ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программы с комментариями

MainWindowUnit.h

//---------------------------------------------------------------------------

#ifndef MainWindowUnitH

#define MainWindowUnitH

//---------------------------------------------------------------------------

#include <System.Classes.hpp>

#include <Vcl.Controls.hpp>

#include <Vcl.StdCtrls.hpp>

#include <Vcl.Forms.hpp>

#include <Vcl.ExtCtrls.hpp>

#include <Vcl.Menus.hpp>

#include <SHDocVw.hpp>

#include <Vcl.ComCtrls.hpp>

#include <Vcl.OleCtrls.hpp>

#include "SHDocVw\_OCX.h"

#include <Vcl.Dialogs.hpp>

#include <Vcl.Buttons.hpp>

#include <Vcl.ToolWin.hpp>

#include <Vcl.ActnList.hpp>

#include <Vcl.ImgList.hpp>

#include "frxCtrls.hpp"

#include <Vcl.Graphics.hpp>

//---------------------------------------------------------------------------

#include <mshtml.h>

#include <VCLTee.TeCanvas.hpp>

#include <Vcl.StdActns.hpp>

#include <vector>

#include "Style.h"

#include "StylesCollection.h"

#include "DocumentFormUnit.h"

//-------------------------------------------------------------------------

class TForm1 : public TForm

{

\_\_published: // IDE-managed Components

TMainMenu \*MainMenu1;

TMenuItem \*N1;

TMenuItem \*N2;

TMenuItem \*NClose;

TMenuItem \*NOpenDocument;

TMenuItem \*AboutAction;

TMenuItem \*NSaveDocument;

TMenuItem \*N3;

TMenuItem \*N4;

TMenuItem \*N5;

TMenuItem \*NCloseDocument;

TMenuItem \*N7;

TActionList \*ActionList1;

TAction \*acOpenFile;

TAction \*acSaveFile;

TAction \*acExit;

TAction \*acNewPage;

TAction \*acSaveFileAs;

TAction \*acShowDom;

TAction \*acUnderline;

TAction \*acBold;

TAction \*acItalics;

TAction \*acInsertImage;

TAction \*acInsertHyperlink;

TAction \*acInsertList;

TImageList \*ImageList1;

TPanel \*Panel1;

TPanel \*pCustomStyle;

TAction \*acSwitchEditMode;

TAction \*acCopy;

TAction \*acPaste;

TAction \*acCut;

TfrxFontComboBox \*cbTextFont;

TComboBox \*cbTextSize;

TLabel \*Label2;

TLabel \*Label3;

TAction \*acSaveUserStyle;

TAction \*acDeleteStyle;

TAction \*acClose;

TMenuItem \*Gh1;

TMenuItem \*N9;

TMenuItem \*N10;

TMenuItem \*N12;

TTimer \*tmUpdater;

TAction \*acUndo;

TAction \*acRedo;

TPopupMenu \*PopupMenu1;

TMenuItem \*N13;

TMenuItem \*N14;

TMenuItem \*N15;

TMenuItem \*N16;

TMenuItem \*N17;

TMenuItem \*N18;

TMenuItem \*N6;

TMenuItem \*N8;

TMenuItem \*N11;

TWindowCascade \*WindowCascade1;

TWindowTileHorizontal \*WindowTileHorizontal1;

TWindowTileVertical \*WindowTileVertical1;

TWindowMinimizeAll \*WindowMinimizeAll1;

TWindowArrange \*WindowArrange1;

TMenuItem \*ileHorizontally1;

TMenuItem \*ileVertically1;

TMenuItem \*Arrange1;

TSaveDialog \*SaveDocumentDialog;

TOpenDialog \*OpenDocumentDialog;

TControlBar \*ControlBar1;

TToolBar \*ToolBar2;

TToolButton \*ToolButton14;

TToolButton \*ToolButton8;

TToolButton \*ToolButton9;

TToolButton \*ToolButton10;

TToolBar \*ToolBar3;

TToolButton \*ToolButton11;

TToolButton \*ToolButton12;

TToolButton \*ToolButton13;

TToolButton \*ToolButton7;

TToolButton \*ToolButton15;

TToolButton \*ToolButton16;

TToolBar \*ToolBar1;

TToolButton \*ToolButton1;

TToolButton \*ToolButton2;

TToolButton \*ToolButton3;

TToolButton \*ToolButton4;

TToolButton \*ToolButton5;

TToolButton \*ToolButton6;

TButtonColor \*bTextColor;

TLabel \*Label1;

TLabel \*Label4;

TComboBox \*cbStyle;

TSpeedButton \*SpeedButton1;

TSpeedButton \*SpeedButton2;

void \_\_fastcall AboutActionClick(TObject \*Sender);

void \_\_fastcall FormCreate(TObject \*Sender);

void \_\_fastcall FormDestroy(TObject \*Sender);

void \_\_fastcall acOpenFileExecute(TObject \*Sender);

void \_\_fastcall acDeleteStyleExecute(TObject \*Sender);

void \_\_fastcall acSaveUserStyleExecute(TObject \*Sender);

void \_\_fastcall acCloseExecute(TObject \*Sender);

void \_\_fastcall acSaveFileExecute(TObject \*Sender);

void \_\_fastcall acNewPageExecute(TObject \*Sender);

void \_\_fastcall acBoldExecute(TObject \*Sender);

void \_\_fastcall acItalicsExecute(TObject \*Sender);

void \_\_fastcall acUnderlineExecute(TObject \*Sender);

void \_\_fastcall acCopyExecute(TObject \*Sender);

void \_\_fastcall acPasteExecute(TObject \*Sender);

void \_\_fastcall acInsertListExecute(TObject \*Sender);

void \_\_fastcall acInsertHyperlinkExecute(TObject \*Sender);

void \_\_fastcall acSaveFileAsExecute(TObject \*Sender);

void \_\_fastcall acCutExecute(TObject \*Sender);

void \_\_fastcall acInsertImageExecute(TObject \*Sender);

void \_\_fastcall tmUpdaterTimer(TObject \*Sender);

void \_\_fastcall acExitExecute(TObject \*Sender);

void \_\_fastcall acUndoExecute(TObject \*Sender);

void \_\_fastcall acRedoExecute(TObject \*Sender);

void \_\_fastcall bTextColorClick(TObject \*Sender);

void \_\_fastcall cbTextSizeSelect(TObject \*Sender);

void \_\_fastcall cbTextFontClick(TObject \*Sender);

void \_\_fastcall cbStyleSelect(TObject \*Sender);

private: // User declarations

String FormTitle;

StylesCollection \*defaultstyles,\*styles; //указатель на коллекции стандарных

//и пользовательских стилей

Style \*curstyle;

public: // User declarations

\_\_fastcall TForm1(TComponent\* Owner);

void SetStyleToUI(Style \*style); // установить элементы управления стилем

// в соответсвие со стилем style

};

//---------------------------------------------------------------------------

extern PACKAGE TForm1 \*Form1;

//---------------------------------------------------------------------------

#endif

MainWindowUnit.cpp

// ---------------------------------------------------------------------------

#include <vcl.h>

#pragma hdrstop

#include "MainWindowUnit.h"

#include "About.h"

// ---------------------------------------------------------------------------

#pragma package(smart\_init)

#pragma link "SHDocVw\_OCX"

#pragma link "frxCtrls"

#pragma resource "\*.dfm"

TForm1 \*Form1;

// ---------------------------------------------------------------------------

\_\_fastcall TForm1::TForm1(TComponent\* Owner) : TForm(Owner) {

}

// ---------------Показать форму "О программе"---------------------------------------------------

void \_\_fastcall TForm1::AboutActionClick(TObject \*Sender) {

AboutBox->ShowModal();

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::FormCreate(TObject \*Sender) {

defaultstyles = new StylesCollection; // создаем коллекцию стандартных и пользовательскиъх стилей

styles = new StylesCollection;

styles->LoadFromFile("styles.dat");

curstyle = new Style;

Style \*st;

st = new Style(0, clBlack, "", fsNONE, L"Текущий стиль");

\*curstyle = \*st;

defaultstyles->AddStyle(st);

delete st;

st = new Style(0, clBlack, "", fsNONE, L"Стандартный");

defaultstyles->AddStyle(st);

delete st;

st = new Style(7, clRed, "", fsBOLD, L"Заголовок");

defaultstyles->AddStyle(st);

delete st;

st = new Style(1, clBlack, "", fsNONE, L"Маленький");

defaultstyles->AddStyle(st);

delete st;

st = new Style(7, clBlack, "", fsNONE, L"Огромный");

defaultstyles->AddStyle(st);

delete st;

st = new Style(0, clBlue, "", fsUNDERLINE, L"Ссылка");

defaultstyles->AddStyle(st);

delete st;

defaultstyles->FillComboBox(cbStyle);

styles->FillComboBox(cbStyle);

cbStyle->ItemIndex = 0;

cbTextFont->ItemIndex = -1;

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::FormDestroy(TObject \*Sender) {

// сохранить пользовательские стили при закрытии

styles->SaveToFile("styles.dat");

delete styles;

delete defaultstyles;

delete curstyle;

}

// Выбор имени файла для открытия и окрытие файла-----------------------------------------------

void \_\_fastcall TForm1::acOpenFileExecute(TObject \*Sender) {

if (OpenDocumentDialog->Execute()) {

acNewPage->Execute();

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

form->doc->OpenFile(OpenDocumentDialog->FileName);

form->Caption = ExtractFileName(OpenDocumentDialog->FileName);

}

}

// ---------------------------------------------------------------------------

// Удаление выбранного стиля ---------------------------------------------------

void \_\_fastcall TForm1::acDeleteStyleExecute(TObject \*Sender) {

String stylename;

int index = cbStyle->ItemIndex;

stylename = cbStyle->Items->Strings[index];

if (stylename[1] == '@') {

styles->DeleteStyle(stylename);

cbStyle->Clear();

defaultstyles->FillComboBox(cbStyle);

styles->FillComboBox(cbStyle);

cbStyle->ItemIndex = 0;

}

else

ShowMessage(L"Нельзя удалить стандартный стиль");

}

//Создание нового стиля---------------------------------------------------------

void \_\_fastcall TForm1::acSaveUserStyleExecute(TObject \*Sender) {

//

String stylename;

stylename = InputBox(L"Выберите имя для нового стиля", L"Имя стиля",

L"Новый стиль");

Style \*style = new Style;

\*style = \*curstyle;

style->setstylename("@" + stylename);

styles->AddStyle(style);

cbStyle->Clear();

defaultstyles->FillComboBox(cbStyle);

styles->FillComboBox(cbStyle);

cbStyle->ItemIndex = cbStyle->Items->Count - 1;

}

//закрыть текущую дочернюю форму------------------------------------------------

void \_\_fastcall TForm1::acCloseExecute(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

form->Close();

}

// Выбор имени файла для сохранения и сохранение--------------------------------

void \_\_fastcall TForm1::acSaveFileExecute(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

if (form->doc->FileName == "") {

SaveDocumentDialog->FileName = form->Caption;

if (SaveDocumentDialog->Execute()) {

form->doc->FileName = SaveDocumentDialog->FileName;

form->Caption = SaveDocumentDialog->FileName;

}

else

return;

}

form->doc->SaveFile(form->doc->FileName);

}

// Новый документ---------------------------------------------------------------

void \_\_fastcall TForm1::acNewPageExecute(TObject \*Sender) {

// TO DO check if document opened

TDocumentForm \*form = new TDocumentForm(Application);

if (!form)

return;

form->Caption = "Безымянный " + IntToStr(MDIChildCount);

form->Show();

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::acBoldExecute(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

form->browser->Bold();

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::acItalicsExecute(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

form->browser->Italic();

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::acUnderlineExecute(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

form->browser->UnderLine();

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::acCopyExecute(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

if (form->RichEdit1->Focused())

form->RichEdit1->CopyToClipboard();

else

form->browser->Copy();

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::acPasteExecute(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

if (form->RichEdit1->Focused())

form->RichEdit1->PasteFromClipboard();

else

form->browser->Paste();

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::acInsertListExecute(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

form->browser->InsertList();

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::acInsertHyperlinkExecute(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

form->browser->HyperLink();

}

// Нажатие кнопки сохранить как ...---------------------------------------------

void \_\_fastcall TForm1::acSaveFileAsExecute(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

SaveDocumentDialog->FileName = ExtractFileName(form->doc->FileName);

SaveDocumentDialog->InitialDir = ExtractFileDir(form->doc->FileName);

if (SaveDocumentDialog->Execute()) {

form->doc->FileName = SaveDocumentDialog->FileName;

form->Caption = SaveDocumentDialog->FileName;

}

else

return;

form->doc->SaveFile(form->doc->FileName);

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::acCutExecute(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

if (form->RichEdit1->Focused())

form->RichEdit1->CutToClipboard();

else

form->browser->Cut();

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::acInsertImageExecute(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

form->browser->InsertImage();

}

//Переодически вызывается раз в секунду для определения изменений в webbrowser ---------------------------------------------------------------------------

void \_\_fastcall TForm1::tmUpdaterTimer(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form) {

return;

}

if (!form->RichEdit1->Focused()) {

form->RichEdit1->Text = form->browser->GetText();

bool is\_txt = form->browser->TxtRange();

acBold->Enabled = is\_txt;

acItalics->Enabled = is\_txt;

acUnderline->Enabled = is\_txt;

acInsertHyperlink->Enabled = is\_txt;

acInsertImage->Enabled = is\_txt;

acInsertList->Enabled = is\_txt;

acUndo->Enabled = form->browser->CanUndo();

acRedo->Enabled = form->browser->CanRedo();

acCut->Enabled = form->browser->CanCut();

acCopy->Enabled = form->browser->CanCopy();

acPaste->Enabled = form->browser->CanPaste();

}

else {

acBold->Enabled = false;

acItalics->Enabled = false;

acUnderline->Enabled = false;

acInsertHyperlink->Enabled = false;

acInsertImage->Enabled = false;

acInsertList->Enabled = false;

acUndo->Enabled = form->RichEdit1->CanUndo;

acRedo->Enabled = false;

acCut->Enabled = form->RichEdit1->SelLength > 0;

acCopy->Enabled = form->RichEdit1->SelLength > 0;

acPaste->Enabled = form->browser->CanPaste();

}

IHTMLTxtRange \*range = form->browser->TxtRange();

if (range != NULL && !form->RichEdit1->Focused()) {

acBold->Checked = form->browser->isBold(range);

acItalics->Checked = form->browser->isItalic(range);

acUnderline->Checked = form->browser->isUnderline(range);

bTextColor->SymbolColor = form->browser->GetColor(range);

cbTextFont->Text = form->browser->GetFont(range);

cbTextSize->ItemIndex = form->browser->GetSize(range);

}

// exper

}

// Закрытие программы---------------------------------------------------------------------------

void \_\_fastcall TForm1::acExitExecute(TObject \*Sender) {

for (int i = MDIChildCount - 1; i >= 0; i--) {

if (MDIChildren[i]->CloseQuery()) {

MDIChildren[i]->Close();

}

else

return;

}

Close();

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::acUndoExecute(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

form->browser->Undo();

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::acRedoExecute(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

form->browser->Redo();

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::bTextColorClick(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

curstyle->setcolor(ColorToRGB(bTextColor->SymbolColor));

cbStyle->ItemIndex = 0;

form->browser->SetColor(ColorToRGB(bTextColor->SymbolColor));

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::cbTextSizeSelect(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

int size = cbTextSize->ItemIndex;

cbStyle->ItemIndex = 0;

curstyle->setsize(size);

form->browser->SetSize(size);

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::cbTextFontClick(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

curstyle->setface(cbTextFont->Text);

cbStyle->ItemIndex = 0;

form->browser->SetFont(cbTextFont->Text);

}

// Установить текущий стиль---------------------------------------------------------------------------

void TForm1::SetStyleToUI(Style \*style) {

cbTextSize->ItemIndex = style->getsize();

cbTextFont->Text = style->getface();

bTextColor->SymbolColor = style->getcolor();

int fontstyle = style->getfontstyle();

acBold->Checked = ((fontstyle & fsBOLD) == fsBOLD);

acItalics->Checked = ((fontstyle & fsITALIC) == fsITALIC);

acUnderline->Checked = ((fontstyle & fsUNDERLINE) == fsUNDERLINE);

}

void \_\_fastcall TForm1::cbStyleSelect(TObject \*Sender) {

TDocumentForm \*form = (TDocumentForm\*)this->ActiveMDIChild;

if (!form)

return;

int index = cbStyle->ItemIndex;

Style \*style = (Style\*)cbStyle->Items->Objects[index];

\*curstyle = \*style;

SetStyleToUI(style);

form->browser->SetStyle(style);

cbStyle->ItemIndex = 0;

}

// ---------------------------------------------------------------------------

DocumentFowmUnit.h

// ---------------------------------------------------------------------------

#ifndef DocumentFormUnitH

#define DocumentFormUnitH

// ---------------------------------------------------------------------------

#include <System.Classes.hpp>

#include <Vcl.Controls.hpp>

#include <Vcl.StdCtrls.hpp>

#include <Vcl.Forms.hpp>

#include "SHDocVw\_OCX.h"

#include <Vcl.ComCtrls.hpp>

#include <Vcl.ExtCtrls.hpp>

#include <Vcl.OleCtrls.hpp>

#include <Vcl.ActnList.hpp>

#include <Vcl.Menus.hpp>

#include "BrowserSys.h"

#include "HTMLDocument.h"

#include <Vcl.ImgList.hpp>

#include <Vcl.AppEvnts.hpp>

// ---------------------------------------------------------------------------

class TDocumentForm : public TForm {

\_\_published: // IDE-managed Components

TRichEdit \*RichEdit1;

TCppWebBrowser \*WebBrowser1;

TSplitter \*Splitter1;

void \_\_fastcall RichEdit1Change(TObject \*Sender);

void \_\_fastcall FormCreate(TObject \*Sender);

void \_\_fastcall FormDestroy(TObject \*Sender);

void \_\_fastcall FormCloseQuery(TObject \*Sender, bool &CanClose);

void \_\_fastcall WebBrowser1DocumentComplete(TObject \*Sender,

LPDISPATCH pDisp, Variant \*URL);

void \_\_fastcall FormClose(TObject \*Sender, TCloseAction &Action);

private: // User declarations

public : // User declarations

\_\_fastcall TDocumentForm(TComponent\* Owner);

HTMLDocument \*doc;

BrowserSys \*browser;

int updatetimes; // счетчик числа обновлений

// RichEdit обновится из webbrowser два раза при создании нового документа

//

};

// ---------------------------------------------------------------------------

extern PACKAGE TDocumentForm \*DocumentForm;

// ---------------------------------------------------------------------------

#endif

DocumentFormUnit.cpp

// ---------------------------------------------------------------------------

#include <vcl.h>

#pragma hdrstop

#include "DocumentFormUnit.h"

#include "MainWindowUnit.h"

// ---------------------------------------------------------------------------

#pragma package(smart\_init)

#pragma link "SHDocVw\_OCX"

#pragma resource "\*.dfm"

TDocumentForm \*DocumentForm;

// ---------------------------------------------------------------------------

\_\_fastcall TDocumentForm::TDocumentForm(TComponent\* Owner) : TForm(Owner) {

}

// Обновление документа при обновлениии содержимого в RichEdit------------------

void \_\_fastcall TDocumentForm::RichEdit1Change(TObject \*Sender) {

if (RichEdit1->Lines->Text != doc->html) {

if (updatetimes > 1)

doc->setchanged(true);

else

updatetimes++;

doc->html = RichEdit1->Lines->Text;

}

if (RichEdit1->Focused()) {

doc->Update(RichEdit1->Lines->Text);

}

}

// ---------------------------------------------------------------------------

void \_\_fastcall TDocumentForm::FormCreate(TObject \*Sender) {

RichEdit1->PopupMenu = static\_cast<TForm1\*>(Application->MainForm)

->PopupMenu1;

browser = new BrowserSys(WebBrowser1, RichEdit1->PopupMenu);

doc = new HTMLDocument(RichEdit1, browser);

doc->setchanged(false);

updatetimes = 0;

}

// ---------------------------------------------------------------------------

void \_\_fastcall TDocumentForm::FormDestroy(TObject \*Sender) {

delete doc;

delete browser;

}

// Запрос на закрытие формы документа-------------------------------------------

void \_\_fastcall TDocumentForm::FormCloseQuery(TObject \*Sender, bool &CanClose) {

if (doc->changed()) {

int qresult;

qresult = MessageDlg("Сохранить текущий документ [" + Caption +

"] перед закрытием", mtConfirmation,

TMsgDlgButtons() << mbYes << mbNo << mbCancel, 0);

if (qresult == mrNo)

CanClose = true;

else {

CanClose = false;

if (qresult == mrYes) {

TForm1 \*form = (TForm1\*)(Application->MainForm);

this->Activate();

form->acSaveFile->Execute();

}

}

}

else

CanClose = true;

}

// Инициализация интерфейсов TWebBrowser----------------------------------------

void \_\_fastcall TDocumentForm::WebBrowser1DocumentComplete(TObject \*Sender,

LPDISPATCH pDisp, Variant \*URL) {

if (browser->InitInterfaces())

browser->EditMode(true);

}

// ---------------------------------------------------------------------------

void \_\_fastcall TDocumentForm::FormClose(TObject \*Sender, TCloseAction &Action)

{

Action = caFree; // необходимо для автоуничтожения формы при закрытии

}

// ---------------------------------------------------------------------------

Style.h

// ---------------------------------------------------------------------------

#include <vcl.h>

#pragma hdrstop

#include "DocumentFormUnit.h"

#include "MainWindowUnit.h"

// ---------------------------------------------------------------------------

#pragma package(smart\_init)

#pragma link "SHDocVw\_OCX"

#pragma resource "\*.dfm"

TDocumentForm \*DocumentForm;

// ---------------------------------------------------------------------------

\_\_fastcall TDocumentForm::TDocumentForm(TComponent\* Owner) : TForm(Owner) {

}

// Обновление документа при обновлениии содержимого в RichEdit------------------

void \_\_fastcall TDocumentForm::RichEdit1Change(TObject \*Sender) {

if (RichEdit1->Lines->Text != doc->html) {

if (updatetimes > 1)

doc->setchanged(true);

else

updatetimes++;

doc->html = RichEdit1->Lines->Text;

}

if (RichEdit1->Focused()) {

doc->Update(RichEdit1->Lines->Text);

}

}

// ---------------------------------------------------------------------------

void \_\_fastcall TDocumentForm::FormCreate(TObject \*Sender) {

RichEdit1->PopupMenu = static\_cast<TForm1\*>(Application->MainForm)

->PopupMenu1;

browser = new BrowserSys(WebBrowser1, RichEdit1->PopupMenu);

doc = new HTMLDocument(RichEdit1, browser);

doc->setchanged(false);

updatetimes = 0;

}

// ---------------------------------------------------------------------------

void \_\_fastcall TDocumentForm::FormDestroy(TObject \*Sender) {

delete doc;

delete browser;

}

// Запрос на закрытие формы документа-------------------------------------------

void \_\_fastcall TDocumentForm::FormCloseQuery(TObject \*Sender, bool &CanClose) {

if (doc->changed()) {

int qresult;

qresult = MessageDlg("Сохранить текущий документ [" + Caption +

"] перед закрытием", mtConfirmation,

TMsgDlgButtons() << mbYes << mbNo << mbCancel, 0);

if (qresult == mrNo)

CanClose = true;

else {

CanClose = false;

if (qresult == mrYes) {

TForm1 \*form = (TForm1\*)(Application->MainForm);

this->Activate();

form->acSaveFile->Execute();

}

}

}

else

CanClose = true;

}

// Инициализация интерфейсов TWebBrowser----------------------------------------

void \_\_fastcall TDocumentForm::WebBrowser1DocumentComplete(TObject \*Sender,

LPDISPATCH pDisp, Variant \*URL) {

if (browser->InitInterfaces())

browser->EditMode(true);

}

// ---------------------------------------------------------------------------

void \_\_fastcall TDocumentForm::FormClose(TObject \*Sender, TCloseAction &Action)

{

Action = caFree; // необходимо для автоуничтожения формы при закрытии

}

// ---------------------------------------------------------------------------

Style.cpp

// ---------------------------------------------------------------------------

#pragma hdrstop

#include "Style.h"

// ---------------------------------------------------------------------------

#pragma package(smart\_init)

Style::Style(int size, int color, const char \*font, int fontstyle,

String stylename) {

setsize(size);

setcolor(color);

setface(String(font));

setfontstyle(fontstyle);

setstylename(stylename);

}

Style::Style(int size, TColor color, const char \*font, int fontstyle,

String stylename) {

setsize(size);

setcolor(color);

setface(String(font));

setfontstyle(fontstyle);

setstylename(stylename);

}

Style::Style(const Style &obj) {

setsize(obj.getsize());

setcolor(obj.getcolor());

setface(obj.getface());

setfontstyle(obj.getfontstyle());

setstylename(obj.getstylename());

}

Style::Style() {

String face("");

setsize(0);

setcolor(0);

setface(face);

setfontstyle(fsNONE);

setstylename("");

}

void Style::setsize(int size) {

if (size <= 7 && size >= 0)

fontsize = size;

else

fontsize = 0;

}

int Style::getsize() const {

return fontsize;

}

void Style::setface(String s) {

fontface = s;

}

String Style::getface() const {

return fontface;

}

void Style::setcolor(int colorRGB) {

fontcolor = colorRGB;

}

TColor Style::getcolor() const {

return static\_cast<TColor>(fontcolor);

}

void Style::setcolor(TColor color) {

fontcolor = ColorToRGB(color);

}

void Style::setfontstyle(int style) {

fontstyle = style;

}

int Style::getfontstyle() const {

return fontstyle;

}

String Style::getstylename() const {

return stylename;

}

void Style::setstylename(String stylename) {

this->stylename = stylename;

}

void Style::write(ofstream &fs) {

int var;

SaveString(getstylename(), fs);

var = getsize();

fs.write((char\*) &var, sizeof(var));

SaveString(getface(), fs);

var = getcolor();

fs.write((char\*) &var, sizeof(var));

var = getfontstyle();

fs.write((char\*) &var, sizeof(var));

}

void Style::read(ifstream &fs) {

int var;

setstylename(LoadString(fs));

fs.read((char\*) &var, sizeof(var));

setsize(var);

setface(LoadString(fs));

fs.read((char\*) &var, sizeof(var));

setcolor(var);

fs.read((char\*) &var, sizeof(var));

setfontstyle(var);

}

void Style::SaveString(String s, ofstream &fs) {

int len;

len = s.Length() \* s.ElementSize();

fs.write((char\*) &len, sizeof(len)); // char count

fs.write((char\*) s.c\_str(), len); // string content in unicode

}

String Style::LoadString(ifstream &fs) {

int len;

fs.read((char \*) &len, sizeof(len));

String s;

s.SetLength((int) len / s.ElementSize());

fs.read((char\*) s.data(), len);

return s;

}

Style& Style:: operator = (const Style & right) {

if (this == &right) {

return \*this;

}

setsize(right.getsize());

setcolor(right.getcolor());

setface(right.getface());

setfontstyle(right.getfontstyle());

setstylename(right.getstylename()); // ??

return \*this;

}

StylesCollection.h

// ---------------------------------------------------------------------------

#ifndef StylesCollectionH

#define StylesCollectionH

// ---------------------------------------------------------------------------

#include <vector>

#include "Style.h"

#include <fstream.h>

#include <Vcl.StdCtrls.hpp>

using namespace std;

typedef vector<Style\*> stylesvector; // typedef для удобства

// коллекция стилей

class StylesCollection {

public:

bool LoadFromFile(String filename); // открытие и чтение из файла

bool SaveToFile(String filename); // запись в файл

void FillComboBox(TComboBox \*cb); // запись содержимого в TComboBox

void AddStyle(Style \*style); // добавление нового стиля

void DeleteStyle(String stylename); // удаление стиля

private:

stylesvector styles; // вектор, хранящий стили

};

#endif

StylesCollection.cpp

// ---------------------------------------------------------------------------

#pragma hdrstop

#include "StylesCollection.h"

// ---------------------------------------------------------------------------

#pragma package(smart\_init)

void StylesCollection::FillComboBox(TComboBox \*cb) {

stylesvector::const\_iterator i;

for (i = styles.begin(); i != styles.end(); i++) {

cb->Items->AddObject((\*i)->getstylename(), \*i);

}

}

bool StylesCollection::LoadFromFile(String filename) {

ifstream fs(filename.c\_str(), ios\_base::binary);

if (!fs) {

return false;

}

int count;

fs.read((char\*)&count, sizeof(count));

for (int i = 0; i < count; i++) {

Style \*style = new Style;

style->read(fs);

this->AddStyle(style);

}

fs.close();

return true;

}

bool StylesCollection::SaveToFile(String filename) {

ofstream fs(filename.c\_str(), ios\_base::binary);

if (!fs) {

return false;

}

int count = styles.size();

fs.write((char\*)&count,sizeof(count));

for (int i = 0; i < count; i++) {

styles[i]->write(fs);

}

fs.close();

return true;

}

void StylesCollection::AddStyle(Style \*style) {

Style \*buf\_style;

buf\_style = new Style(\*style);

styles.push\_back(buf\_style);

}

void StylesCollection::DeleteStyle(String stylename) {

stylesvector::iterator i;

Style \*st;

for (i = styles.begin(); i != styles.end(); i++) {

st = \*i;

if (st->getstylename() == stylename) {

// deleting

styles.erase(i);

delete st;

break;

}

}

}

htmldocument.h

// ---------------------------------------------------------------------------

#ifndef HTMLDocumentH

#define HTMLDocumentH

// ---------------------------------------------------------------------------

#include <system.hpp>

#include <Vcl.ComCtrls.hpp>

#include "BrowserSys.h"

#include "mshtml.h"

// класс для хранения текущего документа

class HTMLDocument {

public:

String FileName; // имя ассоцированного файла на ЖД

String html; // документ в виде строки

// конструктор, связывает RichEdit с BrowserSys

HTMLDocument(TRichEdit \*rche, BrowserSys \*browser);

void OpenFile(const String &filename); // загрузить из файла

void SaveFile(const String &filename); // сохранить в файл

bool changed(); // изменилось ли содержимое документа с последней

//загрузки/сохранения

void setchanged(bool flag); // изменение флага изменения документа

void Update(const String &html); // каждый раз когда содержимое изменилось

private:

bool \_changed; // флаг изменнения

BrowserSys \*browser; // указатель на ассоциированный BrowserSys

TRichEdit \*rcedit;

};

#endif

htmldocument.cpp

//---------------------------------------------------------------------------

#pragma hdrstop

#include <fstream.h>

#include "HTMLDocument.h"

//---------------------------------------------------------------------------

#pragma package(smart\_init)

void HTMLDocument::OpenFile(const String &filename)

{

TStringList \*sl = new TStringList;

FileName = filename;

sl->LoadFromFile(filename);

browser->OpenFile(filename);

rcedit->Lines->Clear();

html = sl->Text;

rcedit->Text = html;

\_changed = false;

delete sl;

}

void HTMLDocument::Update(const String &html){

this->html = html;

\_changed = true;

browser->SetText(html);

}

void HTMLDocument::SaveFile(const String &filename)

{ TStringList \*sl = new TStringList;

sl->Text = html;

sl->SaveToFile(filename);

delete sl;

\_changed = false;

}

bool HTMLDocument::changed(){

return \_changed;

}

HTMLDocument::HTMLDocument(TRichEdit \*rche, BrowserSys \*Browser):

rcedit(rche),browser(Browser)

{

};

void HTMLDocument::setchanged(bool flag){

\_changed = flag;

}

BrowserSys.h

// ---------------------------------------------------------------------------

#ifndef BrowserSysH

#define BrowserSysH

#include <oaidl.h>

#include "SHDocVw\_OCX.h"

#include <SHDocVw.hpp>

#include <mshtml.h>

#include <Vcl.Graphics.hpp>

#include "Style.h"

#include <oaidl.h>

#include <Vcl.Dialogs.hpp>

#include "mshtmhst.h"

#include "SHDocVw\_OCX.h"

class MyDocHandler;

// класс-обертка для webbrowser

class BrowserSys {

public:

BrowserSys(TCppWebBrowser \*wb, TPopupMenu \*popupmenu); // конструктор получает

// \*Webbrowser и указатель на контекстное меню

~BrowserSys();

bool InitInterfaces(); // инициализация

void EditMode(bool on = true); // перевод документа в режим редактирования

bool CanPaste(); // можно ли вставить в документ из буфера обмена

bool CanCopy(); // можно ли копировать

bool CanCut(); // можно ли вырезать в буфер

bool CanRedo(); // можно ли повторить отмененное действие

bool CanUndo(); // отменить последнее действие

void Bold(); // установить полужирное наертание выделенного текста

void Italic();

void Paste(); // вставить из буфера

void Copy();

void Cut();

void UnderLine();

void InsertList(); // пометить как список

void HyperLink(); // пометить как гиперссылку

void InsertImage(); // вставить изображение

void Undo();

void Redo();

void SetSize(int size); // утсановить размер текста

void SetFont(const String &font); // установить шрифт

void SetColor(int color);

int GetSize(IHTMLTxtRange\* TextRange); // узнать размер текста

String GetFont(IHTMLTxtRange\* TextRange);

int GetColor(IHTMLTxtRange\* TextRange);

bool isItalic(IHTMLTxtRange\* TextRange); // является ли текст наклонным?

bool isBold(IHTMLTxtRange\* TextRange);

bool isUnderline(IHTMLTxtRange\* TextRange);

IHTMLTxtRange\* TxtRange(); // получить текущую выделенную область текста

void SetStyle(Style \*style);

String GetText(); // извлечь текст из webbrowser

void SetText(String s); // установить текст

void OpenFile(String filename);

private:

TCppWebBrowser \*wb;

IHTMLDocument2 \*Editor;

MyDocHandler \*dochandler;

VARIANT var;

VARIANT\_BOOL rec;

const String HtmlColor(int color);

};

// ---------------------------------------------------------------------------

// небходимо реализовать для утановки собственного меню и отключения системных

// горячих клавиш

class MyDocHandler : public IDocHostUIHandler {

long refcount;

TPopupMenu \*popup;

public:

MyDocHandler(TPopupMenu \*popupmenu) : refcount(1) {

popup = popupmenu;

}

virtual HRESULT STDMETHODCALLTYPE QueryInterface(REFIID classid,

void\*\* intf) {

if (classid == IID\_IUnknown)

\* intf = (IUnknown\*)this;

else if (classid == IID\_IDocHostUIHandler)

\* intf = (IDocHostUIHandler\*)this;

else

return E\_NOINTERFACE;

return S\_OK;

}

virtual ULONG STDMETHODCALLTYPE AddRef() {

InterlockedIncrement(&refcount);

return refcount;

}

virtual ULONG STDMETHODCALLTYPE Release() {

InterlockedDecrement(&refcount);

if (refcount == 0)

delete this;

return refcount;

}

// Returning S\_OK tells the web browser that it need not display its

// own context menu, presumably because the application hosting it has

// displayed its own menu to replace it.

virtual HRESULT STDMETHODCALLTYPE ShowContextMenu(

/\* [in] \*/ DWORD dwID,

/\* [in] \*/ POINT \_\_RPC\_FAR \*ppt,

/\* [in] \*/ IUnknown \_\_RPC\_FAR \*pcmdtReserved,

/\* [in] \*/ IDispatch \_\_RPC\_FAR \*pdispReserved) {

popup->Popup(ppt->x, ppt->y);

return S\_OK;

}

virtual HRESULT STDMETHODCALLTYPE GetHostInfo(

/\* [out][in] \*/ DOCHOSTUIINFO \_\_RPC\_FAR \*pInfo) {

pInfo->dwFlags = 0x00000000;

return S\_OK;

}

virtual HRESULT STDMETHODCALLTYPE ShowUI(

/\* [in] \*/ DWORD dwID,

/\* [in] \*/ IOleInPlaceActiveObject \_\_RPC\_FAR \*pActiveObject,

/\* [in] \*/ IOleCommandTarget \_\_RPC\_FAR \*pCommandTarget,

/\* [in] \*/ IOleInPlaceFrame \_\_RPC\_FAR \*pFrame,

/\* [in] \*/ IOleInPlaceUIWindow \_\_RPC\_FAR \*pDoc) {

return E\_NOTIMPL;

}

virtual HRESULT STDMETHODCALLTYPE HideUI(void) {

return E\_NOTIMPL;

}

virtual HRESULT STDMETHODCALLTYPE UpdateUI(void) {

return E\_NOTIMPL;

}

virtual HRESULT STDMETHODCALLTYPE EnableModeless(

/\* [in] \*/ BOOL fEnable) {

return E\_NOTIMPL;

}

virtual HRESULT STDMETHODCALLTYPE OnDocWindowActivate(

/\* [in] \*/ BOOL fActivate) {

return E\_NOTIMPL;

}

virtual HRESULT STDMETHODCALLTYPE OnFrameWindowActivate(

/\* [in] \*/ BOOL fActivate) {

return E\_NOTIMPL;

}

virtual HRESULT STDMETHODCALLTYPE ResizeBorder(

/\* [in] \*/ LPCRECT prcBorder,

/\* [in] \*/ IOleInPlaceUIWindow \_\_RPC\_FAR \*pUIWindow,

/\* [in] \*/ BOOL fRameWindow) {

return E\_NOTIMPL;

}

virtual HRESULT STDMETHODCALLTYPE TranslateAccelerator(

/\* [in] \*/ LPMSG lpMsg,

/\* [in] \*/ const GUID \_\_RPC\_FAR \*pguidCmdGroup,

/\* [in] \*/ DWORD nCmdID) {

// ShowMessage(GetKeyState(VK\_CONTROL));

if (lpMsg->message == WM\_KEYDOWN)

if ((GetKeyState(VK\_CONTROL) & 0X80) == 0X80)

return S\_OK;

return E\_NOTIMPL;

}

virtual HRESULT STDMETHODCALLTYPE GetOptionKeyPath(

/\* [out] \*/ LPOLESTR \_\_RPC\_FAR \*pchKey,

/\* [in] \*/ DWORD dw) {

return E\_NOTIMPL;

}

virtual HRESULT STDMETHODCALLTYPE GetDropTarget(

/\* [in] \*/ IDropTarget \_\_RPC\_FAR \*pDropTarget,

/\* [out] \*/ IDropTarget \_\_RPC\_FAR \*\_\_RPC\_FAR \*ppDropTarget) {

return E\_NOTIMPL;

}

virtual HRESULT STDMETHODCALLTYPE GetExternal(

/\* [out] \*/ IDispatch \_\_RPC\_FAR \*\_\_RPC\_FAR \*ppDispatch) {

return E\_NOTIMPL;

}

virtual HRESULT STDMETHODCALLTYPE TranslateUrl(

/\* [in] \*/ DWORD dwTranslate,

/\* [in] \*/ OLECHAR \_\_RPC\_FAR \*pchURLIn,

/\* [out] \*/ OLECHAR \_\_RPC\_FAR \*\_\_RPC\_FAR \*ppchURLOut) {

return E\_NOTIMPL;

}

virtual HRESULT STDMETHODCALLTYPE FilterDataObject(

/\* [in] \*/ IDataObject \_\_RPC\_FAR \*pDO,

/\* [out] \*/ IDataObject \_\_RPC\_FAR \*\_\_RPC\_FAR \*ppDORet) {

return E\_NOTIMPL;

}

};

#endif

BrowserSys.cpp

// ---------------------------------------------------------------------------

#pragma hdrstop

#include "BrowserSys.h"

// ---------------------------------------------------------------------------

#pragma package(smart\_init)

BrowserSys::BrowserSys(TCppWebBrowser \*wb, TPopupMenu \*popupmenu) {

this->wb = wb;

wb->Silent = true;

wb->Navigate(WideString("about:blank").c\_bstr());

while (wb->ReadyState < ::READYSTATE\_INTERACTIVE)

Application->ProcessMessages();

wb->Offline = true;

Editor = NULL;

dochandler = new MyDocHandler(popupmenu);

}

BrowserSys::~BrowserSys() {

if (Editor) {

Editor->Release();

dochandler->Release();

delete dochandler;

}

}

bool BrowserSys::InitInterfaces() {

HRESULT hr;

ICustomDoc \*custdoc;

hr = wb->Document->QueryInterface(IID\_IHTMLDocument2, (void\*\*)&Editor);

if (SUCCEEDED(hr)) {

Editor->Release();

wb->Document->QueryInterface(&custdoc);

if (custdoc)

custdoc->SetUIHandler(dochandler);

return true;

}

else

return false;

}

void BrowserSys::EditMode(bool on) {

if (on)

Editor->put\_designMode(L"On");

else

Editor->put\_designMode(L"Off");

}

bool BrowserSys::CanCopy() {

return wb->QueryStatusWB(Shdocvw::OLECMDID\_COPY) != 1;

}

bool BrowserSys::CanPaste() {

return wb->QueryStatusWB(Shdocvw::OLECMDID\_PASTE) != 1;

}

bool BrowserSys::CanCut() {

return wb->QueryStatusWB(Shdocvw::OLECMDID\_CUT) != 1;

}

IHTMLTxtRange\* BrowserSys::TxtRange() {

IHTMLSelectionObject \*sel;

BSTR SelType;

HRESULT hr;

IDispatch \*disp;

IHTMLTxtRange \*txtrange;

Editor->get\_selection(&sel);

sel->get\_type(&SelType);

if (String(SelType) != "Control") {

hr = sel->createRange(&disp);

if (SUCCEEDED(hr)) {

disp->QueryInterface(IID\_IHTMLTxtRange, (void\*\*)&txtrange);

return txtrange;

}

}

return NULL;

}

void BrowserSys::Bold() {

TxtRange()->execCommand(L"bold", false, var, &rec);

}

void BrowserSys::Italic() {

TxtRange()->execCommand(L"italic", false, var, &rec);

}

void BrowserSys::UnderLine() {

TxtRange()->execCommand(L"underline", false, var, &rec);

}

void BrowserSys::Paste() {

wb->ExecWB(Shdocvw::OLECMDID\_PASTE, Shdocvw::OLECMDEXECOPT\_DONTPROMPTUSER);

}

void BrowserSys::Copy() {

wb->ExecWB(Shdocvw::OLECMDID\_COPY, Shdocvw::OLECMDEXECOPT\_DONTPROMPTUSER);

}

void BrowserSys::Cut() {

wb->ExecWB(Shdocvw::OLECMDID\_CUT, Shdocvw::OLECMDEXECOPT\_DONTPROMPTUSER);

}

void BrowserSys::InsertList() {

TxtRange()->execCommand(L"InsertUnorderedList", false, var, &rec);

}

void BrowserSys::HyperLink() {

TxtRange()->execCommand(L"CreateLink", true, var, &rec);

}

void BrowserSys::InsertImage() {

TxtRange()->execCommand(L"InsertImage", true, var, &rec);

}

void BrowserSys::Undo() {

wb->ExecWB(Shdocvw::OLECMDID\_UNDO, Shdocvw::OLECMDEXECOPT\_DONTPROMPTUSER);

}

void BrowserSys::Redo() {

wb->ExecWB(Shdocvw::OLECMDID\_REDO, Shdocvw::OLECMDEXECOPT\_DONTPROMPTUSER);

}

bool BrowserSys::CanUndo() {

return wb->QueryStatusWB(Shdocvw::OLECMDID\_UNDO) > 1;

}

bool BrowserSys::CanRedo() {

return wb->QueryStatusWB(Shdocvw::OLECMDID\_REDO) > 1;

}

void BrowserSys::SetSize(int size) {

Variant V;

if (size > 7 || size < 1)

V = "error";

else

V = IntToStr(size);

TxtRange()->execCommand(L"FontSize", false, V, &rec);

}

void BrowserSys::SetFont(const String &font) {

Variant V = font;

TxtRange()->execCommand(L"FontName", false, V, &rec);

}

void BrowserSys::SetColor(int color) {

Variant V = HtmlColor(color);

TxtRange()->execCommand(L"ForeColor", false, V, &rec);

}

const String BrowserSys::HtmlColor(int color) {

String s;

TColor col;

col = static\_cast<TColor>(color);

s.sprintf(L"#%.2X%.2X%.2X", GetRValue(col), GetGValue(col), GetBValue(col));

return s;

}

int BrowserSys::GetSize(IHTMLTxtRange\* TextRange) {

OleVariant v;

int size;

HRESULT hr;

if (TextRange) {

hr = TextRange->queryCommandValue(L"FontSize", (tagVARIANT\*)v);

if (SUCCEEDED(hr)) {

if (!v.IsNull())

size = v.intVal;

if (size <= 7 && size >= 0)

return size;

}

}

return 0;

}

String BrowserSys::GetFont(IHTMLTxtRange\* TextRange) {

OleVariant v;

HRESULT hr;

if (TextRange) {

hr = TextRange->queryCommandValue(L"FontName", (tagVARIANT\*)v);

if (SUCCEEDED(hr)) {

if (!v.IsNull())

return v.bstrVal;

}

}

return NULL;

}

int BrowserSys::GetColor(IHTMLTxtRange\* TextRange) {

OleVariant v;

HRESULT hr;

if (TextRange) {

hr = TextRange->queryCommandValue(L"ForeColor", (tagVARIANT\*)v);

if (SUCCEEDED(hr)) {

if (!v.IsNull())

return v.intVal;

}

}

return 0;

}

bool BrowserSys::isItalic(IHTMLTxtRange \*TextRange) {

HRESULT hr;

short sh;

if (TextRange) {

hr = TextRange->queryCommandState(L"Italic", &sh);

if (sh)

return true;

}

return false;

}

bool BrowserSys::isBold(IHTMLTxtRange \*TextRange) {

short sh;

if (TextRange) {

TextRange->queryCommandState(L"Bold", &sh);

if (sh)

return true;

}

return false;

}

bool BrowserSys::isUnderline(IHTMLTxtRange \*TextRange) {

short sh;

if (TextRange) {

TextRange->queryCommandState(L"Underline", &sh);

if (sh)

return true;

}

return false;

}

void BrowserSys::SetStyle(Style \*style) {

SetSize(style->getsize());

SetColor(style->getcolor());

SetFont(style->getface());

int k = style->getfontstyle();

if (isBold(TxtRange())) {

Bold();

}

if (isItalic(TxtRange())) {

Italic();

}

if (isUnderline(TxtRange())) {

UnderLine();

}

if ((k & fsBOLD) == fsBOLD)

this->Bold();

if ((k & fsITALIC) == fsITALIC)

this->Italic();

if ((k & fsUNDERLINE) == fsUNDERLINE)

this->UnderLine();

}

String BrowserSys::GetText() {

IHTMLElement\* pElement;

HRESULT hr;

BSTR s;

if (FAILED(Editor->get\_body(&pElement)))

return NULL;

if (FAILED(pElement->get\_parentElement(&pElement)))

return NULL;

pElement->get\_outerHTML(&s);

pElement->Release();

return String(s);

}

void BrowserSys::SetText(String s) {

IHTMLElement\* pElement;

if (!Editor)

return;

if (FAILED(Editor->get\_body(&pElement)))

return;

pElement->put\_innerHTML(s.c\_str());

pElement->Release();

}

void BrowserSys::OpenFile(String filename) {

wb->Navigate(filename.c\_str());

while (wb->ReadyState < ::READYSTATE\_INTERACTIVE)

Application->ProcessMessages();

}