
    void buildEntropyGraph();
    double chunkEntropy(int offset, int chunkSize);

    // strings
    void findStrings();
    void saveDisplayedStrings();
    void refreshStrings();
    void refreshSavedStrings();
    bool searchStringList(QString searchString, QStringList *list, bool searchFromBeginning,
bool htmlList);
    QString htmlSanitiseString(QString string);

    // dlls
    void findDLLs();
    QString getFunctionName(int location, int dataSectionRVA, int dataStartLoc);

    // hex
    void refreshHex();
    QString byteToHexString(int c);
    void setHexValues();
    void moveCursor(int direction);
```

```
    // disassembly
    void getDisassembly();
    QStringList disassembleSection(int start, int end, int virtualAddress);
    void refreshDisassembly();
    QString registerName(int reg, int operandSize);
    QString segmentRegisterName(int reg);
    QString getSpecialByteInstruction(int specialByte, int reg);
    QString getExtendedByteInstruction(int extendedByte, int reg);
    int getOperandSize(unsigned char byte, bool operandSizeModifier);
    QString immediateFormat(QString s);
    QString getFunctionCallName(int immediateValue);
    QString immediateIsStringOffset(int immediateValue, int physicalAddress, int
virtualAddress);
};
#endif // MAINWINDOW_H
```

Main.cpp

```
#include "mainwindow.h"

#include <QApplication>

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

```
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

MainWindow.cpp

```
#include "customdialog.h"
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->MainDisplayStack->setCurrentIndex(0);

    basicWindowName = "Static File Analysis Tool";

    // based on application instance rather than current file
    savingEdit = false;
```

```
    fileOpened = false;
    fileHash = "";
    backupLoc = "";

    checklistOpened = true;
    on_actionChecklistMain_triggered();

    resetChecks();
    buildChecklist();
    refreshWindow();
}

MainWindow::~MainWindow()
{
    delete ui;
}

// UI ACTIONS

void MainWindow::on_actionOpen_triggered()
{
    saveChanges();
    QFile file;
```

```
    if (!fileOpened) {
        file.setFileName(QFileDialog::getOpenFileName(this, "Select a file to analyse",
"D:/Downloads"));
    }
    else {
        file.setFileName(QFileDialog::getOpenFileName(this, "Select a file to analyse", directory));
    }
    open(&file);
    file.close();
}

void MainWindow::on_actionSave_triggered()
{
    saveChanges();
}

void MainWindow::on_actionUndo_All_Changes_triggered()
{
    undoChanges();
    refreshWindow();
}

void MainWindow::on_actionGenerate_Hash_triggered()
{
```

```
    saveChanges();
    dialogBox = new CustomDialog();
    dialogBox->setWindowTitle("Hash");

    if (fileOpened) {
        fileHash = generateHash(rawData, fileSize);
        dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>The SHA-1 hash of the current data is:<p style='text-align:center;font-size:20px;color:blue;'>" + fileHash);
    }
    else {
        dialogBox->setText("<p style='text-align:center;font-size:25px;'><br><br>No file selected.");
    }

    dialogBox->exec();
    refreshWindow();
}
```

```
void MainWindow::on_actionCreate_Backup_triggered()
{
    saveChanges();
    dialogBox = new CustomDialog();
    dialogBox->setWindowTitle("Backup");
```

```
    if (fileOpened) {
        QDir dir;
        dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>Please select a location to store backups.");
        dialogBox->exec();
        QFile file(QFileDialog::getExistingDirectory(this, "Select a location to store backups."));
        backupLoc = file.fileName();
        dir.setPath(backupLoc);
        file.close();

        if (backupLoc != "") {

            fileHash = generateHash(rawData, fileSize);

            // check if file exists with current backup name
            QString backupName = fileHash + ".bak";
            bool backupConflict = false;
            int i = 0;
            QFileInfoList list = dir.entryInfoList();
            while (!backupConflict && i < list.size()) {
                QFileInfo fileInfo = list.at(i);
                if (backupName == fileInfo.fileName()) {
```

```
            backupConflict = true;
        }
        i++;
    }

    if (backupConflict) {
        if (backupName == fileName) {
            dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>The current file is the backup file. Rename it to back it up again.");
        }
        else {
            dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>A backup of this file already exists.");
        }
    }
    else {
        // create backup
        QString fullName = backupLoc + 92 + backupName;
        QFile createBackup(fullName);

        if (createBackup.open(QIODevice::ReadWrite)) {
            QDataStream ds(&createBackup);
            ds.writeRawData(rawData, fileSize);
```

```
            createBackup.close();
        }

        // create verficication hash
        QString verificationHash = generateFileHash(fullName);

        if (verificationHash == fileHash) {
            dialogBox->setText("<p style='text-align:center;font-size:25px;'>Backup Succesful.<br>" + fileName + "<br>has been backed up as<p style='text-align:center;font-size:18px;color:blue'>" + fileHash + ".bak");
        }
        else {
            dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>An error has occured, the backup may not exist.");
        }
      }
    }
    else {
      dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>No location selected.");
    }
    dialogBox->exec();
  }
  else {
```

```
        dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>You must first select a file to analyse before anything can be backed up.");
        dialogBox->exec();
    }

    refreshWindow();
}

void MainWindow::on_actionExit_triggered()
{
    saveChanges();
    QApplication::quit();
}

void MainWindow::on_actionCheck_if_Packed_triggered()
{
    saveChanges();
    dialogBox = new CustomDialog();
    dialogBox->setWindowTitle("Check if File is Packed");

    if (fileOpened) {
        double entropy = getEntropy();
        QString basicText = "", entropyVar = "";
```

```
        if (isPacked()) {
            basicText = "<p style='text-align:center;font-size:20px;'>The current file IS packed using UPX.";
        }
        else {
            basicText = "<p style='text-align:center;font-size:20px;'>The current file is NOT packed using UPX!";
        }
        if (entropy < 6.5) {
            entropyVar = "<p style='text-align:center;font-size:20px;'>The files entropy is<br>" + QString::number(entropy) + "<br>which would suggest that the file is NOT packed/compressed/encrypted.";
        }
        else {
            entropyVar = "<p style='text-align:center;font-size:20px;'>The files entropy is<br>" + QString::number(entropy) + "<br>which would suggest that the file IS packed/compressed/encrypted.";
        }
        dialogBox->setText(basicText + entropyVar);
    }
    else {
        dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>No file is selected.");
    }
```

```
    dialogBox->exec();
    packChecked = true;
    refreshWindow();
}

void MainWindow::on_actionPack_triggered()
{
    saveChanges();
    dialogBox = new CustomDialog();
    dialogBox->setWindowTitle("Pack");

    if (fileOpened) {
        if (isPacked()) {
            dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>The current file is already packed using UPX.");
        }
        else {
            // pack the current file
            if (pack()) {
                packChecked = false;
                if (isPacked()) {
                    dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>The current file is now packed using UPX.");
```

```
                // open newly packed file to analyse
                QFile file(directory + fileName);
                open(&file);
                file.close();
            }
            else {
                dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>The current file was not packed using UPX.");
            }
        }
        else {
            dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>The current file was not packed using UPX.");
        }
    }
  }
  else {
    dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>No file is selected.");
  }

  dialogBox->exec();
  refreshWindow();
}
```

```
void MainWindow::on_actionUnpack_triggered()
{
    saveChanges();
    dialogBox = new CustomDialog();
    dialogBox->setWindowTitle("Unpack");

    if (fileOpened) {
        if (isPacked()) {
            // unpack the current file
            if (unpack()) {
                dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>The current file has been unpacked using UPX.");
                packChecked = false;
                // open newly packed file to analyse
                QFile file(directory + fileName);
                open(&file);
                file.close();
            }
            else {
                dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>The current file was not unpacked.");
            }
```

```
        }
        else {
            dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>The current file is not packed using UPX.");
        }
    }
    else {
        dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>No file is selected.");
    }

    dialogBox->exec();
    refreshWindow();
}

void MainWindow::on_actionFind_Strings_triggered()
{
    ui->MainDisplayStack->setCurrentIndex(1);
    refreshWindow();
}

void MainWindow::on_actionSaved_Strings_triggered()
{
    ui->MainDisplayStack->setCurrentIndex(2);
```

```
    refreshWindow();
}

void MainWindow::on_actionDLL_s_triggered()
{
    ui->MainDisplayStack->setCurrentIndex(3);
    refreshWindow();
}

void MainWindow::on_actionHex_triggered()
{
    ui->MainDisplayStack->setCurrentIndex(4);
    refreshWindow();
}

void MainWindow::on_actionEntropy_Graph_triggered()
{
    ui->MainDisplayStack->setCurrentIndex(5);
    buildEntropyGraph();
    refreshWindow();
}

void MainWindow::on_actionDisassembly_triggered()
{
```

```
    ui->MainDisplayStack->setCurrentIndex(6);
    refreshWindow();
}
```

```
void MainWindow::on_actionChecklistMain_triggered()
{
    if (!checklistOpened) {
        this->setMaximumWidth(1500);
        this->setFixedWidth(1500);
        checklistOpened = true;
    }
    else {
        // weird issue where making smaller once doesnt work until window is physically moved on
screen
        this->setMaximumWidth(984);
        this->setFixedWidth(984);
        this->setMaximumWidth(985);
        this->setFixedWidth(985);
        checklistOpened = false;
    }
}
```

```
void MainWindow::buildChecklist()
{
    QList<int> sizes;
    QFile file("checklist/file.html");
    if (file.open(QIODevice::ReadOnly)) {
     QTextStream ts(&file);
     ui->checklistFileBrowser->setHtml(ts.readAll());
     sizes += 450;
     sizes += 270;
     ui->fileSplitter->setSizes(sizes);
     file.close();
    }
    file.setFileName("checklist/packers.html");
    if (file.open(QIODevice::ReadOnly)) {
     QTextStream ts(&file);
     ui->checklistPackersBrowser->setHtml(ts.readAll());
     sizes.clear();
     sizes += 450;
     sizes += 270;
     ui->packersSplitter->setSizes(sizes);
     file.close();
    }
    file.setFileName("checklist/strings.html");
```

```
if (file.open(QIODevice::ReadOnly)) {
 QTextStream ts(&file);
 ui->checklistStringsBrowser->setHtml(ts.readAll());
 sizes.clear();
 sizes += 365;
 sizes += 360;
 ui->stringsSplitter->setSizes(sizes);
 file.close();
}
file.setFileName("checklist/dlls.html");
if (file.open(QIODevice::ReadOnly)) {
 QTextStream ts(&file);
 ui->checklistDLLsBrowser->setHtml(ts.readAll());
 sizes.clear();
 sizes += 550;
 sizes += 175;
 ui->DLLsSplitter->setSizes(sizes);
 file.close();
}
file.setFileName("checklist/hex.html");
if (file.open(QIODevice::ReadOnly)) {
 QTextStream ts(&file);
 ui->checklistHexBrowser->setHtml(ts.readAll());
```

```
    sizes.clear();
    sizes += 465;
    sizes += 260;
    ui->hexSplitter->setSizes(sizes);
    file.close();
   }
   file.setFileName("checklist/disassembly.html");
   if (file.open(QIODevice::ReadOnly)) {
    QTextStream ts(&file);
    ui->checklistDisassemblyBrowser->setHtml(ts.readAll());
    sizes.clear();
    sizes += 465;
    sizes += 260;
    ui->disassemblySplitter->setSizes(sizes);
    file.close();
   }
   file.setFileName("checklist/introduction.html");
   if (file.open(QIODevice::ReadOnly)) {
    QTextStream ts(&file);
    ui->introductionBrowser->setHtml(ts.readAll());
    file.close();
   }
}
```

```
void MainWindow::on_stringsScrollBar_valueChanged()
{
    if (!reseting) {
        refreshStrings();
    }
}

void MainWindow::on_hexScrollBar_valueChanged()
{
    if (!reseting) {
        refreshHex();
    }
}

void MainWindow::on_disassemblyScrollBar_valueChanged()
{
    if (!reseting) {
        refreshDisassembly();
    }
}

void MainWindow::on_stringList_itemDoubleClicked(QListWidgetItem *item)
{
```

```
    // check item
    bool itemIndexFound = false;
    int i = 0;
    while (!itemIndexFound && i < maxDisplayStrings) {
        if (ui->stringList->item(i) == item) {
            itemIndexFound = true;
            if (ui->stringList->item(i)->checkState()) {
                ui->stringList->item(i)->setCheckState(Qt::CheckState::Unchecked);
            }
            else {
                ui->stringList->item(i)->setCheckState(Qt::CheckState::Checked);
            }
        }
        else {
            i++;
        }
    }
}
```

```
void MainWindow::on_stringSearchButton_clicked()
{
    // set searching icon (searching text in page value box)
    ui->stringsPageNumberValue->setText("SEARCHING");
```

```
    qApp->processEvents();
    // if found
    if (searchStringList(ui->searchString->text(), &strings, ui->stringsSearchFromBeginningCheckBox->checkState(), false)) {
        int previousPage = ui->stringsScrollBar->value(), displayStrings = ui->stringList->count();
        if (searchStringIndex % displayStrings == 0) {
            ui->stringsScrollBar->setValue((searchStringIndex / displayStrings) - 1);
        }
        else {
            ui->stringsScrollBar->setValue(searchStringIndex / displayStrings);
        }
        // unselect any selected then select search string
        if (previousPage == ui->stringsScrollBar->value()) {
            for (int i = 0; i < displayStrings; i++) {
                ui->stringList->item(i)->setSelected(false);
            }
        }
        refreshWindow();
        ui->stringList->item((searchStringIndex % displayStrings - 1 + displayStrings) % displayStrings)->setSelected(true);
    }
    else {
        refreshWindow();
```

```
    }
}

void MainWindow::on_savedStringSearchButton_clicked()
{
    // set searching icon (searching text in page value box)
    ui->savedStringCountValue->setText("SEARCHING");
    qApp->processEvents();
    // if found
    if (searchStringList(ui->searchSavedString->text(), &savedStrings, ui->savedStringsSearchFromBeginningCheckBox->checkState(), false)) {
        int displayStrings = ui->savedStringList->count();
        for (int i = 0; i < displayStrings; i++) {
            ui->savedStringList->item(i)->setSelected(false);
        }
        refreshWindow();
        ui->savedStringList->item(searchStringIndex - 1)->setSelected(true);
        ui->savedStringList->setCurrentRow(searchStringIndex - 1);
    }
    else {
        refreshWindow();
    }
}
```

```
void MainWindow::on_disassemblySearchButton_clicked()
{
    // set searching icon (red searching text in search box)
    ui->disassemblyPageNumberValue->setText("SEARCHING");
    qApp->processEvents();
    // if found, set scroll bar value
    if (searchStringList(ui->disassemblySearchString->text(), &disassembly, ui->disassemblySearchFromBeginningCheckBox->checkState(), true)) {
        refreshWindow();
        ui->disassemblyScrollBar->setValue(searchStringIndex - 1);
    }
    else {
        refreshWindow();
    }
}

bool MainWindow::searchStringList(QString searchString, QStringList *list, bool searchFromBeginning, bool htmlList)
{
    int count = list->size();
    if (searchStringIndex >= count || searchFromBeginning) {
        searchStringIndex = 0;
    }
```

```
previousSearchString = searchString;


// for each string in list
int searchStartIndex = searchStringIndex + 1;
bool found = false;
while (!found && searchStartIndex != searchStringIndex) {
    QString currentItem = list->at(searchStartIndex);
    searchString = htmlSanitiseString(searchString);
    int searchLength = searchString.length(), currentLength = currentItem.length();


    // for each starting position the search string could fit into the searched string
    bool tag = false;
    int i = 0;
    while (i <= currentLength - searchLength && !found) {
        // if html, dont search through tags
        if (htmlList) {
            if (currentItem[i] == '<') {
                tag = true;
            }
            else if (currentItem[i] == '>') {
                tag = false;
            }
        }
```

```
    if (!tag) {
        // for the length of the search string
        int j;
        for (j = 0; j < searchLength; j++) {
            // if chars dont match
            if (currentItem[i + j] != searchString[j]) {
                // if searching char is A - Z, check a - z aswell
                if (currentItem[i + j].unicode() >= 65 && currentItem[i + j].unicode() <= 90) {
                    if ((currentItem[i + j].unicode() + 32) != searchString[j]) {
                        break;
                    }
                }
                else {
                    break;
                }
            }
        }
        if (j == searchLength) {
            found = true;
        }
    }
    i++;
}
```

```
        if (searchStartIndex >= count - 1) {
            searchStartIndex = 0;
        }
        else {
            searchStartIndex++;
        }
    }

    searchStringIndex = searchStartIndex;

    if (found) {
        return true;
    }
    return false;
}
```

```
void MainWindow::sortStrings()
{
    refreshWindow();
    sorting = true;
    if (stringsSorted) {
        strings = unsortedStrings;
        stringsSorted = false;
```

```
}
else { // sort
    // append string location onto string
    for (int i = 0; i < stringCount; i++) {
        strings[i] += "(" + QString::number(i);
    }
    strings.sort();
    // remove string location and keep track of old location
    for (int i = 0; i < stringCount; i++) {
        QString string = strings[i];
        int startOfOldLoc = string.lastIndexOf("(", string.size() - 1);
        int oldLoc = string.mid(startOfOldLoc + 1, string.size() - 1).toInt();
        strings[i] = string.mid(0, startOfOldLoc);
        swapStringMap[oldLoc] = i;
    }
    stringsSorted = true;
}


// swap values in saved strings map
QMap<int, bool> tmpSavedSwapMap;
if (stringsSorted) {
    for (int i = 0; i < stringCount; i++) {
        tmpSavedSwapMap[swapStringMap[i]] = savedStringMap[i];
```

```
        }
    }
    else {
        for (int i = 0; i < stringCount; i++) {
            tmpSavedSwapMap[i] = savedStringMap[swapStringMap[i]];
        }
    }
    savedStringMap = tmpSavedSwapMap;
    refreshStrings();
    refreshSavedStrings();
}
```

```
void MainWindow::outputStrings()
{
    dialogBox = new CustomDialog();
    dialogBox->setWindowTitle("Output Strings");

    if (fileOpened) {
        dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>Please select a location to store the strings text file.");
        dialogBox->exec();
        QFile file(QFileDialog::getExistingDirectory(this, "Select a location to store strings."));
        QString stringsLoc = file.fileName();
```

```
file.close();


if (stringsLoc != "") {
    // find which stringlist to use and the total size of the strings
    int totalStringsLength = 0, stringsSize = 0;
    QStringList list;
    QString outputFileName = "";
    if (ui->MainDisplayStack->currentIndex() == 1) {
        list = strings;
        outputFileName = "strings." + fileName + ".txt";
        stringsSize = stringCount;
        totalStringsLength = totalStringSize + stringsSize;
    }
    else {
        list = savedStrings;
        outputFileName = "savedStrings." + fileName + ".txt";
        stringsSize = savedStrings.count();
        totalStringsLength = totalSavedStringSize + stringsSize;
    }

    QString fullName = stringsLoc + 92 + outputFileName;
    QFile stringsOutput(fullName);
    char *stringBytes;
```

```
if (stringsOutput.open(QIODevice::ReadWrite)) {

    bool enoughMemory = true;
    try {
        stringBytes = new char[totalStringsLength];
    } catch (...) {
        enoughMemory = false;
    }

    if (enoughMemory) {
        // turn stringlist into char array
        int byte = 0;
        // for each string
        for (int i = 0; i < stringsSize; i++) {
            QString string = list.at(i);
            int size = string.size();
            // for each char in string
            for (int j = 0; j < size; j++) {
                stringBytes[byte] = string.at(j).unicode();
                byte++;
            }
            // append line break
```

```
                stringBytes[byte] = 10;
                byte++;
            }

            QDataStream ds(&stringsOutput);
            ds.writeRawData(stringBytes, totalStringsLength - 1);
            stringsOutput.close();

            dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>Strings outputted successfully.");
        }
        else {
            dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>Not enough system memory.");
        }

        free(stringBytes);
    }
    else {
        dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>Cannot write to disk.");
    }
}
```

```
        else {
            dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>No location selected.");
        }
    }
    else {
        dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>You must first select a file to analyse before strings can be outputted to a text file.");
    }
    dialogBox->exec();
}

void MainWindow::wheelEvent(QWheelEvent *event)
{
    QPoint numDegrees = event->angleDelta() / 8;
    if (!numDegrees.isNull()) {
        QPoint numSteps = numDegrees / 15;
        int move = static_cast<int>(numSteps.y());
        int currentValue = 0, max = 0;
        QPointer<QScrollBar> scrollBar;

        // if displaying strings
        if (ui->MainDisplayStack->currentIndex() == 1) {
```

```
        currentValue = ui->stringsScrollBar->value();
        max = ui->stringsScrollBar->maximum();
        scrollBar = ui->stringsScrollBar;
    }
    // if displaying hex
    if (ui->MainDisplayStack->currentIndex() == 4) {
        currentValue = ui->hexScrollBar->value();
        max = ui->hexScrollBar->maximum();
        scrollBar = ui->hexScrollBar;
    }
    // if displaying disassembly
    if (ui->MainDisplayStack->currentIndex() == 6) {
        currentValue = ui->disassemblyScrollBar->value();
        max = ui->disassemblyScrollBar->maximum();
        scrollBar = ui->disassemblyScrollBar;
    }

    // if scrollbar should move
    if ((move > 0 && currentValue > 0) || (move < 0 && currentValue < max)) {
        scrollBar->setValue(currentValue - move);
    }
}
event->accept();
```

```
    refreshHex();
}

void MainWindow::closeEvent(QCloseEvent *event)
{
    saveChanges();
    event->accept();
}

// BACKGROUND FUNCTIONS

void MainWindow::open(QFile *file)
{
    if (file->fileName() != "") {

        if (file->size() <= 2147483647) {

            bool openFile = true;
            if (file->size() > 1048576) {
                QMessageBox::StandardButton reply;
                reply = QMessageBox::question(this, "Warning!", "The file you choose is over 1MB.\nIt will be noticiably slower for most of the functions and things may not work as intended.\nAre you sure you want to continue?", QMessageBox::Yes|QMessageBox::No);
                if (reply == QMessageBox::Yes) {
```

```
        openFile = true;
    }
    else {
        openFile = false;
    }
}

if (file->open(QIODevice::ReadOnly) && openFile) {

    if (fileOpened) {
        free(rawData);
    }

    QDataStream ds(file);
    fileSize = file->size();

    bool enoughMemory = true;
    try {
        rawData = new char[fileSize];
    } catch (...) {
        enoughMemory = false;
    }
```

```
if (enoughMemory) {
    ds.readRawData(rawData, fileSize);
    fileOpened = true;
    resetChecks();
    getPEinformation();
    setHexValues();

    // get file name and directory
    QString fullFileName = file->fileName();
    bool slashFound = false;
    int i = fullFileName.size() - 1;

    while (!slashFound && i >= 0) {
        if (fullFileName[i] == 47 || fullFileName[i] == 92) {
            slashFound = true;
        }
        else {
            i--;
        }
    }

    if (slashFound) {
        fileName = fullFileName.mid(i + 1, fullFileName.size() - i);
```

```
                directory = fullFileName.mid(0, i + 1);
            }
            basicWindowName = "Static File Analysis Tool - " + fileName;
        }
        else {
            // display not enough memory
            dialogBox = new CustomDialog();
            dialogBox->setWindowTitle("Error");
            dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>The file you choose is too big.<br>This system does not currently have enough memory to open this file.");
            dialogBox->exec();
        }
    }
    refreshWindow();
}
else {
    // file too big
    dialogBox = new CustomDialog();
    dialogBox->setWindowTitle("Error");
    dialogBox->setText("<br><p style='text-align:center;font-size:25px;'>The file you choose is too big.<br>Max file size of 2GB.");
    dialogBox->exec();
}
```

```
    }
}
```

QString MainWindow::generateHash(char *data , int size)

```
{
    QCryptographicHash hash(QCryptographicHash::Sha1);
    hash.addData(data, size);
    return hash.result().toHex().toUpper();
}
```

QString MainWindow::generateFileHash(QString fullName)

```
{
    QFile file(fullName);
    QString hashString;
    if (file.open(QIODevice::ReadOnly)) {
        QCryptographicHash hash(QCryptographicHash::Sha1);
        hash.addData(&file);
        hashString = hash.result().toHex().toUpper();
        file.close();
    }
    return hashString;
}
```

```
bool MainWindow::isPacked()
{
    if (!packChecked) {
        QString fullFileName = 34 + directory + fileName + 34;
        QString command = "upx -l " + fullFileName + " > upx.tmp";
        system(qPrintable(command));

        // check output
        QFile upxCheck("upx.tmp");
        if (upxCheck.open(QIODevice::ReadOnly)) {
            QTextStream in(&upxCheck);
            int i = 0;
            while (!in.atEnd()) {
                in.readLine();
                i++;
            }
            upxCheck.close();
            if (i == 7) {
                packed = true;
            }
            else {
                packed = false;
            }
```

```
        }
        // remove tmp file
        QDir path;
        path.setPath(path.currentPath());
        path.remove("upx.tmp");
        packChecked = true;
    }
    return packed;
}
```

bool MainWindow::pack()

```
{
    QMessageBox::StandardButton reply;
    reply = QMessageBox::question(this, "Warning!", "Make sure to backup the file before packing!\nAre you sure you want to pack the current file?", QMessageBox::Yes|QMessageBox::No);
    if (reply == QMessageBox::Yes) {
        QString fullFileName = 34 + directory + fileName + 34;
        QString command = "upx -5 " + fullFileName;
        system(qPrintable(command));
        return true;
    }
    return false;
```

```
}
```

bool MainWindow::unpack()

```
{
    QMessageBox::StandardButton reply;
    reply = QMessageBox::question(this, "Warning!", "Make sure to backup the file before unpacking!\nAre you sure you want to unpack the current file?", QMessageBox::Yes|QMessageBox::No);
    if (reply == QMessageBox::Yes) {
        QString fullFileName = 34 + directory + fileName + 34;
        QString command = "upx -d " + fullFileName;
        system(qPrintable(command));
        return true;
    }
    return false;
}
```

void MainWindow::findStrings()

```
{
    if (!stringsBuilt && fileOpened) {

        // strings context menu
        ui->stringList->setContextMenuPolicy(Qt::CustomContextMenu);
        ui->savedStringList->setContextMenuPolicy(Qt::CustomContextMenu);
```

```
    connect(ui->stringList, SIGNAL(customContextMenuRequested(QPoint)), this,
SLOT(showContextMenu(QPoint)));
    connect(ui->savedStringList, SIGNAL(customContextMenuRequested(QPoint)), this,
SLOT(showContextMenu(QPoint)));

    QString item = "";
    bool nullSpaced = false, validString = false;
    stringCount = 0;

    // for each char in file
    int stringStartLoc = 0;
    for (int i = 0; i < fileSize; i++) {

        if (item == "") {
            stringStartLoc = i;
        }

        // get char
        char c = rawData[i];
        unsigned char uc = static_cast<unsigned char>(c);

        // check if (unsigned) char is printable
        // if char is between 'space' and '~' on the ascii table or char is horizontal tab
```

```
        if ((uc >= 32 && uc <= 126)) {
            item += c;
        }
        else {
            // if item has value and current char is null and last value is not null
            if (uc == 0) {
                // check if this may be a null spaced string
                if (item.size() == 1 && i + 3 < fileSize) {
                    char tmpc1 = rawData[i + 1], tmpc2 = rawData[i + 2], tmpc3 = rawData[i + 3];
                    unsigned char tmpuc1 = static_cast<unsigned char>(tmpc1);
                    unsigned char tmpuc2 = static_cast<unsigned char>(tmpc2);
                    unsigned char tmpuc3 = static_cast<unsigned char>(tmpc3);
                    if ((tmpuc1 >= 32 && tmpuc1 <= 126) && (tmpuc2 == 0) && (tmpuc3 >= 32
&& tmpuc3 <= 126)) {
                        nullSpaced = true;
                    }
                    else {
                        item = "";
                    }
                }
                // if nullspaced, check if next char is null and if next two chars are printable
                else if (nullSpaced) {
                    if (i + 2 < fileSize) {
```

```
                    char tmpc1 = rawData[i + 1], tmpc2 = rawData[i + 2];
                    unsigned char tmpuc1 = static_cast<unsigned char>(tmpc1), tmpuc2 =
static_cast<unsigned char>(tmpc2);
                    if (tmpuc1 == 0 || ((tmpuc1 >= 32 && tmpuc1 <= 126) && (tmpuc2 >= 32 &&
tmpuc2 <= 126))) {
                        nullSpaced = false;
                        if (item.size() >= stringLength) {
                            validString = true;
                        }
                    }
                }
            }
            else if (item.size() >= stringLength) {
                validString = true;
            }
            else {
                item = "";
            }
        }
        else if (item.size() >= stringLength) {
            validString = true;
        }
        else {
```

```
            item = "";
        }
    }

    // if current item should be added to string list
    if (validString) {
        strings.insert(stringCount,item);
        stringLocationMap[i - item.size()] = item;
        hexLocationMap[stringCount] = stringStartLoc;
        totalStringSize += item.size();
        stringCount++;
        item = "";
        validString = false;
    }
}

// if string exists that is terminated by the file ending
if (item != "" && item.size() >= stringLength) {
    strings.insert(stringCount,item);
    stringLocationMap[fileSize - item.size()] = item;
    hexLocationMap[stringCount] = fileSize - item.size();
    totalStringSize += item.size();
    stringCount++;
```

```
}

for (int i = 0; i < stringCount; i++) {
    swapStringMap[i] = i;
}

unsortedStrings = strings;

// display things
stringCount = strings.size();
QString count = QString::number(stringCount);
ui->stringCountValue->setText(count);

// scroll bar value
if (stringCount % maxDisplayStrings > 0) {
    ui->stringsScrollBar->setMaximum(stringCount / maxDisplayStrings);
}
else {
    if (stringCount == 0) {
        ui->stringsScrollBar->setMaximum(0);
    }
    else {
        ui->stringsScrollBar->setMaximum(stringCount / maxDisplayStrings - 1);
```

```
            }
        }
        stringsBuilt = true;
        ui->stringCountValue->setText(QString::number(stringCount));
    }
}

void MainWindow::refreshStrings()
{
    if (fileOpened) {
        saveDisplayedStrings();
        ui->stringList->clear();

        int displayStringCount = maxDisplayStrings;
        stringOffset = ui->stringsScrollBar->value() * maxDisplayStrings;

        if (ui->stringsScrollBar->value() == ui->stringsScrollBar->maximum()) {
            if (stringCount == 0) {
                displayStringCount = 0;
            }
            else if (stringCount % maxDisplayStrings > 0) {
                displayStringCount = stringCount % maxDisplayStrings;
            }
```

```
}

//build checkboxlist
QStringList displayStrings;
for (int i = 0; i < displayStringCount; i++) {
   displayStrings.insert(i, strings[stringOffset + i]);
}

ui->stringList->addItems(displayStrings);

QSize stringSize;
stringSize.setHeight(ui->stringList->height() / maxDisplayStrings);
// recheck saved strings
for(int i = 0; i < displayStringCount; i++) {
   if (savedStringMap[i + stringOffset]) {
      ui->stringList->item(i)->setCheckState(Qt::Checked);
   }
   else {
      ui->stringList->item(i)->setCheckState(Qt::Unchecked);
   }
   ui->stringList->item(i)->setSizeHint(stringSize);
}
```

```
        // display things
        QString currentPage = QString::number(ui->stringsScrollBar->value() + 1);
        QString maxPage = QString::number(ui->stringsScrollBar->maximum() + 1);
        QString display;
        display.append(currentPage);
        display.append(" / ");
        display.append(maxPage);
        ui->stringsPageNumberValue->setText(display);
    }
}

void MainWindow::saveDisplayedStrings()
{
    if (!firstStringsRefresh) {
        if (!sorting && !removedStrings) {
            if (ui->stringList->count() > 0) {
                for (int i = 0; i < ui->stringList->count(); i++) {
                    if (ui->stringList->item(i)->checkState()) {
                        savedStringMap[stringOffset + i] = true;
                    }
                    else {
                        savedStringMap[stringOffset + i] = false;
                    }
```

```
            }
            firstStringsRefresh = false;
        }
    }
    else {
        sorting = false;
        removedStrings = false;
    }
  }
  else {
    firstStringsRefresh = false;
  }
}
```

```
void MainWindow::refreshSavedStrings()
{
  if (stringsBuilt) {
    saveDisplayedStrings();
    savedStrings.clear();
    savedStringLocationMap.clear();
    totalSavedStringSize = 0;
    for (int i = 0; i < stringCount; i++) {
      if (savedStringMap[i]) {
```

```
            savedStrings += strings[i];
            savedStringLocationMap[savedStrings.count() - 1] = i;
            totalSavedStringSize += strings[i].size();
        }
    }
    ui->savedStringList->clear();
    ui->savedStringList->addItems(savedStrings);

    // display things
    QString count = QString::number(ui->savedStringList->count());
    ui->savedStringCountValue->setText(count);
  }
}

void MainWindow::showContextMenu(const QPoint &point)
{
  QListWidget *list;
  QMenu myMenu;
  myMenu.addAction("Copy",  this, SLOT(copyHighlightedItemsText()));

  // if find strings page
  if (ui->MainDisplayStack->currentIndex() == 1) {
    list = ui->stringList;
```

```
    myMenu.addAction("Check",  this, SLOT(checkHighlighted()));
    myMenu.addAction("Uncheck",  this, SLOT(uncheckHighlighted()));
}
else {
    list = ui->savedStringList;
    myMenu.addAction("Remove",  this, SLOT(removeSelected()));
}

myMenu.addSeparator();
myMenu.addAction("Highlight All",  this, SLOT(highlightAll()));
myMenu.addAction("Sort / Unsort Strings",  this, SLOT(sortStrings()));
myMenu.addSeparator();
myMenu.addAction("View Location in Hex", this, SLOT(stringToHexLocation()));

// if find strings page
if (ui->MainDisplayStack->currentIndex() == 1) {
    myMenu.addAction("Output all strings to txt file", this, SLOT(outputStrings()));
}
else {
    myMenu.addAction("Output saved strings to txt file", this, SLOT(outputStrings()));
}

// set location for men to appear
```

```
    QPoint globalPos = list->mapToGlobal(point);


    myMenu.exec(globalPos);
}


void MainWindow::copyHighlightedItemsText()
{
    // copy all selected text
    QString text = "";
    if (ui->MainDisplayStack->currentIndex() == 1) {
        for (int i = 0; i < maxDisplayStrings; i++) {
            if (ui->stringList->item(i)->isSelected()) {
                text += ui->stringList->item(i)->text() + "\n";
            }
        }
    }
    else {
        for (int i = 0; i < ui->savedStringList->count(); i++) {
            if (ui->savedStringList->item(i)->isSelected()) {
                text += ui->savedStringList->item(i)->text() + "\n";
            }
        }
    }
```

```
    // \n counts as 1 char
    text.remove(text.size() - 1, 1);
    QApplication::clipboard()->setText(text);
}

void MainWindow::checkHighlighted()
{
    for (int i = 0; i < maxDisplayStrings; i++) {
        if (ui->stringList->item(i)->isSelected()) {
            ui->stringList->item(i)->setCheckState(Qt::CheckState::Checked);
        }
    }
}

void MainWindow::uncheckHighlighted()
{
    for (int i = 0; i < maxDisplayStrings; i++) {
        if (ui->stringList->item(i)->isSelected()) {
            ui->stringList->item(i)->setCheckState(Qt::CheckState::Unchecked);
        }
    }
}
```

```
void MainWindow::highlightAll()
{
    if (ui->MainDisplayStack->currentIndex() == 1) {
        for (int i = 0; i < maxDisplayStrings; i++) {
            ui->stringList->item(i)->setSelected(true);
        }
    }
    else {
        for (int i = 0; i < ui->savedStringList->count(); i++) {
            ui->savedStringList->item(i)->setSelected(true);
        }
    }
}
```

```
void MainWindow::removeSelected()
{
    if (fileOpened) {
        QModelIndexList selection = ui->savedStringList->selectionModel()->selectedRows();
        for (int i = 0; i < selection.count(); i++) {
            QModelIndex index = selection.at(i);
            savedStringMap[savedStringLocationMap[index.row()]] = false;
        }
        removedStrings = true;
```

```
        refreshStrings();
        refreshSavedStrings();
    }
}
```

void MainWindow::stringToHexLocation()

```
{
    bool itemIndexFound = false;
    int i = 0; // string location on screen
    int scrollBarValue = 0;

    // if from find strings
    if (ui->MainDisplayStack->currentIndex() == 1) {
        while (!itemIndexFound && i < maxDisplayStrings) {
            if (ui->stringList->item(i)->isSelected()) {
                itemIndexFound = true;
            }
            else {
                i++;
            }
        }
        if (stringsSorted) {
            bool locFound = false;
```

```
        int j = 0;
        while (!locFound && j < stringCount) {
            if (swapStringMap[j] == stringOffset + i) {
                locFound = true;
            }
            else {
                j++;
            }
        }
        scrollBarValue = hexLocationMap[j] / hexDisplayCols;
    }
    else {
        scrollBarValue = hexLocationMap[stringOffset + i] / hexDisplayCols;
    }
}
// if from saved strings
else if (ui->MainDisplayStack->currentIndex() == 2) {
    while (!itemIndexFound && i < ui->savedStringList->count()) {
        if (ui->savedStringList->item(i)->isSelected()) {
            itemIndexFound = true;
        }
        else {
            i++;
```

```
        }
    }
    if (stringsSorted) {
        bool locFound = false;
        int j = 0;
        while (!locFound && j < stringCount) {
            if (swapStringMap[j] == savedStringLocationMap[i]) {
                locFound = true;
            }
            else {
                j++;
            }
        }
        scrollBarValue = hexLocationMap[j] / hexDisplayCols;
    }
    else {
        scrollBarValue = hexLocationMap[savedStringLocationMap[i]] / hexDisplayCols;
    }
}

ui->MainDisplayStack->setCurrentIndex(4);
refreshWindow();
ui->hexScrollBar->setValue(scrollBarValue);
```

```
}

void MainWindow::findDLLs()
{
    if (fileOpened && !dllsBuilt) {

        // font things
        QFont dllFont, titleFont;
        dllFont.setPixelSize(20);
        dllFont.setFamily("Courier");
        titleFont.setPixelSize(25);
        titleFont.setFamily("Courier");
        ui->DLLTitleBrowser->setFont(titleFont);
        ui->DLLFunctionTitleBrowser->setFont(titleFont);
        ui->DLLNameBrowser->setFont(dllFont);
        ui->DLLFunctionNameBrowser->setFont(dllFont);

        // background colour
        ui->DLLTitleBrowser->setStyleSheet("background-color: #FAFAD2;");
        ui->DLLNameBrowser->setStyleSheet("background-color: #FFF8DC;");
        ui->DLLFunctionTitleBrowser->setStyleSheet("background-color: #FAFAD2;");
        ui->DLLFunctionNameBrowser->setStyleSheet("background-color: #FFF8DC;");
```

```
if (PE && (rdataStartLoc > 0 || idataStartLoc > 0)) {

    int dllNamePointerLoc = 0, functionNamePointerLoc = 0, dataSectionRVA = 0,
dataStartLoc = 0, dllCount = 0, functionCount = 0;

    // if Import Directory Table is found
    if (IDTLoc > 0) {
       dllNamePointerLoc = IDTLoc + 12;
    }
    else {
       // if idata section exists
       if (idataStartLoc > 0) {
          dllNamePointerLoc = idataStartLoc;
       }
       else {
          dllNamePointerLoc = rdataStartLoc;
       }
    }

    // if Import Address Table is found
    if (IATLoc > 0) {
       functionNamePointerLoc = IATLoc;
    }
```

```
else {
    // if Import Directory Table is found
    if (IDTLoc > 0) {
        functionNamePointerLoc = IDTLoc + IDTSize;
    }
}

// set current section params
if (idataStartLoc > 0) {
    dataSectionRVA = idataRVA;
    dataStartLoc = idataStartLoc;
}
else {
    dataSectionRVA = rdataRVA;
    dataStartLoc = rdataStartLoc;
}



//
// FIND ALL DLL NAMES
//

if (dllNamePointerLoc > 0) {
```

```
            bool goodName = true;
            int addressOffset = 0;
            while (goodName) {
                int location = 0;
                for (int i = 0; i < 4; i++) {
                    location += pow(16, i * 2) * static_cast<unsigned
char>(rawData[dllNamePointerLoc + i + addressOffset]);
                }
                if (location != 0) {
                    dllNames += getFunctionName(location - 2, dataSectionRVA, dataStartLoc) +
"<br>";
                    dllCount++;
                }
                else {
                    goodName = false;
                }
                addressOffset += 20;
            }
        }

        //
        // FIND ALL FUNCTION NAMES
        //
```

```
        QStringList functions;
        bool functionsFinished = false, secondTry = false;
        int addressOffset = 0, dllsCompletedCount = 0;


        while (!functionsFinished) {
            qint64 location = 0;
            for (int i = 0; i < 4; i++) {
                location += pow(16, i * 2) * static_cast<unsigned
char>(rawData[functionNamePointerLoc + i + addressOffset]);
            }
            // end of a specific dlls functions
            if (location == 0) {
                functions.sort();
                functions += "--------------------------------------------------------------------------------";
                if (dllsCompletedCount == dllCount) {
                    functionsFinished = true;
                }
                else {
                    dllFunctionNames += functions;
                    functions.clear();
                    dllsCompletedCount++;
                }
```

```
        }
        else {
            QString functionName = "";
            // if ordinal function
            if (location > 2147483647) {
                QString ordinal = QString::number(location - 2147483648, 16).toUpper();
                for (int i = ordinal.size(); i < 8; i++) {
                    ordinal.prepend("0");
                }
                functionName = "Ordinal: " + ordinal;
            }
            else {
                functionName = getFunctionName(location, dataSectionRVA, dataStartLoc);
            }
            functions += functionName;
            functionCount++;
        }
        addressOffset += 4;

        // if functions finshed but none found
        if (functionsFinished && dllFunctionNames.size() == dllsCompletedCount &&
!secondTry) {
            // try searching from data start location
```

```
            functionsFinished = false, secondTry = true;
            addressOffset = 0, dllsCompletedCount = 0;
            functionNamePointerLoc = dataStartLoc;
            dllFunctionNames.clear();
            functions.clear();
        }
    }

    // set dll name and funtion title bars
    DLLTitle += "DLL Names (" + QString::number(dllCount) + ") - Import Directory Table (<a href='" + QString::number(dllNamePointerLoc) + "'>" + QString::number(dllNamePointerLoc, 16).toUpper() + "h</a>)";
    FunctionTitle += "Function Names (" + QString::number(functionCount) + ") - Import Address Table (<a href='" + QString::number(functionNamePointerLoc) + "'>" + QString::number(functionNamePointerLoc, 16).toUpper() + "h</a>)";
    ui->DLLTitleBrowser->insertHtml(DLLTitle);
    ui->DLLFunctionTitleBrowser->insertHtml(FunctionTitle);

    if (dllNames.count() == 0) {
        dllNames += "There was an error when finding dll names and function calls.";
    }
    else {
        dllNames.remove(dllNames.size() - 4, 4);
```

```
            dllFunctionNames.prepend("--------------------------------------------------------------------------------");
        }
    }
    else {
        if (!PE) {
            dllNames += "This file is not a PE file.\nDLL names and function calls could not be found.";
        }
        else {
            dllNames += "This file is a PE file.\nHowever, idata or rdata section could not be found.";
        }
    }

    QString html;
    for (int i = 0; i < dllFunctionNames.size(); i++) {
        html += dllFunctionNames.at(i) + "<br>";
    }

    // set dll names and function
    ui->DLLNameBrowser->insertHtml(dllNames);
    ui->DLLFunctionNameBrowser->insertHtml(html);
```

```
        dllsBuilt = true;
    }
    else if (dllsBuilt) {
        ui->DLLTitleBrowser->clear();
        ui->DLLFunctionTitleBrowser->clear();
        ui->DLLTitleBrowser->insertHtml(DLLTitle);
        ui->DLLFunctionTitleBrowser->insertHtml(FunctionTitle);
    }
}

void MainWindow::on_DLLTitleBrowser_anchorClicked(const QUrl &arg1)
{
    ui->MainDisplayStack->setCurrentIndex(4);
    ui->hexScrollBar->setValue(arg1.url().toInt() / hexDisplayCols);
    refreshWindow();
}

void MainWindow::on_DLLFunctionTitleBrowser_anchorClicked(const QUrl &arg1)
{
    ui->MainDisplayStack->setCurrentIndex(4);
    ui->hexScrollBar->setValue(arg1.url().toInt() / hexDisplayCols);
    refreshWindow();
```

```
}
```

QString MainWindow::getFunctionName(int location, int dataSectionRVA, int dataStartLoc)

```
{
    bool terminator = false;
    int i = 0;
    QString functionName = "";
    int loc = location - dataSectionRVA + dataStartLoc + 2;
    if (loc > 0) {
        while (!terminator && loc + i < fileSize) {
            char c = rawData[loc + i];
            if (static_cast<unsigned char>(c) == 0 && functionName.size() > 0) {
                terminator = true;
            }
            else if (static_cast<unsigned char>(c) != 0) {
                functionName += c;
            }
            i++;
        }
        if (loc + i < fileSize) {
            return functionName;
        }
        return "";
```

```
    }
    return "";
}
```

QString MainWindow::byteToHexString(int c)

```
{
    int temp, i = 1;
    QString hexText = "00";
    while(c != 0) {
        temp = c % 16;
        if(temp < 10)
        {
            temp += 48;
        }
        else
        {
            temp += 55;
        }
        hexText[i] = temp;
        i--;
        c = c / 16;
    }
    return hexText;
```

```
}

void MainWindow::refreshWindow()
{
    saveChanges();

    // refresh window based on current page
    switch (ui->MainDisplayStack->currentIndex()) {

        case 0: extendedWindowName = "";
        break;

        case 1: extendedWindowName = " - Strings";
        findStrings();
        refreshStrings();
        break;

        case 2: extendedWindowName = " - Saved Strings";
        saveDisplayedStrings();
        refreshSavedStrings();
        break;

        case 3: extendedWindowName = " - DLLs";
```

```
        findDLLs();
        break;

        case 4: extendedWindowName = " - Hex";
        refreshHex();
        break;

        case 5: extendedWindowName = " - Entropy Graph";
        buildEntropyGraph();
        break;

        case 6: extendedWindowName = " - Disassembly";
        refreshDisassembly();
        break;
    }

    // window name
    this->setWindowTitle(basicWindowName + extendedWindowName);
}

void MainWindow::resetChecks()
{
    if (!savingEdit) {
```

```
    reseting = true;

    dataChanged = false;

    // checklist notepads
    ui->fileNotes->clear();
    ui->packersNotes->clear();
    ui->stringsNotes->clear();
    ui->DLLsNotes->clear();
    ui->hexNotes->clear();
    ui->disassemblyNotes->clear();

    reseting = false;
}
packChecked = false;
dllsBuilt = false;
packed = false;

// strings
stringLocationMap.clear();
savedStringMap.clear();
savedStringLocationMap.clear();
swapStringMap.clear();
```

```
strings.clear();
savedStrings.clear();
unsortedStrings.clear();
stringCount = 0;
stringOffset = 0;
maxDisplayStrings = 30;
totalStringSize = 0;
totalSavedStringSize = 0;
stringLength = 3;
sorting = false;
removedStrings = false;
stringsBuilt = false;
stringsSorted = false;
firstStringsRefresh = true;
ui->stringList->clear();
ui->savedStringList->clear();

// dlls
dllNames.clear();
dllFunctionNames.clear();
ui->DLLNameBrowser->clear();
ui->DLLFunctionNameBrowser->clear();
ui->DLLTitleBrowser->clear();
```

```
ui->DLLFunctionTitleBrowser->clear();
DLLTitle = "";
FunctionTitle = "";

// hex
hexLocationMap.clear();
originalDataMap.clear();
changedDataMap.clear();
previousPosition = 0;
byteDisplaySize = 0;
editing = false;
refreshing = false;
nextPage = false;
secondTime = false;
cursorLocation = 0;

// entropy
entropyGraphBuilt = false;

// disassembly
disassemblyBuilt = false;
disassembly.clear();
opTypeMap.clear();
```

```
    opcodeMap.clear();
    locOffsetMap.clear();
    codeStartProcedure = 0;
    jumpStack.clear();
}

void MainWindow::undoChanges()
{
    if (dataChanged) {
        for (int i = 0; i < fileSize; i++) {
            if (changedDataMap[i]) {
                rawData[i] = originalDataMap[i];
            }
        }
        dataChanged = false;
    }
}

void MainWindow::saveChanges()
{
    if (dataChanged && !savingEdit) {
        QMessageBox::StandardButton reply;
```

```
reply = QMessageBox::question(this, "Warning!", "Do you want to save the changes made in the hex editor?\nThis will reset all saved strings.", QMessageBox::Yes|QMessageBox::No);
if (reply == QMessageBox::Yes) {

    // change actual file data
    QString tmpHash = generateHash(rawData, fileSize);
    QString tmpName = tmpHash + ".tmp";
    QString fullTmpName = directory + tmpName;

    // write to temporary file
    QFile newFile(fullTmpName);
    if (newFile.open(QIODevice::ReadWrite)) {
        QDataStream ds(&newFile);
        ds.writeRawData(rawData, fileSize);
        newFile.close();
    }

    // create verficication hash
    QString verificationHash = generateFileHash(fullTmpName);

    // verify save
    QDir path(directory);
    if (verificationHash == tmpHash) {
```

```
            // delete original file and rename tmp file
            path.remove(fileName);
            path.rename(tmpName, fileName);

            // open new file
            QFile f(directory + fileName);
            savingEdit = true;
            open(&f);
            savingEdit = false;
            f.close();
        }
        else {
            dialogBox = new CustomDialog();
            dialogBox->setWindowTitle("Error");
            dialogBox->setText("dialogBoxAn error has occured.<br>No changes have been saved.");
            dialogBox->exec();

            path.remove(tmpName);
            undoChanges();
        }
    } else {
        undoChanges();
```

```
        }
        dataChanged = false;
    }
}
```

double MainWindow::getEntropy()

```
{
    double entropy = 0;
    if (fileOpened) {
        entropy = chunkEntropy(0, fileSize);
    }
    return entropy;
}
```

double MainWindow::chunkEntropy(int offset, int chunkSize)

```
{
    double chunkEntropy = 0;
    if (fileOpened) {
        if (offset + chunkSize > fileSize) {
            chunkSize = ((fileSize - (offset + chunkSize)) + chunkSize) % chunkSize;
        }

        QMap<unsigned char, double> freqMap;
```

```
    // set all to 0
    for (int i = 0; i < 256; i++) {
        freqMap[i] = 0;
    }

    // get char freq
    for (int i = 0; i < chunkSize; i++) {
        char c = rawData[i + offset];
        unsigned char uc = static_cast<unsigned char>(c);
        freqMap[uc]++;
    }

    //if (chunkSize < )
    // get probabilty of each char and append to entropy
    for (int i = 0; i < 256; i++) {
        unsigned char c = i;
        double prob = freqMap[c] / chunkSize;
        if (prob > 0) {
            chunkEntropy += (-1 * (prob * (log2(prob))));
        }
    }
}
```

```
    return chunkEntropy;
}


void MainWindow::buildEntropyGraph()
{
    if (fileOpened && fileSize > 0) {
        if (!entropyGraphBuilt) {
            int chunks = 64; // max chunks
            int chunkSize = 256;

            // get good chunk size and no. of chunks(max 64)
            bool goodChunkCount = false, goodChunkSize = false;
            while ((!goodChunkSize || !goodChunkCount) && chunks > 0) {
                if (fileSize / chunks > chunkSize && !goodChunkCount) {
                    chunkSize *= 2;
                }
                else {
                    if (chunks * chunkSize > fileSize && !goodChunkSize) {
                        chunks--;
                    }
                    else {
                        goodChunkSize = true;
                    }
```

```
            goodChunkCount = true;
        }
    }

    QBarSet *entropySet = new QBarSet("Chunk Entropy");
    QStringList chunkRange;

    // for all full chunks
    for (int i = 0; i < chunks; i++) {
        *entropySet << chunkEntropy(i * chunkSize, chunkSize);
        chunkRange << QString::number(i);
    }

    // last and incomplete chunk
    /*
    if (fileSize % chunkSize > 0) {
        *entropySet << chunkEntropy(chunks * chunkSize, (fileSize % chunkSize +
chunkSize) % chunkSize);
        chunkRange << QString::number(chunks); //QString::number(chunks * chunkSize);//
<< " - " << QString::number((chunks * chunkSize) + ((fileSize % chunkSize + chunkSize) %
chunkSize));
    }*/
```

```
QBarCategoryAxis *axisY = new QBarCategoryAxis();
QValueAxis *axisX = new QValueAxis();
QHorizontalBarSeries *series = new QHorizontalBarSeries();

// reverse data for graph output
QBarSet *tmpSet = new QBarSet("");
QStringList tmpList;
for (int i = 0; i < entropySet->count(); i++) {
    *tmpSet << entropySet->at(entropySet->count() - 1 - i);
    tmpList << chunkRange.at(chunkRange.count() - 1 - i);
}
entropySet = tmpSet;
chunkRange = tmpList;

series->append(entropySet);
axisY->append(chunkRange);

QChart *chart = new QChart();
chart->addSeries(series);

QString chunkBytes = QString::number(chunkSize) + " bytes";
if (chunkSize >= 1024) {
    chunkBytes = QString::number(chunkSize / 1024) + " kibibytes";
```

```
            if (chunkSize >= 1048576) {
                chunkBytes = QString::number(chunkSize / 1048576) + " mebibytes";
                if (chunkSize >= 1073741824) {
                    chunkBytes = QString::number(chunkSize / 1073741824) + " gibibytes";
                }
            }
        }
        if (fileSize % chunkSize > 0) {
            /*
            if (chunks == 0) {
                chart->setTitle("Average file entropy across " + QString::number(1) + " chunk of "
+ chunkBytes);
            }
            else {
                chart->setTitle("Average file entropy across " + QString::number(chunks+1) + "
chunks of " + chunkBytes + "  each, with the last being the remaining " +
QString::number(fileSize % chunkSize) + " bytes.");
            }*/
            chart->setTitle("Average file entropy across " + QString::number(chunks+1) + " chunk
of " + chunkBytes);
        }
        else {
            chart->setTitle("Average file entropy across " + QString::number(chunks) + " chunks
of " + chunkBytes + " each.");
```

```
            }

            chart->addAxis(axisX, Qt::AlignBottom);
            series->attachAxis(axisX);
            axisX->setMax(8);
            axisX->applyNiceNumbers();

            chart->legend()->setVisible(false);
            chart->setAnimationOptions(QChart::AllAnimations);

            QChartView *chartView = new QChartView(chart);
            ui->scrollArea->setWidget(chartView);

            entropyGraphBuilt = true;
        }
    }
}

void MainWindow::refreshDisassembly()
{
    if (fileOpened) {

        if (!disassemblyBuilt) {
```

```
    maxDisplayInstructions = 31;
    getDisassembly();
}

ui->disassemblyBrowser->clear();

int instructionOffset = ui->disassemblyScrollBar->value();

// build display
QString displayInstructions;
for (int i = 0; i < maxDisplayInstructions; i++) {
    displayInstructions += disassembly[instructionOffset + i];
}

// remove last <br>
displayInstructions.remove(displayInstructions.size() - 4, 4);

// display things
QString currentPage = QString::number(ui->disassemblyScrollBar->value() + 1);
QString maxPage = QString::number(ui->disassemblyScrollBar->maximum() + 1);
QString display;
display.append(currentPage);
display.append(" / ");
```

```
        display.append(maxPage);
        ui->disassemblyPageNumberValue->setText(display);

        // font
        displayInstructions.prepend("<p style='font-family:Courier;font-size:20px;'>");
        displayInstructions.append("</p>");
        ui->disassemblyBrowser->setHtml(displayInstructions);
    }
}

void MainWindow::getDisassembly()
{
    if (!disassemblyBuilt && codeEndLoc != 0) {

        // put opcodes in map
        QFile file("OPCODES.txt");
        if (file.open(QIODevice::ReadOnly)) {
            QTextStream in(&file);
            int i = 0;
            while (!in.atEnd()) {
                QString line = in.readLine();
                QStringRef type(&line, 0, 1);
                QStringRef opcode(&line, 1, line.size() - 1);
```

```
            opTypeMap[i] = type.toInt();
            opcodeMap[i] = opcode.toString();
            i++;
        }
        file.close();
    }
    else {
        codeEndLoc = 0;
        disassembly += "Opcode text file could not be opened!<br>Try running as administrator or reinstalling the program!";
    }

    // get disassembled code
    disassembly += disassembleSection(codeStartLoc, codeEndLoc, codeVirtualAddress);

    // set ui things
    ui->disassemblyScrollBar->setMaximum(disassembly.count() - maxDisplayInstructions);
    ui->disassemblyBrowser->setOpenExternalLinks(false);
    reseting = true;
    ui->disassemblyScrollBar->setValue(codeStartProcedure);
    reseting = false;
    ui->disassemblyBrowser->setStyleSheet("background-color: #F0F0F0;");
}
```

```
    else {
        // if error during PE scan
        if (codeEndLoc == 0) {
            if (PE) {
                disassembly += "File could not be disassembled. However, it is a PE file. Is the file packed?<br>";
            }
            else {
                disassembly += "File could not be disassembled. Not a 32 bit PE File?<br>";
            }
            maxDisplayInstructions = 1;
        }
    }
    disassemblyBuilt = true;
}
```

QStringList MainWindow::disassembleSection(int start, int end, int virtualAddress)

```
{
    int instructionSizeOffset = 0, instrucionDisplayOffset = imagebase + virtualAddress, startLocation = codeEntryPoint - baseOfCode;
    QMap<QString, bool> locMap, subMap;
    QString startLocationString = "";
    QStringList disassemblyList;
```

```
    // for each instruction in a code section
    while (start + instructionSizeOffset < end) {

        // current instruction variables
        bool instructionComplete = false, error = false, secondImmediate = false, hasModByte =
false, seperator = false;
        int opcodeByte = 0, instructionStartByte = instructionSizeOffset, errorStartLocation = 0;
        int immediateValue = 0, displacementValue = 0;
        QString instruction = "", parameters = "";

        // current byte types
        bool prefix = true, opcode = false, modByte = false, SIB = false, extendedOpcode = false;
        bool operandSizeModifier = false, specialInstruction = false, aligning = false,
rareInstruction = false, hasPrefix = false, segementOverride = false, canHaveString = false;
        int mod = 0, reg = 0, rm = 0, scale = 0, index = 0, base = 0, displacementIndex = 0,
maxDisplacements = 0, immediateIndex = 0, maxImmediates = 0, operandSize = 8, extended =
0;
        QString operand1 = "", operand2 = "", operandPrefix = "", disassemblyLine = "", segment =
"ds:";
        bool operand1isDestination = false, noOperands = false;
        bool weird = false;
        QString imVal = "", secImVal = "";
```

```
// for each byte in instruction
while (!instructionComplete) {
    char c = rawData[codeStartLoc + instructionSizeOffset];
    unsigned char byte = static_cast<unsigned char>(c);

    //
    // OPTIONAL INSTRUCTION PREFIX BYTE CHECK (F0, F2, F3, 26, 2E, 36, 3E, 64, 65, 66, 67)
    //

    if (prefix) {

        // if LOCK prefix (F0)
        if (byte == 240) {
            instruction = "lock ";
            hasPrefix = true;
        }
        // if string manipulation prefix (F2, F3)
        else if (byte == 242 || byte == 243) {
            if (byte == 242) {
                instruction = "repnz ";
            }
            else {
```

```
            instruction = "rep ";
        }
        hasPrefix = true;
    }
    // if segment override prefix (26, 2E, 36, 3E, 64, 65)
    else if (byte == 38 || byte == 46 || byte == 54 || byte == 62 ||  byte == 100 || byte ==
101) {
        switch (byte) {
            case 38: segment = "es:";
            break;
            case 46: segment = "cs:";
            break;
            case 54: segment = "ss:";
            break;
            case 62: segment = "ds:";
            break;
            case 100: segment = "fs:";
            break;
            case 101: segment = "gs:";
            break;
        }
        segementOverride = true;
    }
```

```
    // if operand override (66)
    else if (byte == 102) {
        operandSizeModifier = true;
    }
    // if address override (67)
    else if (byte == 103) {
        rareInstruction = true;
    }
    else {
        prefix = false;
        opcode = true;
    }
}

//
// OPCODE BYTE CHECK
//

if (opcode) {
    opcodeByte = byte;
    // if extended instruction
    if (byte == 15) {
        extendedOpcode = true;
```

```
    extended = 256;
    rareInstruction = true;
}
else {
    opcode = false;
    // if this instruction has a mod byte
    if (opTypeMap[byte + extended] == 2) {
        modByte = true;
        hasModByte = true;

        // calculate operand size (default = 8bits)
        if (byte % 2 == 1) {
            if (operandSizeModifier) {
                operandSize = 16;
                operandPrefix = "word ptr ";
            }
            else {
                operandSize = 32;
                operandPrefix = "dword ptr ";
            }
        }
        else {
            operandPrefix = "byte ptr ";
```

```
            }


            if (extendedOpcode) {
                // has immediate constant
                if (byte == 58 || (byte >= 112 && byte <= 115) || byte == 164 || byte == 172 ||
byte == 178 || byte == 180 || byte == 181 || byte == 186 || byte == 194 || byte == 196 || byte ==
197 || byte == 198) {
                    if (byte == 178 || byte == 180 || byte == 181) {
                        maxImmediates = operandSize / 8;
                    }
                    else {
                        maxImmediates = 1;
                    }
                }
                // is correct
                if ((byte >= 144 && byte <= 159)) {
                    rareInstruction = false;
                }
            }
            else {
                // has immediate constant (69, 6B, 80 - 83, C0, C1, C6, C7)
                if (byte == 105 || byte == 107 || (byte >= 128 && byte <= 131) || byte == 192 ||
byte == 193 || byte == 198 || byte == 199) {
```

```
                // only 69, 81, C7 has immediate bigger than 1
                if (byte == 105 || byte == 129 || byte == 199) {
                    maxImmediates = operandSize / 8;
                }
                else {
                    maxImmediates = 1;
                }
                // immediate can refer to a string (C6, C7)
                if (byte == 198 || byte == 199) {
                    canHaveString = true;
                }
            }

            // if opcode has multiple possible instructions (80 - 83, C0, C1, D0 - D3, D8 - DF, F6, F7, FF)
            if ((byte >= 128 && byte <= 131) || byte == 192 || byte == 193 || (byte >= 208 && byte <= 211) || (byte >= 216 && byte <= 223) || byte == 246 || byte == 247 || byte == 255) {
                specialInstruction = true;
            }
        }
    }
    // else if single byte instruction
    else if (opTypeMap[byte + extended] == 1) {
```

```
instructionComplete = true;
operand1isDestination = true;

int opcodeRmBits;
opcodeRmBits = (byte / static_cast<int>(pow(2, 2)) % 2) * 100;
opcodeRmBits += (byte / 2 % 2) * 10;
opcodeRmBits += (byte % 2);

if (operandSizeModifier) {
   operandSize = 16;
   operandPrefix = "word ptr ";
}
else {
   operandSize = 32;
   operandPrefix = "dword ptr ";
}

// if opcode is (40 - 5F)
if (byte >= 64 && byte <= 95) {
   operand1 = registerName(opcodeRmBits, operandSize);
}
// if opcode is (6C - 6F)
else if (byte >= 108 && byte <= 111) {
```

```
    // if 8 bit
    if (byte % 2 == 0) {
        operand2 = "byte ptr ";
    }
    else {
        operand2 = operandPrefix;
    }
    // if 6C or 6D
    if (byte / 2 % 2 == 0) {
        operand1isDestination = false;
        operand2 += "es:[edi]";
    }
    else {
        operand2 += segment + "[esi]";
    }
    operand1 = "dx";
}
// if opcode is (91 - 97)
else if (byte >= 145 && byte <= 151) {
    operand1 = registerName(opcodeRmBits, operandSize);
    operand2 = registerName(0, operandSize);
}
// if opcode is (98)
```

```
else if (byte == 152) {
    if (operandSizeModifier) {
        instruction += "cbw";
    }
    else {
        instruction += "cwde";
    }
}
// if opcode is (99)
else if (byte == 153) {
    if (operandSizeModifier) {
        instruction += "cwd";
    }
    else {
        instruction += "cdq";
    }
}
// if opcode is (A4 - A7)
else if (byte >= 164 && byte <= 167) {
    if (byte % 2 == 1) {
        operand1 = operandPrefix;
        operand2 = operandPrefix;
    }
```

```
    else {
        operand1 = "byte ptr ";
        operand2 = "byte ptr ";
    }

    if (byte / 2 % 2 == 0) {
        operand1 += "es:[edi]";
        operand2 += segment + "[esi]";
    }
    else {
        operand1 += segment + "[esi]";
        operand2 += "es:[edi]";
    }
}
// if opcode is (AA - AF)
else if (byte >= 170 && byte <= 175) {
    // if AA or AB
    if (byte / 4 % 2 == 0) {
        operand1isDestination = false;
    }
    // if AA, AB, AE, AF
    if (byte / 2 % 2 == 1) {
        operand2 += "es:[edi]";
```

```
    }
    else {
        operand2 += segment + "[esi]";
    }
    operand1 = registerName(0, operandSize);
}
// if opcode is (CC)
else if (byte == 204) {
    if (!aligning) {
        errorStartLocation = instructionStartByte;
        aligning = true;
        error = true;
    }
    if (instructionSizeOffset % 16 != 15) {
        instructionComplete = false;
        opcode = true;
    }
}
// if opcode is (D7)
else if (byte == 215) {
    operand1 = "byte ptr " + segment + "[ebx]";
}
// if opcode is (EC - EF)
```

```
    else if (byte >= 236 && byte <= 239) {
        operand1 = "dx";
        if (byte / 2 % 2 == 0) {
            operand1isDestination = false;
        }
        if (byte % 2 == 0) {
            operand2 = "al";
        }
        else {
            operand2 = "eax";
        }
    }
    else {
        noOperands = true;
    }

    // if call or jump, add seperator
    if (byte == 195 || byte == 203) {
        seperator = true;
    }
}
// else has immediate value(s) only
else {
```

```
// calculate operand size (default 8 bits)
if (operandSizeModifier) {
    operandSize = 16;
}
else {
    operandSize = 32;
}

if (extendedOpcode) {
    if (byte >= 128 && byte <= 144) {
        maxImmediates = operandSize / 8;
        rareInstruction = false;
    }
}
else {
    int opcodeRmBits;
    opcodeRmBits = (byte / static_cast<int>(pow(2, 2)) % 2) * 100;
    opcodeRmBits += (byte / 2 % 2) * 10;
    opcodeRmBits += (byte % 2);

    // if opcode is (04, 05, 0C, 0D, 14, 15, 1C, 1D, 24, 25, 2C, 2D, 34, 35, 3C, 3D)
    if (byte >= 4 && byte <= 61) {
        operand1 = registerName(0, operandSize);
```

```
                operand1isDestination = true;
                maxImmediates = operandSize / 8;
            }
            // if has Short jump on condition (70 - 7F, E0 - E3, EB)
            else if ((byte >= 112 && byte <= 127) || (byte >= 224 && byte <= 227) || byte
== 235) {
                operand2 = "short ";
                maxImmediates = 1;
            }
            // if opcode is (E4 - E7)
            else if (byte >= 228 && byte <= 231) {
                if (byte % 2 == 0) {
                    operand1 = "al";
                }
                else {
                    operand1 = "eax";
                }
                if (byte / 2 % 2 == 0) {
                    operand1isDestination = true;
                }
                maxImmediates = 1;
            }
            // if opcode is (E8, E9)
```

```
else if (byte == 232 || byte == 233) {
    if (byte == 233) {
        operand2 = "short ";
    }
    if (operandSizeModifier) {
        maxImmediates = 2;
    }
    else {
        maxImmediates = 4;
    }
}
// if move immediate byte into 8 bit register (B0 - B7)
else if (byte >= 176 && byte <= 183) {
    maxImmediates = 1;
    operand1 = registerName(opcodeRmBits, 8);
    operand1isDestination = true;
}
// if move immediate word/double into 16/32 bit register (B8 - BF)
else if (byte >= 184 && byte <= 191) {
    maxImmediates = operandSize / 8;
    operand1 = registerName(opcodeRmBits, operandSize);
    operand1isDestination = true;
}
```

```
// if single operand immediate value (68, E8, E9)
else if (byte == 104 || byte == 232 || byte == 233) {
    maxImmediates = operandSize / 8;
}
// if single operand immediate byte (6A)
else if (byte == 106) {
     maxImmediates = 1;
}
// if opcode is (9A)
else if (byte == 154) {
    maxImmediates = (operandSize / 8) + 2;
}
// if opcode is (EA)
else if (byte == 234) {
    operand2 = "short ";
    maxImmediates = (operandSize / 8) + 2;
}
// if opcode is (A0 - A3)
else if (byte >= 160 && byte <= 163) {
    if (byte / 2 % 2 == 0) {
        operand1isDestination = true;
    }
    maxImmediates = 4;
```

```
    operand1 = registerName(0, operandSize);
    operand2 = segment;
}
// if opcode is (A8, A9)
else if (byte == 168 || byte == 169) {
    if (byte % 2 == 1) {
        operand1 = registerName(0, operandSize);
    }
    else {
        operand1 = registerName(0, 8);
    }
    maxImmediates = operandSize / 8;
    operand1isDestination = true;
}
// if opcode is (C2)
else if (byte == 194) {
    maxImmediates = 2;
}
// if opcode is (C8)
else if (byte == 200) {
    maxImmediates = 3;
}
// if opcode is (CA)
```

```
                else if (byte == 202) {
                    maxImmediates = 2;
                }
                // if opcode is (CD)
                else if (byte == 205) {
                    maxImmediates = 1;
                }
                // if opcode is (D4, D5)
                else if (byte == 212 || byte == 213) {
                    maxImmediates = 1;
                }

                // if call or jump, add seperator
                if ((byte >= 233 && byte <= 235) || byte == 194 || byte == 202) {
                    seperator = true;
                }

                // immediate can refer to a string (68, 6A, (A0 - A3), (B0 - BF))
                if (byte == 104 || byte == 106 || (byte >= 160 && byte <= 163) || (byte >= 176
&& byte <= 191)) {
                    canHaveString = true;
                }
            }
```

```
            }

            // if weird opcode

            if (byte == 98 || byte == 99 || byte == 141 || byte == 196 || byte == 197 || (byte >=
216 && byte <= 223) || byte == 255) {

                weird = true;

            }

        }

    }


    //

    // MOD REG R/M BYTE CHECK

    //


    else if (modByte) {

        // get addressing mode bits (first 2 bits)

        mod = (byte / static_cast<int>(pow(2, 7)) % 2) * 10;

        mod += (byte / static_cast<int>(pow(2, 6)) % 2);


        // get register bits (next 3 bits)

        reg = (byte / static_cast<int>(pow(2, 5)) % 2) * 100;

        reg += (byte / static_cast<int>(pow(2, 4)) % 2) * 10;

        reg += (byte / static_cast<int>(pow(2, 3)) % 2);
```

```
// get r/m bits (last 3 bits)
rm = (byte / static_cast<int>(pow(2, 2)) % 2) * 100;
rm += (byte / 2 % 2) * 10;
rm += (byte % 2);

if (specialInstruction) {
    instruction += getSpecialByteInstruction(opcodeByte, reg);
}

if (mod == 1) {
    maxDisplacements = 1;
}
else if (mod == 10) {
    maxDisplacements = 4;
}

if (opcodeByte == 255) {
    operand2 = operandPrefix;
}

// if has SIB
if (rm == 100 && mod != 11) {
    SIB = true;
```

```
}


// if not extended
if (!extended) {
    // if opcode is (00 - 03, 08 - 0B, 10 - 13, 18 - 1B, 20 - 23, 28 - 2B, 30 - 33, 38 - 3B)
    if (opcodeByte < 60) {
        operand1 = registerName(reg, operandSize);
        if (opcodeByte / 2 % 2 == 1) {
            operand1isDestination = true;
        }
    }
    // if opcode is (62)
    else if (opcodeByte == 98) {
        if (operandSizeModifier) {
            operand2 = "dword ptr ";
            operandSize = 16;
        }
        else {
            operand2 = "qword ptr ";
            operandSize = 32;
        }
        operand1 = registerName(reg, operandSize);
        operand1isDestination = true;
```

```
    if (mod == 0) {
        maxDisplacements = 4;
    }
}
// if opcode is (63)
else if (opcodeByte == 99) {
    operand1 = registerName(reg, 16);
    operand2 = "word ptr ";
}
// if opcode is (69, 6B)
else if (opcodeByte == 105 || opcodeByte == 107) {
    operand1 = registerName(reg, operandSize);
    operand1isDestination = true;
}
// if opcode is (84 - 89)
else if (opcodeByte >= 132 && opcodeByte <= 137) {
    operand1 = registerName(reg, operandSize);
}
// if opcode is (8A, 8B)
else if (opcodeByte == 138 || opcodeByte == 139) {
    operand1 = registerName(reg, operandSize);
    operand1isDestination = true;
}
```

```
// if opcode is (8C, 8E)
else if (opcodeByte == 140 || opcodeByte == 142) {
    operand1 = segmentRegisterName(reg);
    operand2 = "word ptr ";
    if (opcodeByte / 2 % 2 == 1) {
        operand1isDestination = true;
    }
}
// if opcode is (8D)
else if (opcodeByte == 141) {
    operand1 = registerName(reg, operandSize);
    operand1isDestination = true;
}
// if opcode is (C4, C5)
else if (opcodeByte == 196 || opcodeByte == 197) {
    operand1 = registerName(reg, 32);
    operand1isDestination = true;
    if (operandSizeModifier) {
        operand2 = "dword ptr ";
    }
    else {
        operand2 = "fword ptr ";
    }
```

```
}
// if opcode is (D0 - D3)
else if (opcodeByte >= 208 && opcodeByte <= 211) {
    if (opcodeByte / 2 % 2 == 0) {
        operand1 = "1";
    }
    else {
        operand1 = "cl";
    }
}
// if opcode is (F6)
else if (opcodeByte == 246) {
    if (reg == 0) {
        maxImmediates = 1;
    }
}
// if opcode is (F7)
else if (opcodeByte == 247) {
    if (reg == 0) {
        maxImmediates = operandSize / 8;
    }
}
// if opcode is (FE)
```

```
    else if (opcodeByte == 254) {
        if (reg == 0) {
            instruction += "inc ";
        }
        else {
            instruction += "dec ";
        }
    }
}

if (!SIB) {
    // register addressing mode
    if (mod == 11) {
        operand2 = registerName(rm, operandSize);
    }
    // if 32 bit displacement only mode
    else if (mod == 0 && rm == 101) {
        maxDisplacements = 4;
    }
    else {
        if (maxDisplacements == 0) {
            operand2 += "[" + registerName(rm, 32) + "]";
        }
```

```
            else {
                operand2 += "[" + registerName(rm, 32);
            }
        }
    }

    if (maxDisplacements == 0 && maxImmediates == 0 && !SIB) {
        instructionComplete = true;
    }

    modByte = false;
}

//
// SIB CHECK
//

else if (SIB) {
    // get scale bits (first 2 bits)
    scale = (byte / static_cast<int>(pow(2, 7)) % 2) * 10;
    scale += (byte / static_cast<int>(pow(2, 6)) % 2);

    // get index bits (next 3 bits)
```

```
index = (byte / static_cast<int>(pow(2, 5)) % 2) * 100;
index += (byte / static_cast<int>(pow(2, 4)) % 2) * 10;
index += (byte / static_cast<int>(pow(2, 3)) % 2);

// get base bits (last 3 bits)
base = (byte / static_cast<int>(pow(2, 2)) % 2) * 100;
base += (byte / 2 % 2) * 10;
base += (byte % 2);

// get index register
if (!weird) {
    // get base register
    if (mod != 101) {
        operand2 += "[" + registerName(base, 32);
    }
    else if (mod == 1 || mod == 10) {
        operand2 = "[ebp";
    }
    operand2 += "+" + registerName(index, 32);
}
else {
    operand2 += "[" + registerName(index, 32);
    maxDisplacements = 4;
```

```
    }

    // get index scale value
    switch (scale) {
        case 0: operand2 += "*1";
        break;
        case 1: operand2 += "*2";
        break;
        case 10: operand2 += "*4";
        break;
        case 11: operand2 += "*8";
        break;
    }

    SIB = false;
    if (maxDisplacements == 0) {
        instructionComplete = true;
        operand2 += "]";
    }
    if (maxImmediates != 0) {
        instructionComplete = false;
    }
}
```

```
//
// DISPLACEMENT BYTE CHECK
//

else if (displacementIndex < maxDisplacements) {
    // sign check
    bool negative = false;
    if (displacementIndex == maxDisplacements - 1) {
        if (byte >= 128) {
            negative = true;
            if (maxDisplacements == 1) {
                byte -= (byte % 128) * 2;
            }
        }
    }

    qint64 k = pow(16,displacementIndex * 2) * byte;
    displacementValue += k;
    displacementIndex++;

    if (displacementIndex == maxDisplacements) {
        QString fontStart = "", fontEnd = "";
```

```
            if (!rareInstruction) {
                fontStart = "<font color='green'>";
                fontEnd = "</font>";
            }
            if (negative) {
                if (maxDisplacements > 1) {
                    displacementValue *= -1;
                }
                operand2 += fontStart + "-" + QString::number(displacementValue, 16).toUpper()
+ "h" + fontEnd + "]";
            }
            else {
                operand2 += fontStart + "+" + QString::number(displacementValue,
16).toUpper() + "h" + fontEnd + "]";
            }

            // if register addressing mode
            if (mod == 0 && rm == 101) {
                QString function = getFunctionCallName(displacementValue);

                if (function.size() > 4) {
                    fontStart = segment + "<font color='red'>";
                    fontEnd = "</font>";
```

```
                operand2 = fontStart + function + fontEnd;
            }
            else {
                if (segementOverride) {
                    operand2 = "large " + segment + "<font color='green'>" +
QString::number(displacementValue, 16).toUpper() + "</font>";
                }
                else {
                    operandPrefix.remove(operandPrefix.size() - 5, 5);
                    operandPrefix += "_";
                    operand2 = operandPrefix + "<font color='green'>" +
QString::number(displacementValue, 16).toUpper() + "</font>";
                }
            }
        }

        if (maxImmediates == 0) {
            instructionComplete = true;
        }
    }
}

//
```

```
// IMMEDIATE BYTE CHECK
//


else if (immediateIndex < maxImmediates) {
   // if opcode is (9A, EA)
   if (opcodeByte == 154 || opcodeByte == 234) {
      if (immediateIndex == maxImmediates - 2 && !secondImmediate) {
         secImVal = imVal;
         imVal = "";
         immediateIndex = 0;
         maxImmediates = 2;
         secondImmediate = true;
      }
   }
   // if opcode is (C8)
   else if (opcodeByte == 200) {
      if (immediateIndex == 2  && !secondImmediate) {
         secImVal = imVal;
         imVal = "";
         immediateIndex = 0;
         maxImmediates = 1;
         secondImmediate = true;
      }
```

```
}


// immediate value in hex
imVal.prepend(byteToHexString(byte));
// if signed value
if (opcodeByte == 106 || opcodeByte == 107 || opcodeByte == 131) {
    if (byte >= 128) {
        imVal.prepend("FFFFFF");
    }
}


// immediate value in dec
bool negative = false;
if (maxImmediates == 1) {
    if (byte >= 128) {
        negative = true;
        byte -= (byte % 128) * 2;
    }
}
qint64 k = pow(16,immediateIndex * 2) * byte;
immediateValue += k;
immediateIndex++;
```

```
            // end
            if (immediateIndex == maxImmediates) {
                imVal = immediateFormat(imVal);
                secImVal = immediateFormat(secImVal);


                if (opcodeByte == 154 || opcodeByte == 234) {
                    operand2 = imVal + ":";
                    operand2 += secImVal;
                }
                else if (opcodeByte == 200) {
                    operand1 = imVal;
                    operand2 = secImVal;
                }
                else if (!hasModByte) {
                    if (negative) {
                        immediateValue *= -1;
                    }
                    // if a jump to a location
                    if (operand2 == "short " || (opcodeByte >= 128 && opcodeByte <= 143 &&
extended)) {
                        QString loc = QString::number(immediateValue + (instructionSizeOffset + 1)
+ 4198400, 16).toUpper();
                        operand2 += "<a href='" + loc + "'>";
```

```
                operand2 += "loc_" + loc + "</a>";
                locMap[loc] = true;
            }
            // if a call to a location
            else if (opcodeByte == 154 || opcodeByte == 232) {
                QString sub = QString::number(immediateValue + (instructionSizeOffset + 1)
+ 4198400, 16).toUpper();
                operand2 += "<a href='" + sub + "'>";
                operand2 += "sub_" + sub + "</a>";
                subMap[sub] = true;
            }
            // if a segment pointer, try to get name
            else if (operand2 == segment) {
                QString function = getFunctionCallName(immediateValue);
                if (function.size() > 4) {
                    QString fontStart = segment + "<font color='red'>";
                    QString fontEnd = "</font>";
                    operand2 = fontStart + function + fontEnd;
                }
                else if (segementOverride) {
                    operand2 = "large " + segment + "<font color='green'>" +
QString::number(immediateValue, 16).toUpper() + "</font>";
                }
```

```
                    else {
                        operandPrefix.remove(operandPrefix.size() - 5, 5);
                        operandPrefix += "_";
                        operand2 = operandPrefix + + "<font color='green'>" +
QString::number(immediateValue, 16).toUpper() + "</font>";
                    }
                }
                // if immediate value can be reference to a string
                else if (canHaveString) {
                    operand2 += imVal;
                    if (!stringsBuilt) {
                        findStrings();
                    }
                    QString string = "";
                    // try rdata section if found
                    if (rdataRVA > 0) {
                        string = immediateIsStringOffset(immediateValue, rdataStartLoc,
rdataRVA);
                    }
                    // else try data section if found and rdata is not the right section
                    if (dataVirtualAddress > 0 && string == "") {
                        string = immediateIsStringOffset(immediateValue, dataStartLoc,
dataVirtualAddress);
```

```
                }
                // if a string is being referenced, display it
                if (string.size() > 0) {
                    operand2 += " ; ";
                    operand2 += 34 + string + 34;
                }
            }
            else {
                operand2 += imVal;
            }
        }
        else {
            // if immediate value can be reference to a string
            if (canHaveString) {
                operand1 += imVal;
                if (!stringsBuilt) {
                    findStrings();
                }
                QString string = "";
                // try rdata section if found
                if (rdataRVA > 0) {
                    string = immediateIsStringOffset(immediateValue, rdataStartLoc,
rdataRVA);
```

```
                    }
                    // else try data section if found
                    if (dataVirtualAddress > 0 && string == "") {
                        string = immediateIsStringOffset(immediateValue, dataStartLoc,
dataVirtualAddress);
                    }
                    // if a string is being referenced, display it
                    if (string.size() > 0) {
                        operand1 += " ; ";
                        operand1 += 34 + string + 34;
                    }
                }
                else {
                    operand1 += imVal;
                }
            }
            instructionComplete = true;
        }
    }

    //
    // END OF INSTRUCTION LOOP CHECKS
    //
```

```
            else if (!prefix) {
                instructionComplete = true;
                error = true;
            }
            instructionSizeOffset++;
        }

        //
        // GET LINE READY FOR OUTPUT
        //

        QString location = QString::number(instructionStartByte + instrucionDisplayOffset,
16).toUpper();
        disassemblyLine += location;
        QString spacing =
"            
  ";

        if (!error) {
            // get instruction mnemonic if not special
            if (instruction == "" || hasPrefix) {
                if (extendedOpcode) {
```

```
        opcodeByte += 256;
    }
    QString line =  opcodeMap[opcodeByte];
    if (noOperands) {
        instruction += line;
    }
    else {
        int split = line.indexOf(" ");
        if (split > 0) {
            QStringRef instructionRef(&line, 0, split + 1);
            instruction += instructionRef.toString();
        }
        else {
            instruction += line;
        }
    }
}

disassemblyLine += spacing;

if (rareInstruction) {
    disassemblyLine += instruction;
}
```

```
    else {
        disassemblyLine += "<font color='blue'>";
        disassemblyLine += instruction;
        disassemblyLine += "</font>";
    }

    if (!operand1isDestination) {
        if (operand2 != "") {
            disassemblyLine += operand2;
            if (operand1 != "") {
                disassemblyLine += ", " + operand1;
            }
        }
    }
    else if (operand1 != "") {
        disassemblyLine += operand1;
        if (operand2 != "") {
            disassemblyLine += ", " + operand2;
        }
    }

    disassemblyLine += "<br>";
}
```

```
        // aligning or extended opcode that is not handled
        else {
            if (aligning) {
                aligning = false;
                disassemblyLine = QString::number(errorStartLocation + 4198400, 16).toUpper() +
spacing + "align 10h<br>";
            }
            else {
                disassemblyLine += "<font color='red'>ERROR!<br></font>";
            }
        }

        if (rareInstruction) {
            disassemblyLine.prepend("<font color='red'>");
            disassemblyLine.append("</font>");
        }

        disassemblyList += disassemblyLine;

        // if code start procedure
        if (startLocation == instructionStartByte) {
            // for finding start location when location and sub location labels are inserted after all
instructions are built
```

```
            startLocationString = location;
        }
        else if (seperator) {
            disassemblyLine += location + " ; ----------------------------------------------------------------
-<br>";
        }
    }

    //
    // FORMATTING AFTER DISASSEMBLY BUILT
    //

    // for all locations and sub locations
    int i = 0, locCount = 0, totalLocs = disassemblyList.count(), startjumpOffset = 0;
    while (i < totalLocs) {
        QString instruction = disassemblyList[i + (locCount * 2)];
        int split = instruction.indexOf('&');
        QStringRef instructionRef(&instruction, split - 6, split);
        QString loc = instructionRef.toString();

        // if start location, jump offsets need to be increased by start label length (currently 6 lines)
        if (loc == startLocationString) {
            startjumpOffset = 6;
```

```
        }


        if (locMap[loc]) {
            disassemblyList.insert(i + (locCount * 2), loc + "<br>");
            disassemblyList.insert(i + (locCount * 2) + 1, loc + " loc_" + loc + "<br>");
            locOffsetMap[loc] = i + (locCount * 2) + startjumpOffset;
            locCount++;
        }
        else if (subMap[loc]) {
            disassemblyList.insert(i + (locCount * 2), loc + "<br>");
            disassemblyList.insert(i + (locCount * 2) + 1, loc + " sub_" + loc + "<br>");
            locOffsetMap[loc] = i + (locCount * 2) + startjumpOffset;
            locCount++;
        }
        i++;
    }


    // find start procedure (needs to be after locations and sublocations are added because jumps after start may jump to locations behind the start)
    bool startFound = false;
    int size = disassemblyList.count();
    i = 0;
    codeStartProcedure = 0;
```

```
    while (!startFound && i < size) {
        QString instruction = disassemblyList[i];
        QStringRef instructionRef(&instruction, 0, 6);
        QString loc = instructionRef.toString();
        if (loc == startLocationString) {
            QStringList list;
            list += startLocationString + "<font color='blue'> ; ----------------------------------------------------------<br></font>";
            list += startLocationString + "<font color='blue'> ; ----------------------------------------------------------<br></font>";
            list += startLocationString + "<font color='blue'> ; ---------- <font color='red'>START PROCEDURE</font> ----------<br></font>";
            list += startLocationString + "<font color='blue'> ; ----------------------------------------------------------<br></font>";
            list += startLocationString + "<font color='blue'> ; ----------------------------------------------------------<br></font>";
            list += startLocationString + "<br>";

            for (int j = 0; j < list.size(); j++) {
                disassemblyList.insert(i + j , list.at(j));
            }

            codeStartProcedure = i;
            startFound = true;
```

```
        }
        else {
            i++;
        }
    }
    return disassemblyList;
}
```

QString MainWindow::immediateIsStringOffset(int immediateValue, int startLoc, int virtualAddress)

```
{
    int dataOffset = immediateValue - (imagebase + virtualAddress);
    int physicalAddress = startLoc + dataOffset;
    if (physicalAddress > 0 && physicalAddress < fileSize) {
        QString string = stringLocationMap[physicalAddress];
        if (string.size() > 0) {
            string = htmlSanitiseString(string);
            return string;
        }
    }
    return "";
}
```

QString MainWindow::htmlSanitiseString(QString string)

```
{
    // if string has '<' or '>'
    for (int i = 0; i < string.size(); i++) {
        if (string.at(i) == '<') {
            string.remove(i, 1);
            string.insert(i, "&#60;");
        }
        else if (string.at(i) == '>') {
            string.remove(i, 1);
            string.insert(i, "&#62;");
        }
    }
    return string;
}
```

QString MainWindow::immediateFormat(QString s)

```
{
    // remove zeros
    bool stillZero = true;
    while (stillZero) {
        if (s.size() > 0 && s[0] == '0') {
            s.remove(0, 1);
```

```
        }
        else {
            stillZero = false;
        }
    }
    // add h if not value between 0 and 9
    if (s.size() == 1) {
        if (s[0] >= 65 && s[0] <= 70) {
            s += "h";
        }
    }
    else if (s.size() > 0) {
        s += "h";
    }
    else {
        s+= "0";
    }
    s.prepend("<font color='green'>");
    s.append("</font>");
    return s;
}
```

```
QString MainWindow::getFunctionCallName(int immediateValue)
{
    // get function name
    int location = 0, sectionStart = 0, sectionRVA = 0, pointerStartLoc = 0, pointerLocationOffset
= 0;
    if (IATLoc > 0) {
        pointerStartLoc = IATLoc;
    }
    else {
        pointerStartLoc = IDTLoc + IDTSize;
    }
    if (idataStartLoc > 0) {
        sectionRVA = idataRVA;
        sectionStart = idataStartLoc;
    }
    else {
        sectionRVA = rdataRVA;
        sectionStart = rdataStartLoc;
    }
    pointerLocationOffset = immediateValue - (imagebase + sectionRVA);
    if (pointerStartLoc + pointerLocationOffset + 4 < fileSize && pointerStartLoc +
pointerLocationOffset + 4 > 0) {
        QString functionName = "";
```

```
        for (int i = 0; i < 4; i++) {
            location += pow(16, i * 2) * static_cast<unsigned char>(rawData[pointerStartLoc + pointerLocationOffset + i]);
        }
        functionName = getFunctionName(location, sectionRVA, sectionStart);
        if (!dllsBuilt) {
            findDLLs();
        }
        // check if function exists in imports
        for (int i = 0; i < dllFunctionNames.size(); i++) {
            if (functionName == dllFunctionNames.at(i)) {
                return functionName;
            }
        }
    }
    return "";
}
```

QString MainWindow::registerName(int reg, int operandSize)

```
{
    QString fontStart = "<font color='#ff00ff'>", fontEnd = "</font>", registerName = "";
    switch (reg) {
        case 0: switch (operandSize) {
```

```
            case 8: registerName = "al";
            break;
            case 16: registerName = "ax";
            break;
            case 32: registerName = "eax";
            break;
        }
break;
case 1: switch (operandSize) {
            case 8: registerName = "cl";
            break;
            case 16: registerName = "cx";
            break;
            case 32: registerName = "ecx";
            break;
        }
break;
case 10: switch (operandSize) {
            case 8: registerName = "dl";
            break;
            case 16: registerName = "dx";
            break;
            case 32: registerName = "edx";
```

```
            break;
        }
break;
case 11: switch (operandSize) {
            case 8: registerName = "bl";
            break;
            case 16: registerName = "bx";
            break;
            case 32: registerName = "ebx";
            break;
        }
break;
case 100: switch (operandSize) {
            case 8: registerName = "ah";
            break;
            case 16: registerName = "sp";
            break;
            case 32: registerName = "esp";
            break;
        }
break;
case 101: switch (operandSize) {
            case 8: registerName = "ch";
```

```
            break;
            case 16: registerName = "bp";
            break;
            case 32: registerName = "ebp";
            break;
        }
break;
case 110: switch (operandSize) {
            case 8: registerName = "dh";
            break;
            case 16: registerName = "si";
            break;
            case 32: registerName = "esi";
            break;
        }
break;
case 111: switch (operandSize) {
            case 8: registerName = "bh";
            break;
            case 16: registerName = "di";
            break;
            case 32: registerName = "edi";
            break;
```

```
            }
        break;
    }
    return fontStart + registerName + fontEnd;
}
```

QString MainWindow::getSpecialByteInstruction(int specialByte, int reg)

```
{
    if (specialByte == 255) {
        switch (reg) {
            case 0: return "inc ";
            break;
            case 1: return "dec ";
            break;
            case 10: return "call ";
            break;
            case 11: return "callf ";
            break;
            case 100: return "jmp ";
            break;
            case 101: return "jmpf ";
            break;
            case 110: return "push ";
```

```
        break;
    }
}
else if (specialByte == 128 || specialByte == 129 || specialByte == 130 || specialByte == 131) {
    switch (reg) {
        case 0: return "add ";
        break;
        case 1: return "or ";
        break;
        case 10: return "adc ";
        break;
        case 11: return "sbb ";
        break;
        case 100: return "and ";
        break;
        case 101: return "sub ";
        break;
        case 110: return "xor ";
        break;
        case 111: return "cmp ";
        break;
    }
}
```

```
    else if (specialByte == 192 || specialByte == 193 || (specialByte >= 208 && specialByte <=
211)) {
        switch (reg) {
            case 0: return "rol ";
            break;
            case 1: return "ror ";
            break;
            case 10: return "rcl ";
            break;
            case 11: return "rcr ";
            break;
            case 100: return "shl ";
            break;
            case 101: return "shr ";
            break;
            case 110: return "sal ";
            break;
            case 111: return "sar ";
            break;
        }
    }
    else if (specialByte == 246 || specialByte == 247) {
        switch (reg) {
```

```
            case 0: return "test ";
            break;
            case 1: return "test ";
            break;
            case 10: return "not ";
            break;
            case 11: return "neg ";
            break;
            case 100: return "mul ";
            break;
            case 101: return "imul ";
            break;
            case 110: return "div ";
            break;
            case 111: return "idiv ";
            break;
        }
    }
    return "";
}
```

```
QString MainWindow::getExtendedByteInstruction(int extendedByte, int reg)
{
```

```
    if (extendedByte == 0) {
        switch (reg) {
            case 0: return "sldt ";
            break;
            case 1: return "str ";
            break;
            case 10: return "lldt ";
            break;
            case 11: return "ltr ";
            break;
            case 100: return "verr ";
            break;
            case 101: return "verw ";
            break;
        }
    }
    return "";
}
```

QString MainWindow::segmentRegisterName(int reg)

```
{
    switch (reg) {
        case 0: return "es";
```

```
        break;
        case 1: return "cs";
        break;
        case 10: return "ss";
        break;
        case 11: return "ds";
        break;
        case 100: return "fs";
        break;
        case 101: return "gs";
        break;
    }
    return "";
}
```

```
void MainWindow::getPEinformation()
{
    // check if file is a PE file (first two bytes are 4D, 5A or M, Z in text)
    if (rawData[0] == 'M' && rawData[1] == 'Z') {
        PE = true;
    }
    else {
        PE = false;
```

```
}

imagebase = 0;
codeStartLoc = 0, codeEndLoc = 0, codeVirtualAddress = 0;
rdataStartLoc = 0, rdataRVA = 0;
idataStartLoc = 0, idataRVA = 0;
IDTLoc = 0, IDTSize = 0;
IATLoc = 0;
codeEntryPoint = 0, baseOfCode = 0;
dataStartLoc = 0, dataVirtualAddress = 0;

if (PE) {
   try {
      // if PE, find PE header location (3C - 3F)
      int peHeaderStartLoc = 0;
      for (int i = 0; i < 4; i++) {
         peHeaderStartLoc += pow(16, i * 2) * static_cast<unsigned char>(rawData[60 + i]);
      }
      int peHeaderSize = 24;

      // get number of sections in the file
      int sectionCount = 0;
      sectionCount = static_cast<unsigned char>(rawData[peHeaderStartLoc + 6]);
```

```
        sectionCount += pow(16, 2) * static_cast<unsigned char>(rawData[peHeaderStartLoc +
7]);


        // get size of optional header
        int optionalHeaderSize;
        optionalHeaderSize = static_cast<unsigned char>(rawData[peHeaderStartLoc + 20]);
        optionalHeaderSize += pow(16, 2) * static_cast<unsigned
char>(rawData[peHeaderStartLoc + 21]);


        // if file has an optional header
        if (optionalHeaderSize > 0) {


            // get image base
            for (int i = 0; i < 4; i++) {
                imagebase += pow(16, i * 2) * static_cast<unsigned
char>(rawData[peHeaderStartLoc + 52 + i]);
            }


            // find text, idata and rdata and data sections
            bool textFound = false, rdataFound = false, idataFound = false, dataFound = false;
            int sectionSize = 40, textLocation = 0, rdataLocation = 0, idataLocation = 0,
dataLocation = 0;
            int sectionLocation = peHeaderStartLoc + peHeaderSize + optionalHeaderSize,
sectionNumber = 0;
```

```
        while ((!textFound || !rdataFound || !idataFound || !dataFound) && sectionNumber <
sectionCount) {
            QString sectionName = "";
            for (int i = 0; i < 5; i++) {
                sectionName += rawData[sectionLocation + i];
            }
            if (sectionName == ".text") {
                textFound = true;
                textLocation = sectionLocation;
            }
            else if (sectionName == ".rdat") {
                rdataFound = true;
                rdataLocation = sectionLocation;
            }
            else if (sectionName == ".idat") {
                idataFound = true;
                idataLocation = sectionLocation;
            }
            else if (sectionName == ".data") {
                dataFound = true;
                dataLocation = sectionLocation;
            }
```

```
            sectionLocation += sectionSize;
            sectionNumber++;
        }

        if (textFound) {
            // find code start physical location
            for (int i = 0; i < 4; i++) {
                codeStartLoc += pow(16, i * 2) * static_cast<unsigned
char>(rawData[textLocation + 20 + i]);
            }
            // find code end physical location
            int virtualSize = 0;
            for (int i = 0; i < 4; i++) {
                virtualSize += pow(16, i * 2) * static_cast<unsigned char>(rawData[textLocation
+ 8 + i]);
            }
            codeEndLoc = virtualSize + codeStartLoc;
            // find code virtual address
            for (int i = 0; i < 4; i++) {
                codeVirtualAddress += pow(16, i * 2) * static_cast<unsigned
char>(rawData[textLocation + 12 + i]);
            }
        }
```

```
        if (idataFound) {
            // find idata physical address
            for (int i = 0; i < 4; i++) {
                idataStartLoc += pow(16, i * 2) * static_cast<unsigned
char>(rawData[idataLocation + 20 + i]);
            }
            // find idata virtual address
            for (int i = 0; i < 4; i++) {
                idataRVA += pow(16, i * 2) * static_cast<unsigned char>(rawData[idataLocation
+ 12 + i]);
            }
        }

        if (rdataFound) {
            // find rdata physical address
            for (int i = 0; i < 4; i++) {
                rdataStartLoc += pow(16, i * 2) * static_cast<unsigned
char>(rawData[rdataLocation + 20 + i]);
            }
            // find rdata virtual address
            for (int i = 0; i < 4; i++) {
                rdataRVA += pow(16, i * 2) * static_cast<unsigned char>(rawData[rdataLocation
+ 12 + i]);
```

```
            }
            // find data virtual address
        }

        if (dataFound) {
            // find data physical address
            for (int i = 0; i < 4; i++) {
                dataStartLoc += pow(16, i * 2) * static_cast<unsigned
char>(rawData[dataLocation + 20 + i]);
            }
            // find data virtual address
            for (int i = 0; i < 4; i++) {
                dataVirtualAddress += pow(16, i * 2) * static_cast<unsigned
char>(rawData[dataLocation + 12 + i]);
            }
        }

        int sectionLoc = 0, sectionRVA = 0;
        if (idataFound) {
            sectionLoc = idataStartLoc;
            sectionRVA = idataRVA;
        }
        else if (rdataFound) {
```

```
            sectionLoc = rdataStartLoc;
            sectionRVA = rdataRVA;
        }

        // find Import Directory Table and Import Address Table locations and sizes
        if (idataFound || rdataFound) {
            // find IDT location and size
            int virtualSize = 0;
            for (int i = 0; i < 4; i++) {
                virtualSize += pow(16, i * 2) * static_cast<unsigned
char>(rawData[peHeaderStartLoc + 128 + i]);
            }
            if (virtualSize > 0) {
                IDTLoc = virtualSize + sectionLoc - sectionRVA;
            }
            for (int i = 0; i < 4; i++) {
                IDTSize += pow(16, i * 2) * static_cast<unsigned
char>(rawData[peHeaderStartLoc + 132 + i]);
            }

            // find IAT location and size
            virtualSize = 0;
            for (int i = 0; i < 4; i++) {
```

```
                virtualSize += pow(16, i * 2) * static_cast<unsigned
char>(rawData[peHeaderStartLoc + 216 + i]);
            }
            if (virtualSize > 0) {
                IATLoc = virtualSize + sectionLoc - sectionRVA;
            }
        }

        // find code entry point and base of code
        for (int i = 0; i < 4; i++) {
            codeEntryPoint += pow(16, i * 2) * static_cast<unsigned
char>(rawData[peHeaderStartLoc + 40 + i]);
        }
        for (int i = 0; i < 4; i++) {
            baseOfCode += pow(16, i * 2) * static_cast<unsigned
char>(rawData[peHeaderStartLoc + 44 + i]);
        }
      }
    } catch (...) {
      codeEndLoc = 0;
    }
  }
}
```

```
void MainWindow::on_disassemblyBrowser_anchorClicked(const QUrl &arg1)
{
    // save current location in jump map
    jumpStack.push(ui->disassemblyScrollBar->value());
    // jump to new location
    ui->disassemblyScrollBar->setValue(locOffsetMap[arg1.url()]);
    refreshWindow();
}

void MainWindow::on_disassemblyJumpBackButton_clicked()
{
    if (jumpStack.size() > 0) {
        ui->disassemblyScrollBar->setValue(jumpStack.pop());
    }
}

void MainWindow::on_disassemblyStartLocationButton_clicked()
{
    // save current location in jump map
    jumpStack.push(ui->disassemblyScrollBar->value());

    ui->disassemblyScrollBar->setValue(codeStartProcedure);
```

```
}



void MainWindow::refreshHex()
{
    if (fileOpened) {
        QString offset = "", bytes = "", text = "";
        bool lastRowDisplayed = false;
        int maxDisplayRows = hexDisplayRows, maxDisplayCols = hexDisplayCols;

        // if last row will be displayed
        if (ui->hexScrollBar->value() == ui->hexScrollBar->maximum()) {
            lastRowDisplayed = true;
        }

        // for each row
        for (int displayRow = 0; displayRow < maxDisplayRows; displayRow++) {

            // offset
            int dataStart = ui->hexScrollBar->value();
            QString rowOffset = QString::number(dataStart + displayRow, 16).toUpper() += "0";
            int zeroPaddingCount = 8 - rowOffset.size();
            for (int i = 0; i < zeroPaddingCount; i++) {
```

```
    rowOffset.prepend("0");
}

// bytes
char c;
unsigned char uc;
QString rowBytes = "", rowText = "";
int displayCols = maxDisplayCols;

if (lastRowDisplayed) {
    if (displayRow == maxDisplayRows - 1) {
        if (fileSize % maxDisplayCols != 0) {
            displayCols = fileSize % maxDisplayCols;
        }
    }
}

// for each column in current row
for (int col = 0; col < displayCols; col++) {
    c = rawData[(dataStart * maxDisplayCols) + col + (displayRow * maxDisplayCols)];
    uc = static_cast<unsigned char>(c);

    // append char to decoded text
```

```
    if (uc >= 32 && uc < 127) {
        if (uc == '<') {
            rowText += "&#60;";
        }
        else if (uc == '>') {
            rowText += "&#62;";
        }
        else if (uc == ' ') {
            rowText += " ";
        }
        else {
            rowText += c;
        }
    }
    else {
        rowText += ".";
    }

    // convert char to hex byte
    rowBytes += " " + byteToHexString(uc);
}

// append row data to display
```

```
    offset += rowOffset;
    bytes += rowBytes;
    text += rowText + "<br>";
}

// remove first space
bytes.remove(0, 1);

// remove last line break
text.remove(text.size() - 4, 4);

// text colour
text.prepend("<font color='brown'>");

refreshing = true;

ui->hexOffsetDisplay->setTextColor(QColor(qBlue(255)));
ui->hexOffsetDisplay->setText(offset);
ui->hexByteDisplay->setPlainText(bytes);
ui->hexTextDisplay->setText(text);

// set cursor
QTextCursor c(ui->hexByteDisplay->textCursor());
```

```
        c.setPosition(cursorLocation);
        ui->hexByteDisplay->setTextCursor(c);

        byteDisplaySize = ui->hexByteDisplay->toPlainText().size();

        refreshing = false;
    }
}

void MainWindow::setHexValues()
{
    hexDisplayRows = 31;
    hexDisplayCols = 16;

    // font
    QFont hexFont;
    hexFont.setPixelSize(20);
    hexFont.setFamily("Courier");
    ui->hexOffsetDisplay->setFont(hexFont);
    ui->hexByteDisplay->setFont(hexFont);
    ui->hexTextDisplay->setFont(hexFont);
    ui->hexTopOffsetDisplay->setFont(hexFont);
    ui->hexTopOffsetDisplay->setTextColor(QColor(qBlue(255)));
```

```
    ui->hexTopOffsetDisplay->setText("Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D
0E 0F    Decoded Text");

    // borders
    ui->hexTopOffsetDisplay->setFrameStyle(QFrame::NoFrame);
    ui->hexOffsetDisplay->setFrameStyle(QFrame::NoFrame);
    ui->hexByteDisplay->setFrameStyle(QFrame::NoFrame);
    ui->hexTextDisplay->setFrameStyle(QFrame::NoFrame);
    ui->hexBackground->setFrameStyle(QFrame::NoFrame);

    // set editing parameters
    ui->hexByteDisplay->setOverwriteMode(true);
    ui->hexTextDisplay->setOverwriteMode(true);

    // set scroll bar maximum
    if (fileSize / (hexDisplayRows * hexDisplayCols) > 0) {
        // if last row is full with data
        if (fileSize % hexDisplayCols == 0) {
            ui->hexScrollBar->setMaximum(fileSize / hexDisplayCols - hexDisplayRows);
        }
        else {
            ui->hexScrollBar->setMaximum(fileSize / hexDisplayCols - (hexDisplayRows - 1));
        }
```

```
    }
    else {
        ui->hexScrollBar->setMaximum(0);
        if (fileSize > 0) {
            if (fileSize % hexDisplayCols == 0) {
                hexDisplayRows = fileSize / hexDisplayCols;
            }
            else {
                hexDisplayRows = fileSize / hexDisplayCols + 1;
            }
        }
        else {
            hexDisplayRows = 0;
        }
    }
}
```

```
void MainWindow::on_hexByteDisplay_cursorPositionChanged()
{
    if (!refreshing) {

        int newPosition = ui->hexByteDisplay->textCursor().position();
```

```
        // if any text is deleted
        if (newPosition < previousPosition && byteDisplaySize != ui->hexByteDisplay-
>toPlainText().size()){
            ui->hexByteDisplay->undo();
        }

        // if moving right
        if (newPosition > previousPosition) {
            moveCursor(1);
        }
        // if moving left
        else if (newPosition < previousPosition) {
            moveCursor(-1);
        }
    }
}
```

```
void MainWindow::moveCursor(int direction)
{
    QTextCursor c(ui->hexByteDisplay->textCursor());
    int newPosition = c.position();
    previousPosition = newPosition;
    if ((newPosition % 3 == 2) && newPosition < ui->hexByteDisplay->toPlainText().size()) {
```

```
        // left
        if (direction > 0) {
            previousPosition++;
        }
        // right
        else {
            previousPosition--;
        }
    }
    if (previousPosition > byteDisplaySize - 1) {
        previousPosition = byteDisplaySize - 1;
    }
    c.setPosition(previousPosition);
    refreshing = true;
    ui->hexByteDisplay->setTextCursor(c);
    refreshing = false;
}
```

```
void MainWindow::on_hexByteDisplay_textChanged()
{
    if (!refreshing) {

        int newPosition = ui->hexByteDisplay->textCursor().position();
```

```
    // if cursor postion is the same but total size is down 1
    if (newPosition == previousPosition && ui->hexByteDisplay->toPlainText().size() ==
byteDisplaySize - 1) {
        editing = true;
    }
    else if (editing) {
        int cursorPosition = 0;

        if (newPosition % 3 == 1) {
            cursorPosition = newPosition - 1;
        }
        else {
            cursorPosition = newPosition - 2;
        }

        if (newPosition == ui->hexByteDisplay->toPlainText().size() - 1) {
            cursorPosition++;
        }

        QString bytes = ui->hexByteDisplay->toPlainText();
        unsigned char c = bytes.at(cursorPosition).unicode();
        bool shouldSave = false;
```

```
// if lower case a - f, change to upper case
if (c >= 97 && c <= 102) {
    c -= 32;
    refreshing = true;
    bytes.replace(cursorPosition, 1, c);
    ui->hexByteDisplay->setPlainText(bytes);
    QTextCursor c(ui->hexByteDisplay->textCursor());
    c.setPosition(newPosition);
    ui->hexByteDisplay->setTextCursor(c);
    refreshing = false;
    shouldSave = true;
}
// if not between 0 - 9 or A - F
else if ((c < 48 || c > 57) && (c < 65 || c > 70)) {
    refreshing = true;
    ui->hexByteDisplay->undo();
    ui->hexByteDisplay->undo();
    moveCursor(-1);
    refreshing = false;
}
else {
    shouldSave = true;
```

```
        }


        if (shouldSave) {
            int byteLocation = 0, pageOffset = ui->hexScrollBar->value() * hexDisplayCols,
currentPageByte = 0, page = ui->hexScrollBar->value();


            if (cursorPosition == ui->hexByteDisplay->toPlainText().size() - 1) {
                if (page == ui->hexScrollBar->maximum()) {
                    cursorLocation = cursorPosition;
                }
                else {
                    if (secondTime) {
                        secondTime = false;
                        nextPage = true;
                        cursorLocation = cursorPosition - (hexDisplayCols * 3 - 2);
                        previousPosition = cursorLocation;
                    }
                    else {
                        secondTime = true;
                        cursorLocation = cursorPosition;
                    }
                }
                cursorPosition--;
```

```
}
else if (cursorPosition > 0) {
    // if pointing to second char of byte
    if (cursorPosition % 3 == 1) {
        cursorLocation = cursorPosition + 2;
        cursorPosition--;
    }
    else {
        cursorLocation = cursorPosition + 1;
    }
}
else {
    cursorLocation = 1;
}

QString byteHex = bytes.at(cursorPosition);
byteHex += bytes.at(cursorPosition+1);

// hex string to dec
int charDec1 = byteHex[0].unicode(), charDec2 = byteHex[1].unicode();
if (charDec1 > 57) {
    charDec1 = charDec1 - 55;
}
```

```
else {
    charDec1 = charDec1 - 48;
}
if (charDec2 > 57) {
    charDec2 = charDec2 - 55;
}
else {
    charDec2 = charDec2 - 48;
}
unsigned char byte = charDec1 * 16 + charDec2;

currentPageByte = cursorPosition / 3;
byteLocation =  currentPageByte + pageOffset;
if (!changedDataMap[byteLocation]) {
    changedDataMap[byteLocation] = true;
    originalDataMap[byteLocation] = rawData[byteLocation];
}
rawData[byteLocation] = byte;
dataChanged = true;
if (nextPage){
    ui->hexScrollBar->setValue(page + 1);
    nextPage = false;
}
```

```
            else {
                refreshHex();
            }
        }
        editing = false;
    }
    else {
        ui->hexByteDisplay->undo();
    }
  }
}
```

MainWindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>1500</width>
    <height>820</height>
```

```
 </rect>
</property>
<property name="minimumSize">
 <size>
  <width>985</width>
  <height>820</height>
 </size>
</property>
<property name="maximumSize">
 <size>
  <width>1500</width>
  <height>820</height>
 </size>
</property>
<property name="windowTitle">
 <string>MainWindow</string>
</property>
<widget class="QWidget" name="centralwidget">
 <widget class="QStackedWidget" name="MainDisplayStack">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
```

```
  <width>985</width>
  <height>770</height>
 </rect>
</property>
<property name="minimumSize">
 <size>
  <width>985</width>
  <height>770</height>
 </size>
</property>
<property name="maximumSize">
 <size>
  <width>985</width>
  <height>770</height>
 </size>
</property>
<property name="palette">
 <palette>
  <active>
   <colorrole role="WindowText">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>0</red>
```

```
   <green>0</green>
   <blue>0</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Button">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>220</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Light">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>255</blue>
  </color>
 </brush>
</colorrole>
```

```
<colorrole role="Midlight">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>237</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Dark">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>127</red>
   <green>127</green>
   <blue>110</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Mid">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>170</red>
   <green>170</green>
```

```
   <blue>147</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Text">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>0</red>
   <green>0</green>
   <blue>0</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="BrightText">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>255</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="ButtonText">
```

```
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>0</red>
   <green>0</green>
   <blue>0</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Base">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>255</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Window">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>220</blue>
```

```
  </color>
 </brush>
</colorrole>
<colorrole role="Shadow">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>0</red>
   <green>0</green>
   <blue>0</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="AlternateBase">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>237</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="ToolTipBase">
 <brush brushstyle="SolidPattern">
```

```
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>220</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="ToolTipText">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>0</red>
   <green>0</green>
   <blue>0</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="PlaceholderText">
 <brush brushstyle="SolidPattern">
  <color alpha="128">
   <red>0</red>
   <green>0</green>
   <blue>0</blue>
  </color>
```

```
 </brush>
</colorrole>
</active>
<inactive>
 <colorrole role="WindowText">
  <brush brushstyle="SolidPattern">
   <color alpha="255">
    <red>0</red>
    <green>0</green>
    <blue>0</blue>
   </color>
  </brush>
 </colorrole>
 <colorrole role="Button">
  <brush brushstyle="SolidPattern">
   <color alpha="255">
    <red>255</red>
    <green>255</green>
    <blue>220</blue>
   </color>
  </brush>
 </colorrole>
 <colorrole role="Light">
```

<brush brushstyle="SolidPattern">
<color alpha="255">
<red>255</red>
<green>255</green>
<blue>255</blue>
</color>
</brush>
</colorrole>
<colorrole role="Midlight">
<brush brushstyle="SolidPattern">
<color alpha="255">
<red>255</red>
<green>255</green>
<blue>237</blue>
</color>
</brush>
</colorrole>
<colorrole role="Dark">
<brush brushstyle="SolidPattern">
<color alpha="255">
<red>127</red>
<green>127</green>
<blue>110</blue>

```
  </color>
 </brush>
</colorrole>
<colorrole role="Mid">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>170</red>
   <green>170</green>
   <blue>147</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Text">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>0</red>
   <green>0</green>
   <blue>0</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="BrightText">
 <brush brushstyle="SolidPattern">
```

```
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>255</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="ButtonText">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>0</red>
   <green>0</green>
   <blue>0</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Base">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>255</blue>
  </color>
```

```
 </brush>
</colorrole>
<colorrole role="Window">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>220</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Shadow">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>0</red>
   <green>0</green>
   <blue>0</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="AlternateBase">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
```

<red>255</red>

<green>255</green>

<blue>237</blue>

</color>

</brush>

</colorrole>

<colorrole role="ToolTipBase">

<brush brushstyle="SolidPattern">

<color alpha="255">

<red>255</red>

<green>255</green>

<blue>220</blue>

</color>

</brush>

</colorrole>

<colorrole role="ToolTipText">

<brush brushstyle="SolidPattern">

<color alpha="255">

<red>0</red>

<green>0</green>

<blue>0</blue>

</color>

</brush>

```
 </colorrole>
 <colorrole role="PlaceholderText">
  <brush brushstyle="SolidPattern">
   <color alpha="128">
    <red>0</red>
    <green>0</green>
    <blue>0</blue>
   </color>
  </brush>
 </colorrole>
</inactive>
<disabled>
 <colorrole role="WindowText">
  <brush brushstyle="SolidPattern">
   <color alpha="255">
    <red>127</red>
    <green>127</green>
    <blue>110</blue>
   </color>
  </brush>
 </colorrole>
 <colorrole role="Button">
  <brush brushstyle="SolidPattern">
```

```
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>220</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Light">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>255</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Midlight">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>237</blue>
  </color>
```

```
 </brush>
</colorrole>
<colorrole role="Dark">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>127</red>
   <green>127</green>
   <blue>110</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Mid">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>170</red>
   <green>170</green>
   <blue>147</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Text">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
```

```
   <red>127</red>
   <green>127</green>
   <blue>110</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="BrightText">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>255</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="ButtonText">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>127</red>
   <green>127</green>
   <blue>110</blue>
  </color>
 </brush>
```

```
</colorrole>
<colorrole role="Base">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>220</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Window">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>220</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="Shadow">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>0</red>
```

```
   <green>0</green>
   <blue>0</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="AlternateBase">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>220</blue>
  </color>
 </brush>
</colorrole>
<colorrole role="ToolTipBase">
 <brush brushstyle="SolidPattern">
  <color alpha="255">
   <red>255</red>
   <green>255</green>
   <blue>220</blue>
  </color>
 </brush>
</colorrole>
```

```
   <colorrole role="ToolTipText">
    <brush brushstyle="SolidPattern">
     <color alpha="255">
      <red>0</red>
      <green>0</green>
      <blue>0</blue>
     </color>
    </brush>
   </colorrole>
   <colorrole role="PlaceholderText">
    <brush brushstyle="SolidPattern">
     <color alpha="128">
      <red>0</red>
      <green>0</green>
      <blue>0</blue>
     </color>
    </brush>
   </colorrole>
  </disabled>
 </palette>
</property>
<property name="currentIndex">
 <number>6</number>
```

```
</property>
<widget class="QWidget" name="page_0_Main_Window">
 <widget class="QTextBrowser" name="introductionBrowser">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>985</width>
    <height>753</height>
   </rect>
  </property>
  <property name="frameShape">
   <enum>QFrame::NoFrame</enum>
  </property>
  <property name="verticalScrollBarPolicy">
   <enum>Qt::ScrollBarAsNeeded</enum>
  </property>
 </widget>
</widget>
<widget class="QWidget" name="page_1_Find_Strings">
 <widget class="QLabel" name="stringCountLabel">
  <property name="geometry">
   <rect>
```

```
  <x>235</x>
  <y>725</y>
  <width>100</width>
  <height>30</height>
 </rect>
</property>
<property name="mouseTracking">
 <bool>false</bool>
</property>
<property name="text">
 <string>String Count:</string>
</property>
</widget>
<widget class="QLabel" name="stringCountValue">
 <property name="geometry">
  <rect>
   <x>335</x>
   <y>725</y>
   <width>95</width>
   <height>30</height>
  </rect>
 </property>
 <property name="text">
```

```
 <string/>
 </property>
</widget>
<widget class="QScrollBar" name="stringsScrollBar">
 <property name="geometry">
  <rect>
   <x>945</x>
   <y>10</y>
   <width>30</width>
   <height>700</height>
  </rect>
 </property>
 <property name="minimumSize">
  <size>
   <width>30</width>
   <height>700</height>
  </size>
 </property>
 <property name="maximumSize">
  <size>
   <width>30</width>
   <height>700</height>
  </size>
```

```
 </property>
 <property name="maximum">
  <number>0</number>
 </property>
 <property name="pageStep">
  <number>16</number>
 </property>
 <property name="orientation">
  <enum>Qt::Vertical</enum>
 </property>
</widget>
<widget class="QListWidget" name="stringList">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>10</y>
   <width>925</width>
   <height>700</height>
  </rect>
 </property>
 <property name="minimumSize">
  <size>
   <width>925</width>
```

```
  <height>700</height>
 </size>
</property>
<property name="maximumSize">
 <size>
  <width>925</width>
  <height>700</height>
 </size>
</property>
<property name="contextMenuPolicy">
 <enum>Qt::DefaultContextMenu</enum>
</property>
<property name="verticalScrollBarPolicy">
 <enum>Qt::ScrollBarAlwaysOff</enum>
</property>
<property name="horizontalScrollBarPolicy">
 <enum>Qt::ScrollBarAlwaysOff</enum>
</property>
<property name="selectionMode">
 <enum>QAbstractItemView::ExtendedSelection</enum>
</property>
</widget>
<widget class="QLineEdit" name="searchString">
```

```
 <property name="geometry">
  <rect>
   <x>630</x>
   <y>725</y>
   <width>265</width>
   <height>30</height>
  </rect>
 </property>
</widget>
<widget class="QPushButton" name="stringSearchButton">
 <property name="geometry">
  <rect>
   <x>900</x>
   <y>725</y>
   <width>70</width>
   <height>30</height>
  </rect>
 </property>
 <property name="text">
  <string>search</string>
 </property>
</widget>
<widget class="QLabel" name="stringsPageNumberLabel">
```

```
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>725</y>
   <width>45</width>
   <height>30</height>
  </rect>
 </property>
 <property name="text">
  <string>Page:</string>
 </property>
</widget>
<widget class="QLabel" name="stringsPageNumberValue">
 <property name="geometry">
  <rect>
   <x>55</x>
   <y>725</y>
   <width>180</width>
   <height>30</height>
  </rect>
 </property>
 <property name="text">
  <string/>
```

```
 </property>
</widget>
 <widget class="QCheckBox" name="stringsSearchFromBeginningCheckBox">
  <property name="geometry">
   <rect>
    <x>430</x>
    <y>725</y>
    <width>190</width>
    <height>30</height>
   </rect>
  </property>
  <property name="text">
   <string>search from beginning</string>
  </property>
 </widget>
</widget>
<widget class="QWidget" name="page_2_Saved_Strings">
 <widget class="QListWidget" name="savedStringList">
  <property name="geometry">
   <rect>
    <x>10</x>
    <y>10</y>
    <width>960</width>
```

```
  <height>700</height>
 </rect>
</property>
<property name="sizePolicy">
 <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
  <horstretch>0</horstretch>
  <verstretch>0</verstretch>
 </sizepolicy>
</property>
<property name="minimumSize">
 <size>
  <width>960</width>
  <height>700</height>
 </size>
</property>
<property name="maximumSize">
 <size>
  <width>960</width>
  <height>700</height>
 </size>
</property>
<property name="focusPolicy">
 <enum>Qt::NoFocus</enum>
```

```
</property>
<property name="verticalScrollBarPolicy">
 <enum>Qt::ScrollBarAsNeeded</enum>
</property>
<property name="horizontalScrollBarPolicy">
 <enum>Qt::ScrollBarAsNeeded</enum>
</property>
<property name="autoScroll">
 <bool>true</bool>
</property>
<property name="autoScrollMargin">
 <number>16</number>
</property>
<property name="editTriggers">
 <set>QAbstractItemView::DoubleClicked|QAbstractItemView::EditKeyPressed</set>
</property>
<property name="defaultDropAction">
 <enum>Qt::CopyAction</enum>
</property>
<property name="selectionMode">
 <enum>QAbstractItemView::ExtendedSelection</enum>
</property>
</widget>
```

```
<widget class="QLabel" name="savedStringCountValue">
 <property name="geometry">
  <rect>
   <x>160</x>
   <y>725</y>
   <width>95</width>
   <height>30</height>
  </rect>
 </property>
 <property name="text">
  <string/>
 </property>
</widget>
<widget class="QLineEdit" name="searchSavedString">
 <property name="geometry">
  <rect>
   <x>595</x>
   <y>725</y>
   <width>300</width>
   <height>30</height>
  </rect>
 </property>
</widget>
```

```
<widget class="QPushButton" name="savedStringSearchButton">
 <property name="geometry">
  <rect>
   <x>900</x>
   <y>725</y>
   <width>70</width>
   <height>30</height>
  </rect>
 </property>
 <property name="text">
  <string>search</string>
 </property>
</widget>
<widget class="QLabel" name="savedStringCountLabel">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>725</y>
   <width>150</width>
   <height>30</height>
  </rect>
 </property>
 <property name="mouseTracking">
```

```
  <bool>false</bool>
 </property>
 <property name="text">
  <string>Saved String Count:</string>
 </property>
</widget>
<widget class="QCheckBox" name="savedStringsSearchFromBeginningCheckBox">
 <property name="geometry">
  <rect>
   <x>385</x>
   <y>725</y>
   <width>190</width>
   <height>30</height>
  </rect>
 </property>
 <property name="text">
  <string>search from beginning</string>
 </property>
</widget>
</widget>
<widget class="QWidget" name="page_3_DLLs">
 <widget class="QTextBrowser" name="DLLNameBrowser">
  <property name="geometry">
```

<rect>

<x>10</x>

<y>50</y>

<width>965</width>

<height>180</height>

</rect>

</property>

<property name="html">

<string><!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-html40/strict.dtd">

<html><head><meta name="qrichtext" content="1" /><style type="text/css">

p, li { white-space: pre-wrap; }

</style></head><body style=" font-family:'MS Shell Dlg 2'; font-size:8pt; font-weight:400; font-style:normal;">

<p style="-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;">
</p></body></html></string>

</property>

</widget>

<widget class="QTextBrowser" name="DLLFunctionNameBrowser">

<property name="geometry">

<rect>

<x>10</x>

```
        <y>280</y>
        <width>965</width>
        <height>470</height>
       </rect>
      </property>
      <property name="html">
       <string>&lt;!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.0//EN&quot; &quot;http://www.w3.org/TR/REC-html40/strict.dtd&quot;&gt;
&lt;html&gt;&lt;head&gt;&lt;meta name=&quot;qrichtext&quot; content=&quot;1&quot; /&gt;&lt;style type=&quot;text/css&quot;&gt;
p, li { white-space: pre-wrap; }
&lt;/style&gt;&lt;/head&gt;&lt;body style=&quot; font-family:'MS Shell Dlg 2'; font-size:8pt; font-weight:400; font-style:normal;&quot;&gt;
&lt;p style=&quot;-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;&quot;&gt;&lt;br /&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
      </property>
     </widget>
     <widget class="QTextBrowser" name="DLLTitleBrowser">
      <property name="geometry">
       <rect>
        <x>10</x>
        <y>10</y>
        <width>965</width>
```

```
  <height>40</height>
 </rect>
</property>
<property name="autoFillBackground">
 <bool>false</bool>
</property>
<property name="frameShape">
 <enum>QFrame::StyledPanel</enum>
</property>
<property name="frameShadow">
 <enum>QFrame::Sunken</enum>
</property>
<property name="lineWidth">
 <number>1</number>
</property>
<property name="midLineWidth">
 <number>0</number>
</property>
<property name="verticalScrollBarPolicy">
 <enum>Qt::ScrollBarAlwaysOff</enum>
</property>
<property name="horizontalScrollBarPolicy">
 <enum>Qt::ScrollBarAlwaysOff</enum>
```

```
    </property>
    <property name="readOnly">
     <bool>true</bool>
    </property>
    <property name="html">
     <string>&lt;!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.0//EN&quot;
&quot;http://www.w3.org/TR/REC-html40/strict.dtd&quot;&gt;
&lt;html&gt;&lt;head&gt;&lt;meta name=&quot;qrichtext&quot; content=&quot;1&quot;
/&gt;&lt;style type=&quot;text/css&quot;&gt;
p, li { white-space: pre-wrap; }
&lt;/style&gt;&lt;/head&gt;&lt;body style=&quot; font-family:'MS Shell Dlg 2'; font-size:8pt;
font-weight:400; font-style:normal;&quot;&gt;
&lt;p style=&quot;-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-
left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;&quot;&gt;&lt;br
/&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
    </property>
   </widget>
   <widget class="QTextBrowser" name="DLLFunctionTitleBrowser">
    <property name="geometry">
     <rect>
      <x>10</x>
      <y>240</y>
      <width>965</width>
      <height>40</height>
```

```
 </rect>
</property>
<property name="frameShape">
 <enum>QFrame::StyledPanel</enum>
</property>
<property name="frameShadow">
 <enum>QFrame::Sunken</enum>
</property>
<property name="lineWidth">
 <number>1</number>
</property>
<property name="midLineWidth">
 <number>0</number>
</property>
<property name="verticalScrollBarPolicy">
 <enum>Qt::ScrollBarAlwaysOff</enum>
</property>
<property name="horizontalScrollBarPolicy">
 <enum>Qt::ScrollBarAlwaysOff</enum>
</property>
<property name="html">
 <string>&lt;!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.0//EN&quot;
&quot;http://www.w3.org/TR/REC-html40/strict.dtd&quot;&gt;
```

<html><head><meta name="qrichtext" content="1" /><style type="text/css">

p, li { white-space: pre-wrap; }

</style></head><body style=" font-family:'MS Shell Dlg 2'; font-size:8pt; font-weight:400; font-style:normal;">

<p style="-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;">
</p></body></html></string>

</property>

</widget>

</widget>

<widget class="QWidget" name="page_4_Hex">

<widget class="QScrollBar" name="hexScrollBar">

<property name="geometry">

<rect>

<x>945</x>

<y>15</y>

<width>25</width>

<height>735</height>

</rect>

</property>

<property name="minimumSize">

<size>

<width>25</width>

```
  <height>700</height>
 </size>
</property>
<property name="maximumSize">
 <size>
  <width>25</width>
  <height>735</height>
 </size>
</property>
<property name="maximum">
 <number>0</number>
</property>
<property name="pageStep">
 <number>16</number>
</property>
<property name="orientation">
 <enum>Qt::Vertical</enum>
</property>
</widget>
<widget class="QTextBrowser" name="hexOffsetDisplay">
 <property name="geometry">
  <rect>
   <x>15</x>
```

```
        <y>50</y>
        <width>110</width>
        <height>700</height>
       </rect>
      </property>
      <property name="verticalScrollBarPolicy">
       <enum>Qt::ScrollBarAlwaysOff</enum>
      </property>
      <property name="horizontalScrollBarPolicy">
       <enum>Qt::ScrollBarAlwaysOff</enum>
      </property>
      <property name="html">
       <string>&lt;!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.0//EN&quot;
&quot;http://www.w3.org/TR/REC-html40/strict.dtd&quot;&gt;
&lt;html&gt;&lt;head&gt;&lt;meta name=&quot;qrichtext&quot; content=&quot;1&quot;
/&gt;&lt;style type=&quot;text/css&quot;&gt;
p, li { white-space: pre-wrap; }
&lt;/style&gt;&lt;/head&gt;&lt;body style=&quot; font-family:'MS Shell Dlg 2'; font-size:8pt;
font-weight:400; font-style:normal;&quot;&gt;
&lt;p style=&quot;-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-
left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;&quot;&gt;&lt;br
/&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
      </property>
     </widget>
```

```
    <widget class="QTextBrowser" name="hexTopOffsetDisplay">
     <property name="geometry">
      <rect>
       <x>15</x>
       <y>15</y>
       <width>920</width>
       <height>30</height>
      </rect>
     </property>
     <property name="verticalScrollBarPolicy">
      <enum>Qt::ScrollBarAlwaysOff</enum>
     </property>
     <property name="horizontalScrollBarPolicy">
      <enum>Qt::ScrollBarAlwaysOff</enum>
     </property>
     <property name="html">
      <string>&lt;!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.0//EN&quot;
&quot;http://www.w3.org/TR/REC-html40/strict.dtd&quot;&gt;
&lt;html&gt;&lt;head&gt;&lt;meta name=&quot;qrichtext&quot; content=&quot;1&quot;
/&gt;&lt;style type=&quot;text/css&quot;&gt;
p, li { white-space: pre-wrap; }
&lt;/style&gt;&lt;/head&gt;&lt;body style=&quot; font-family:'MS Shell Dlg 2'; font-size:8pt;
font-weight:400; font-style:normal;&quot;&gt;
```

<p style=" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;">Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded Text</p></body></html></string>

</property>

</widget>

<widget class="QPlainTextEdit" name="hexByteDisplay">

<property name="geometry">

<rect>

<x>135</x>

<y>50</y>

<width>580</width>

<height>700</height>

</rect>

</property>

<property name="verticalScrollBarPolicy">

<enum>Qt::ScrollBarAlwaysOff</enum>

</property>

<property name="horizontalScrollBarPolicy">

<enum>Qt::ScrollBarAlwaysOff</enum>

</property>

</widget>

<widget class="QTextBrowser" name="hexTextDisplay">

<property name="geometry">
<rect>
<x>725</x>
<y>50</y>
<width>210</width>
<height>700</height>
</rect>
</property>
<property name="verticalScrollBarPolicy">
<enum>Qt::ScrollBarAlwaysOff</enum>
</property>
<property name="horizontalScrollBarPolicy">
<enum>Qt::ScrollBarAlwaysOff</enum>
</property>
<property name="html">
<string><!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-html40/strict.dtd">

<html><head><meta name="qrichtext" content="1" /><style type="text/css">

p, li { white-space: pre-wrap; }

</style></head><body style=" font-family:'MS Shell Dlg 2'; font-size:8pt; font-weight:400; font-style:normal;">

<p style="-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;">
</p></body></html></string>
</property>
</widget>
<widget class="QListWidget" name="hexBackground">
<property name="geometry">
<rect>
<x>10</x>
<y>10</y>
<width>965</width>
<height>745</height>
</rect>
</property>
</widget>
<zorder>hexBackground</zorder>
<zorder>hexScrollBar</zorder>
<zorder>hexOffsetDisplay</zorder>
<zorder>hexTopOffsetDisplay</zorder>
<zorder>hexByteDisplay</zorder>
<zorder>hexTextDisplay</zorder>
</widget>
<widget class="QWidget" name="page_5_Entropy">

```
<widget class="QScrollArea" name="scrollArea">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>10</y>
   <width>965</width>
   <height>745</height>
  </rect>
 </property>
 <property name="widgetResizable">
  <bool>true</bool>
 </property>
 <widget class="QWidget" name="scrollAreaWidgetContents">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>963</width>
    <height>743</height>
   </rect>
  </property>
 </widget>
</widget>
```

```
</widget>
<widget class="QWidget" name="page_6_Disassembly">
 <widget class="QTextBrowser" name="disassemblyBrowser">
  <property name="geometry">
   <rect>
    <x>10</x>
    <y>10</y>
    <width>925</width>
    <height>700</height>
   </rect>
  </property>
  <property name="minimumSize">
   <size>
    <width>925</width>
    <height>700</height>
   </size>
  </property>
  <property name="maximumSize">
   <size>
    <width>925</width>
    <height>700</height>
   </size>
  </property>
```

```
 <property name="frameShape">
  <enum>QFrame::StyledPanel</enum>
 </property>
 <property name="verticalScrollBarPolicy">
  <enum>Qt::ScrollBarAlwaysOff</enum>
 </property>
 <property name="horizontalScrollBarPolicy">
  <enum>Qt::ScrollBarAlwaysOff</enum>
 </property>
 <property name="undoRedoEnabled">
  <bool>true</bool>
 </property>
</widget>
<widget class="QScrollBar" name="disassemblyScrollBar">
 <property name="geometry">
  <rect>
   <x>945</x>
   <y>10</y>
   <width>30</width>
   <height>700</height>
  </rect>
 </property>
 <property name="minimumSize">
```

```
 <size>
  <width>30</width>
  <height>700</height>
 </size>
</property>
<property name="maximumSize">
 <size>
  <width>30</width>
  <height>700</height>
 </size>
</property>
<property name="maximum">
 <number>0</number>
</property>
<property name="pageStep">
 <number>16</number>
</property>
<property name="orientation">
 <enum>Qt::Vertical</enum>
</property>
</widget>
<widget class="QLabel" name="disassemblyPageNumberLabel">
 <property name="geometry">
```

```
 <rect>
  <x>10</x>
  <y>725</y>
  <width>45</width>
  <height>30</height>
 </rect>
</property>
<property name="text">
 <string>Line: </string>
</property>
</widget>
<widget class="QLabel" name="disassemblyPageNumberValue">
 <property name="geometry">
  <rect>
   <x>55</x>
   <y>725</y>
   <width>200</width>
   <height>30</height>
  </rect>
 </property>
 <property name="text">
  <string/>
 </property>
```

```
</widget>
<widget class="QPushButton" name="disassemblyStartLocationButton">
 <property name="geometry">
  <rect>
   <x>255</x>
   <y>725</y>
   <width>112</width>
   <height>30</height>
  </rect>
 </property>
 <property name="text">
  <string>jump to start</string>
 </property>
</widget>
<widget class="QPushButton" name="disassemblySearchButton">
 <property name="geometry">
  <rect>
   <x>900</x>
   <y>725</y>
   <width>70</width>
   <height>30</height>
  </rect>
 </property>
```

```
 <property name="text">
  <string>search</string>
 </property>
</widget>
<widget class="QLineEdit" name="disassemblySearchString">
 <property name="geometry">
  <rect>
   <x>664</x>
   <y>725</y>
   <width>231</width>
   <height>30</height>
  </rect>
 </property>
</widget>
<widget class="QCheckBox" name="disassemblySearchFromBeginningCheckBox">
 <property name="geometry">
  <rect>
   <x>460</x>
   <y>725</y>
   <width>190</width>
   <height>30</height>
  </rect>
 </property>
```

```
      <property name="text">
       <string>search from beginning</string>
      </property>
     </widget>
     <widget class="QPushButton" name="disassemblyJumpBackButton">
      <property name="geometry">
       <rect>
        <x>385</x>
        <y>720</y>
        <width>60</width>
        <height>40</height>
       </rect>
      </property>
      <property name="text">
       <string>jump
back</string>
      </property>
     </widget>
    </widget>
   </widget>
   <widget class="QTabWidget" name="ChecklistTab">
    <property name="geometry">
     <rect>
```

```
  <x>985</x>
  <y>0</y>
  <width>515</width>
  <height>755</height>
 </rect>
</property>
<property name="sizePolicy">
 <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
  <horstretch>0</horstretch>
  <verstretch>0</verstretch>
 </sizepolicy>
</property>
<property name="minimumSize">
 <size>
  <width>515</width>
  <height>755</height>
 </size>
</property>
<property name="maximumSize">
 <size>
  <width>515</width>
  <height>755</height>
 </size>
```

```
</property>
<property name="focusPolicy">
 <enum>Qt::TabFocus</enum>
</property>
<property name="tabPosition">
 <enum>QTabWidget::North</enum>
</property>
<property name="tabShape">
 <enum>QTabWidget::Triangular</enum>
</property>
<property name="currentIndex">
 <number>0</number>
</property>
<property name="elideMode">
 <enum>Qt::ElideNone</enum>
</property>
<property name="documentMode">
 <bool>false</bool>
</property>
<widget class="QWidget" name="File">
 <property name="minimumSize">
  <size>
   <width>515</width>
```

```
  <height>770</height>
 </size>
</property>
<property name="maximumSize">
 <size>
  <width>515</width>
  <height>770</height>
 </size>
</property>
<attribute name="title">
 <string>File</string>
</attribute>
<widget class="QSplitter" name="fileSplitter">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>509</width>
   <height>730</height>
  </rect>
 </property>
 <property name="sizePolicy">
  <sizepolicy hsizetype="Preferred" vsizetype="Expanding">
```

```
  <horstretch>0</horstretch>
  <verstretch>0</verstretch>
 </sizepolicy>
</property>
<property name="minimumSize">
 <size>
  <width>509</width>
  <height>730</height>
 </size>
</property>
<property name="maximumSize">
 <size>
  <width>509</width>
  <height>1515</height>
 </size>
</property>
<property name="orientation">
 <enum>Qt::Vertical</enum>
</property>
<widget class="QTextBrowser" name="checklistFileBrowser">
 <property name="minimumSize">
  <size>
   <width>509</width>
```

```
  <height>0</height>
 </size>
</property>
<property name="maximumSize">
 <size>
  <width>509</width>
  <height>755</height>
 </size>
</property>
<property name="frameShape">
 <enum>QFrame::StyledPanel</enum>
</property>
<property name="html">
 <string>&lt;!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.0//EN&quot;
&quot;http://www.w3.org/TR/REC-html40/strict.dtd&quot;&gt;

&lt;html&gt;&lt;head&gt;&lt;meta name=&quot;qrichtext&quot; content=&quot;1&quot;
/&gt;&lt;style type=&quot;text/css&quot;&gt;

p, li { white-space: pre-wrap; }

&lt;/style&gt;&lt;/head&gt;&lt;body style=&quot; font-family:'MS Shell Dlg 2'; font-size:8pt;
font-weight:400; font-style:normal;&quot;&gt;

&lt;p style=&quot;-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-
left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;&quot;&gt;&lt;br
/&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
 </property>
```

```
  </widget>
  <widget class="QTextEdit" name="fileNotes">
   <property name="minimumSize">
    <size>
     <width>509</width>
     <height>0</height>
    </size>
   </property>
   <property name="maximumSize">
    <size>
     <width>509</width>
     <height>755</height>
    </size>
   </property>
  </widget>
 </widget>
</widget>
<widget class="QWidget" name="Packers">
 <property name="minimumSize">
  <size>
   <width>515</width>
   <height>770</height>
  </size>
```

```
</property>
<property name="maximumSize">
 <size>
  <width>515</width>
  <height>770</height>
 </size>
</property>
<attribute name="title">
 <string>Packers</string>
</attribute>
<widget class="QSplitter" name="packersSplitter">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>509</width>
   <height>730</height>
  </rect>
 </property>
 <property name="minimumSize">
  <size>
   <width>509</width>
   <height>730</height>
```

```
 </size>
</property>
<property name="orientation">
 <enum>Qt::Vertical</enum>
</property>
<widget class="QTextBrowser" name="checklistPackersBrowser">
 <property name="minimumSize">
  <size>
   <width>509</width>
   <height>0</height>
  </size>
 </property>
 <property name="maximumSize">
  <size>
   <width>509</width>
   <height>755</height>
  </size>
 </property>
</widget>
<widget class="QTextEdit" name="packersNotes">
 <property name="minimumSize">
  <size>
   <width>509</width>
```

```
    <height>0</height>
   </size>
  </property>
  <property name="maximumSize">
   <size>
    <width>509</width>
    <height>755</height>
   </size>
  </property>
 </widget>
</widget>
<widget class="QWidget" name="Strings">
 <property name="minimumSize">
  <size>
   <width>515</width>
   <height>770</height>
  </size>
 </property>
 <property name="maximumSize">
  <size>
   <width>515</width>
   <height>770</height>
```

```
 </size>
</property>
<attribute name="title">
 <string>Strings</string>
</attribute>
<widget class="QSplitter" name="stringsSplitter">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>509</width>
   <height>730</height>
  </rect>
 </property>
 <property name="minimumSize">
  <size>
   <width>509</width>
   <height>730</height>
  </size>
 </property>
 <property name="orientation">
  <enum>Qt::Vertical</enum>
 </property>
```

```
<widget class="QTextBrowser" name="checklistStringsBrowser">
 <property name="minimumSize">
  <size>
   <width>509</width>
   <height>0</height>
  </size>
 </property>
 <property name="maximumSize">
  <size>
   <width>509</width>
   <height>755</height>
  </size>
 </property>
</widget>
<widget class="QTextEdit" name="stringsNotes">
 <property name="minimumSize">
  <size>
   <width>509</width>
   <height>0</height>
  </size>
 </property>
 <property name="maximumSize">
  <size>
```

```
    <width>509</width>
    <height>755</height>
   </size>
  </property>
 </widget>
</widget>
<widget class="QWidget" name="DLLs">
 <property name="minimumSize">
  <size>
   <width>515</width>
   <height>770</height>
  </size>
 </property>
 <property name="maximumSize">
  <size>
   <width>515</width>
   <height>770</height>
  </size>
 </property>
 <attribute name="title">
  <string>DLL's</string>
 </attribute>
```

```
<widget class="QSplitter" name="DLLsSplitter">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>509</width>
   <height>730</height>
  </rect>
 </property>
 <property name="minimumSize">
  <size>
   <width>509</width>
   <height>730</height>
  </size>
 </property>
 <property name="orientation">
  <enum>Qt::Vertical</enum>
 </property>
 <widget class="QTextBrowser" name="checklistDLLsBrowser">
  <property name="minimumSize">
   <size>
    <width>509</width>
    <height>0</height>
```

```
 </size>
</property>
<property name="maximumSize">
 <size>
  <width>509</width>
  <height>755</height>
 </size>
</property>
</widget>
<widget class="QTextEdit" name="DLLsNotes">
 <property name="minimumSize">
  <size>
   <width>509</width>
   <height>0</height>
  </size>
 </property>
 <property name="maximumSize">
  <size>
   <width>509</width>
   <height>755</height>
  </size>
 </property>
</widget>
```

```
 </widget>
</widget>
<widget class="QWidget" name="Disassembly">
 <property name="minimumSize">
  <size>
   <width>515</width>
   <height>770</height>
  </size>
 </property>
 <property name="maximumSize">
  <size>
   <width>515</width>
   <height>770</height>
  </size>
 </property>
 <attribute name="title">
  <string>Disassembly</string>
 </attribute>
 <widget class="QSplitter" name="disassemblySplitter">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
```

```
  <width>509</width>
  <height>730</height>
 </rect>
</property>
<property name="minimumSize">
 <size>
  <width>509</width>
  <height>730</height>
 </size>
</property>
<property name="orientation">
 <enum>Qt::Vertical</enum>
</property>
<widget class="QTextBrowser" name="checklistDisassemblyBrowser">
 <property name="minimumSize">
  <size>
   <width>509</width>
   <height>0</height>
  </size>
 </property>
 <property name="maximumSize">
  <size>
   <width>509</width>
```

```
    <height>755</height>
   </size>
  </property>
 </widget>
 <widget class="QTextEdit" name="disassemblyNotes">
  <property name="minimumSize">
   <size>
    <width>509</width>
    <height>0</height>
   </size>
  </property>
  <property name="maximumSize">
   <size>
    <width>509</width>
    <height>755</height>
   </size>
  </property>
 </widget>
</widget>
<widget class="QWidget" name="Hex">
 <property name="minimumSize">
  <size>
```

```
  <width>515</width>
  <height>770</height>
 </size>
</property>
<property name="maximumSize">
 <size>
  <width>515</width>
  <height>770</height>
 </size>
</property>
<attribute name="title">
 <string>Hex</string>
</attribute>
<widget class="QSplitter" name="hexSplitter">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>509</width>
   <height>730</height>
  </rect>
 </property>
 <property name="minimumSize">
```

```
 <size>
  <width>509</width>
  <height>730</height>
 </size>
</property>
<property name="orientation">
 <enum>Qt::Vertical</enum>
</property>
<widget class="QTextBrowser" name="checklistHexBrowser">
 <property name="minimumSize">
  <size>
   <width>509</width>
   <height>0</height>
  </size>
 </property>
 <property name="maximumSize">
  <size>
   <width>509</width>
   <height>755</height>
  </size>
 </property>
</widget>
<widget class="QTextEdit" name="hexNotes">
```

```
    <property name="minimumSize">
     <size>
      <width>509</width>
      <height>0</height>
     </size>
    </property>
    <property name="maximumSize">
     <size>
      <width>509</width>
      <height>755</height>
     </size>
    </property>
   </widget>
  </widget>
 </widget>
</widget>
<widget class="QStatusBar" name="statusbar"/>
<widget class="QMenuBar" name="menubar">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
```

<width>1500</width>

<height>31</height>

</rect>

</property>

<widget class="QMenu" name="menuFile">

<property name="title">

<string>File</string>

</property>

<addaction name="actionOpen"/>

<addaction name="actionSave"/>

<addaction name="actionUndo_All_Changes"/>

<addaction name="separator"/>

<addaction name="actionGenerate_Hash"/>

<addaction name="actionCreate_Backup"/>

<addaction name="separator"/>

<addaction name="actionExit"/>

</widget>

<widget class="QMenu" name="menuPackers">

<property name="title">

<string>Packers</string>

</property>

<addaction name="actionCheck_if_Packed"/>

<addaction name="actionEntropy_Graph"/>

```
 <addaction name="separator"/>
 <addaction name="actionPack"/>
 <addaction name="actionUnpack"/>
</widget>
<widget class="QMenu" name="menuStrings">
 <property name="title">
  <string>Strings</string>
 </property>
 <addaction name="actionFind_Strings"/>
 <addaction name="actionSaved_Strings"/>
</widget>
<widget class="QMenu" name="menuChecklist">
 <property name="title">
  <string>Checklist</string>
 </property>
 <addaction name="actionChecklistMain"/>
</widget>
<widget class="QMenu" name="menuAdvanced">
 <property name="title">
  <string>Advanced</string>
 </property>
 <addaction name="actionDLL_s"/>
 <addaction name="actionHex"/>
```

```
  <addaction name="actionDisassembly"/>
 </widget>
 <addaction name="menuFile"/>
 <addaction name="menuPackers"/>
 <addaction name="menuStrings"/>
 <addaction name="menuAdvanced"/>
 <addaction name="menuChecklist"/>
</widget>
<action name="actionOpen">
 <property name="text">
  <string>Open</string>
 </property>
</action>
<action name="actionGenerate_Hash">
 <property name="text">
  <string>Generate Hash</string>
 </property>
</action>
<action name="actionCreate_Backup">
 <property name="text">
  <string>Create Backup</string>
 </property>
</action>
```

```
<action name="actionExit">
 <property name="text">
  <string>Exit</string>
 </property>
</action>
<action name="actionCheck_if_Packed">
 <property name="text">
  <string>Check if UPX Packed</string>
 </property>
</action>
<action name="actionPack">
 <property name="text">
  <string>Pack</string>
 </property>
</action>
<action name="actionUnpack">
 <property name="text">
  <string>Unpack</string>
 </property>
</action>
<action name="actionFind_Strings">
 <property name="text">
  <string>Find Strings</string>
```

```
 </property>
</action>
<action name="action4_or_more_characters">
 <property name="text">
  <string>4 or more characters</string>
 </property>
</action>
<action name="actionSaved_Strings">
 <property name="text">
  <string>Saved Strings</string>
 </property>
 <property name="iconVisibleInMenu">
  <bool>true</bool>
 </property>
</action>
<action name="actionDLL_s">
 <property name="text">
  <string>DLL's</string>
 </property>
</action>
<action name="actionHex">
 <property name="text">
  <string>Hex</string>
```

```
 </property>
</action>
<action name="actionChecklistMain">
 <property name="text">
  <string>Show / Hide</string>
 </property>
</action>
<action name="actionSeperate_Window">
 <property name="text">
  <string>Seperate Window</string>
 </property>
</action>
<action name="actionSave">
 <property name="text">
  <string>Save</string>
 </property>
</action>
<action name="actionUndo_All_Changes">
 <property name="text">
  <string>Undo Any Unsaved Changes</string>
 </property>
 <property name="toolTip">
  <string>Undo Any Unsaved Changes</string>
```

```
  </property>
 </action>
 <action name="actionEntropy_Graph">
  <property name="text">
   <string>Entropy Graph</string>
  </property>
 </action>
 <action name="actionDisassembly">
  <property name="text">
   <string>Disassembly</string>
  </property>
 </action>
</widget>
<resources/>
<connections/>
</ui>
```

CustomDialog.h

```
#ifndef DIALOGBOX_H
#define DIALOGBOX_H

#include <QDialog>
```

```
namespace Ui {
class DialogBox;
}

class CustomDialog : public QDialog
{
    Q_OBJECT

public:
    explicit CustomDialog(QWidget *parent = nullptr);
    ~CustomDialog();
    void setText(QString text);

private slots:
    void on_pushButton_clicked();

private:
    Ui::DialogBox *ui;
};

#endif // DIALOGBOX_H
```

CustomDialog.cpp

```
#include "customdialog.h"

#include "ui_customdialog.h"


CustomDialog::CustomDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::DialogBox)
{
    ui->setupUi(this);
}


CustomDialog::~CustomDialog()
{
    delete ui;
}


void CustomDialog::setText(QString text)
{
    ui->dialogBrowser->setHtml(text);
}


void CustomDialog::on_pushButton_clicked()
{
    this->close();
```

```
}
```

CustomDialog.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>DialogBox</class>
 <widget class="QDialog" name="DialogBox">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>480</width>
    <height>225</height>
   </rect>
  </property>
  <property name="minimumSize">
   <size>
    <width>480</width>
    <height>225</height>
   </size>
  </property>
  <property name="maximumSize">
   <size>
```

```
  <width>480</width>
  <height>225</height>
 </size>
</property>
<property name="windowTitle">
 <string>Dialog</string>
</property>
<widget class="QPushButton" name="pushButton">
 <property name="geometry">
  <rect>
   <x>185</x>
   <y>180</y>
   <width>110</width>
   <height>35</height>
  </rect>
 </property>
 <property name="text">
  <string>OK</string>
 </property>
</widget>
<widget class="QTextBrowser" name="dialogBrowser">
 <property name="geometry">
  <rect>
```

```
    <x>0</x>
    <y>0</y>
    <width>480</width>
    <height>225</height>
   </rect>
  </property>
  <property name="frameShape">
   <enum>QFrame::NoFrame</enum>
  </property>
  <property name="html">
   <string>&lt;!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.0//EN&quot;
&quot;http://www.w3.org/TR/REC-html40/strict.dtd&quot;&gt;
&lt;html&gt;&lt;head&gt;&lt;meta name=&quot;qrichtext&quot; content=&quot;1&quot;
/&gt;&lt;style type=&quot;text/css&quot;&gt;
p, li { white-space: pre-wrap; }
&lt;/style&gt;&lt;/head&gt;&lt;body style=&quot; font-family:'MS Shell Dlg 2'; font-size:8pt;
font-weight:400; font-style:normal;&quot;&gt;
&lt;p style=&quot;-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-
left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;&quot;&gt;&lt;br
/&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
  </property>
 </widget>
 <zorder>dialogBrowser</zorder>
 <zorder>pushButton</zorder>
```

```
 </widget>
 <resources/>
 <connections/>
</ui>
```