unit Loading;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, Sys-tem.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ComCtrls, Vcl.ExtCtrls, Vcl.StdCtrls,

Vcl.Imaging.pngimage;

type

TLoadingForm = class(TForm)

Timer1: TTimer;

ProgressBar1: TProgressBar;

Logo: TImage;

LogoText: TImage;

procedure Timer1Timer(Sender: TObject);

procedure FormCreate(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

LoadingForm: TLoadingForm;

implementation

{$R \*.dfm}

procedure TLoadingForm.FormCreate(Sender: TObject);

begin

ProgressBar1.Position := 0;

Timer1.Interval := 20;

Timer1.Enabled := True;

end;

procedure TLoadingForm.Timer1Timer(Sender: TObject);

begin

if ProgressBar1.Position < ProgressBar1.Max then

ProgressBar1.Position := ProgressBar1.Position + 1

else

begin

Timer1.Enabled := False;

Close;

end;

end;

end.

unit Main;

interface

uses

Windows, SysUtils, Graphics, Forms, Controls, Menus, StdCtrls, Dialogs, Buttons, Messages,

ExtCtrls, ComCtrls, ToolWin, ImageEnProc, ImageEnView, ImageEnIO, hyiedefs, IEOpenSaveDlg, hyieutils,

iexBitmaps, iesettings, iexLayers, iexRulers, iexToolbars, uCloseTabSheet, ieview, iexColorButton,

iexUserInteractions, System.UITypes, System.Classes, ShellA-PI;

type

TMainForm = class(TForm)

MainMenu1: TMainMenu;

miFileMenu: TMenuItem;

miFileNew: TMenuItem;

miFileOpen: TMenuItem;

miFileClose: TMenuItem;

miHelpMenu: TMenuItem;

miExit: TMenuItem;

miAbout: TMenuItem;

miFileSave: TMenuItem;

miFileSaveAs: TMenuItem;

miEditMenu: TMenuItem;

miCut: TMenuItem;

miCopy: TMenuItem;

miPaste: TMenuItem;

pnlToolbar: TPanel;

btnOpen: TSpeedButton;

btnSave: TSpeedButton;

btnCut: TSpeedButton;

btnCopy: TSpeedButton;

btnPaste: TSpeedButton;

btnNew: TSpeedButton;

miToolsMenu: TMenuItem;

miBackground: TMenuItem;

N2: TMenuItem;

miNegative: TMenuItem;

miGrayscale: TMenuItem;

miHorizontalflip: TMenuItem;

miVerticalflip: TMenuItem;

N6: TMenuItem;

miUndo: TMenuItem;

miImageMenu: TMenuItem;

N7: TMenuItem;

btnUndo: TSpeedButton;

pnlStatus: TPanel;

btnRectSelect: TSpeedButton;

N8: TMenuItem;

btnDefault: TSpeedButton;

btnEllipticalSelect: TSpeedButton;

miCropToSel: TMenuItem;

btnLassoSelect: TSpeedButton;

btnRedo: TSpeedButton;

miRedo: TMenuItem;

btnCropTool: TSpeedButton;

pgcEditor: TPageControl;

ProgressBar1: TProgressBar;

lblStatus: TLabel;

OpenImageEnDialog1: TOpenImageEnDialog;

SaveImageEnDialog1: TSaveImageEnDialog;

btnZoom100: TSpeedButton;

btnZoomIn: TSpeedButton;

btnZoomOut: TSpeedButton;

btnSmudgeTool: TSpeedButton;

btnCloneTool: TSpeedButton;

btnRotateTool: TSpeedButton;

updBrush: TUpDown;

edtBrush: TEdit;

miRetouchTools: TMenuItem;

miClone: TMenuItem;

sepRT: TMenuItem;

miSmudge: TMenuItem;

miBlur: TMenuItem;

miSharpen: TMenuItem;

miSmooth: TMenuItem;

miPixelize: TMenuItem;

miWave: TMenuItem;

miViewMenu: TMenuItem;

N12: TMenuItem;

N14: TMenuItem;

btnSelectZoom: TSpeedButton;

miPixelize2: TMenuItem;

miSmooth2: TMenuItem;

btnBrushTool: TSpeedButton;

btnEraserTool: TSpeedButton;

btnColorPicker: TSpeedButton;

btnBrushColor: TIEColorButton;

miBrushTools: TMenuItem;

miSolidBrush: TMenuItem;

sepBR: TMenuItem;

miEraser: TMenuItem;

miAddText: TMenuItem;

SelectFont1: TMenuItem;

N15: TMenuItem;

FontDialog1: TFontDialog;

bevel2: TPaintBox;

bevel1: TPaintBox;

bevel3: TPaintBox;

bevel5: TPaintBox;

bevel4: TPaintBox;

miZoom10: TMenuItem;

miZoom25: TMenuItem;

miZoom50: TMenuItem;

miZoom75: TMenuItem;

miZoom100: TMenuItem;

miZoom200: TMenuItem;

miZoom500: TMenuItem;

ZoomtoFit1: TMenuItem;

N16: TMenuItem;

miSelectMenu: TMenuItem;

miRectangularSelect: TMenuItem;

miCircularSelect: TMenuItem;

miLassoSelect: TMenuItem;

sepSL: TMenuItem;

miSelZoom: TMenuItem;

miScrollView: TMenuItem;

miEditingToolsMenu: TMenuItem;

miRotateTl: TMenuItem;

miCropTl: TMenuItem;

btnShapeTool: TSpeedButton;

miAddShapeMenu: TMenuItem;

N18: TMenuItem;

miArrowDown: TMenuItem;

miArrowLeft: TMenuItem;

miArrowRight: TMenuItem;

miArrowUp: TMenuItem;

miCloud: TMenuItem;

miEllipse: TMenuItem;

miExplosion: TMenuItem;

miHeart: TMenuItem;

miLightningLeft: TMenuItem;

miRectangle: TMenuItem;

miSpeechBubble: TMenuItem;

miStar5: TMenuItem;

miTriangle: TMenuItem;

sepSH: TMenuItem;

miShapeBorder: TMenuItem;

miSelectColor: TMenuItem;

btnFillTool: TSpeedButton;

miFillTool: TMenuItem;

N19: TMenuItem;

bevel8: TPaintBox;

trkZoom: TTrackBar;

Help: TMenuItem;

procedure AddShapeClick(Sender: TObject);

procedure BevelPaint(Sender: TObject);

procedure BrushClick(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure miFileNewClick(Sender: TObject);

procedure miFileCloseClick(Sender: TObject);

procedure miFileOpenClick(Sender: TObject);

procedure miExitClick(Sender: TObject);

procedure miFileSaveClick(Sender: TObject);

procedure miFileSaveAsClick(Sender: TObject);

procedure miCutClick(Sender: TObject);

procedure miCopyClick(Sender: TObject);

procedure miPasteClick(Sender: TObject);

procedure miAboutClick(Sender: TObject);

procedure miPasteToRectClick(Sender: TObject);

procedure trkZoomChange(Sender: TObject);

procedure miBackgroundClick(Sender: TObject);

procedure miNegativeClick(Sender: TObject);

procedure miGrayscaleClick(Sender: TObject);

procedure miVerticalflipClick(Sender: TObject);

procedure miHorizontalflipClick(Sender: TObject);

procedure miUndoClick(Sender: TObject);

procedure MouseButtonClick(Sender: TObject);

procedure miCropToSelClick(Sender: TObject);

procedure miRedoClick(Sender: TObject);

procedure btnZoom100Click(Sender: TObject);

procedure btnZoomInClick(Sender: TObject);

procedure btnZoomOutClick(Sender: TObject);

procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);

procedure FormShow(Sender: TObject);

procedure btnBrushColorChange(Sender: TObject);

procedure miAddTextClick(Sender: TObject);

procedure miBrushToolsClick(Sender: TObject);

procedure miRetouchToolsClick(Sender: TObject);

procedure miRetouchClick(Sender: TObject);

procedure pgcEditorChange(Sender: TObject);

procedure miPixelize2Click(Sender: TObject);

procedure miSmooth2Click(Sender: TObject);

procedure SelectFont1Click(Sender: TObject);

procedure MenuClick(Sender: TObject);

procedure miAddShapeMenuClick(Sender: TObject);

procedure miEDGrayScaleClick(Sender: TObject);

procedure miFillToolClick(Sender: TObject);

procedure miEditingToolsMenuClick(Sender: TObject);

procedure miPixelsClick(Sender: TObject);

procedure miEditingToolClick(Sender: TObject);

procedure miSelectColorClick(Sender: TObject);

procedure miSelectMenuClick(Sender: TObject);

procedure miShapeBorderClick(Sender: TObject);

procedure SelectClick(Sender: TObject);

procedure updBrushChanging(Sender: TObject; var Al-lowChange: Boolean);

procedure ZoomClick(Sender: TObject);

procedure HelpClick(Sender: TObject);

private

{ Private declarations }

fCurrentRetouchMode: TIERetouchMode;

fCurrentBrushFill: TIEBrushFill;

fCurrentShape: TIEShape;

fCurrentBorderSize: Integer;

fClosePageControlHandler: TClosePageControlHandler;

procedure UpdateStatusCaption(Sender: TObject);

procedure SaveCurrentEditor();

function CreateEditorTab(const Name: string; NewImage: boolean): TImageEnView;

procedure TabSheetCloseQuery(Sender: TObject; var Can-Close: Boolean);

procedure ImageEnViewViewChange(Sender: TObject; Change: Integer);

procedure ImageEnViewImageChange(Sender: TObject);

procedure ImageEnViewProgress(Sender: TObject; per: Inte-ger);

procedure ImageEnViewFinishWork(Sender: TObject);

procedure ImageEnViewUserInteraction(Sender: TObject; Event: TIEUserInteractionEvent; Info: Integer);

procedure ImageEnViewNewLayer(Sender: TObject; LayerIdx: integer; LayerKind: TIELayerKind);

procedure ImageEnViewLayerNotifyEx(Sender: TObject; lay-er: Integer; event: TIELayerEvent);

public

{ Public declarations }

function ActiveImageEnView(): TImageEnView;

function GetImageEnView(Index: Integer): TImageEnView;

procedure UpdateButtonStatus(UpdateMI: Boolean = False);

end;

var

MainForm: TMainForm;

implementation

{$R \*.DFM}

uses

Loading, About;

{$R demores.res}

const

DEMORES\_PNG\_CLOSE\_BUTTON = 3004;

DEMORES\_PNG\_BRUSH\_IMAGE = 3005;

function TMainForm.CreateEditorTab(const Name: string; NewImage: boolean): TImageEnView;

var

tab: TCloseTabSheet;

begin

tab := TCloseTabSheet.Create(pgcEditor);

tab.Caption := ExtractFilename(Name);

tab.OnCloseQuery := TabSheetCloseQuery;

tab.OnClose := pgcEditorChange;

Result := TImageEnView.Create(tab);

Result.Parent := tab;

Result.Align := alClient;

Result.SelColor1 := clBlack;

Result.SelColor2 := clWhite;

Result.Background := $00606060;

Result.SetChessboardStyle(6, bsSolid);

Result.SetSelectionGripStyle(clBlack, clWhite, bsClear, 3, true);

Result.ZoomFilter := rfFastLinear;

Result.MouseInteractGeneral := [miSelect];

Result.BrushTool.BrushColor := btnBrushColor.SelectedColor;

Result.FillTool.ColorFillValue := btnBrushColor.SelectedColor;

Result.CloneTool.BrushSize := updBrush.Position;

Result.RetouchTool.BrushSize := updBrush.Position;

Result.BrushTool.BrushSize := updBrush.Position;

Result.Proc.UndoLimit := 20;

Result.Proc.PreviewsParams := [prppDefaultLockPreview];

Result.OnViewChange := ImageEnViewViewChange;

Result.OnImageChange := ImageEnViewImageChange;

Result.OnSelectionChange := ImageEnViewImageChange;

Result.OnProgress := ImageEnViewProgress;

Result.OnFinishWork := ImageEnViewFinishWork;

Result.OnNewLayer := ImageEnViewNewLayer;

Result.OnLayerNotifyEx := ImageEnViewLayerNotifyEx;

if pgcEditor.PageCount > 0 then

begin

tab.PageControl := pgcEditor;

tab.PageIndex := pgcEditor.ActivePageIndex + 1;

pgcEditor.ActivePageIndex := tab.PageIndex;

end

else

tab.PageControl := pgcEditor;

if (not NewImage) and FileExists(Name) then

begin

Result.IO.AutoAdjustDPI := true;

Result.IO.LoadFromFile(Name, True);

end

else

begin

Result.Proc.ImageResize(700, 500);

Result.Proc.ClearUndo;

Result.IO.Params.Filename := Name;

Result.IEBitmap.Fill(clWhite);

end;

Result.IEBitmap.Modified := false;

UpdateButtonStatus(True);

end;

procedure TMainForm.TabSheetCloseQuery(Sender: TObject; var CanClose: Boolean);

var

iev: TImageEnView;

begin

iev := nil;

if ( TCloseTabSheet( Sender ).ControlCount > 0 ) then

iev := TImageEnView( TCloseTabSheet( Sender ).Controls[0] );

if assigned( iev ) and iev.IEBitmap.Modified then

begin

pgcEditor.ActivePageIndex := TCloseTabSheet( Sender ).PageIndex;

case MessageDlg( format( 'Сохранить изменения в файл %s?', [ ExtractFilename( iev.IO.Params.Filename )]), mtConfirmation, [mbYes, mbNo, mbCancel], 0) of

mrYes : miFileSaveAsClick( nil );

mrCancel : CanClose := false;

end;

end;

end;

procedure TMainForm.ImageEnViewViewChange(Sender: TObject; Change: Integer);

begin

UpdateButtonStatus();

end;

procedure TMainForm.ImageEnViewImageChange(Sender: TObject);

begin

UpdateButtonStatus();

end;

procedure TMainForm.ImageEnViewProgress(Sender: TObject; per: Integer);

begin

ProgressBar1.Position := per;

ProgressBar1.Visible := True;

end;

procedure TMainForm.ImageEnViewFinishWork(Sender: TObject);

begin

ProgressBar1.Visible := False;

end;

procedure TMainForm.ImageEnViewUserInteraction(Sender: TObject; Event: TIEUserInteractionEvent; Info: Integer);

begin

if Event = ieiColorPickerClick then

begin

btnBrushColor.SelectedColor := TColor( Info );

btnBrushColorChange(nil);

btnBrushTool.Down := True;

MouseButtonClick(nil);

end;

end;

procedure TMainForm.ImageEnViewNewLayer(Sender: TObject; LayerIdx: integer; LayerKind: TIELayerKind);

begin

TImageEnView(Sender).Layers[LayerIdx].FillColor := btnBrushColor.SelectedColor;

TImageEnView(Sender).Layers[LayerIdx].BorderWidth := fCur-rentBorderSize;

if LayerKind = ielkShape then

TIEShapeLayer( TImageEnView(Sender).Layers[LayerIdx] ).Shape := fCurrentShape;

end;

procedure TMainForm.ImageEnViewLayerNotifyEx(Sender: TObject; layer: Integer; event: TIELayerEvent);

const

Auto\_Merge\_Layers = True;

begin

if Auto\_Merge\_Layers and ( layer > 0 ) then

begin

if (( TImageEnView(Sender).Layers[layer].Kind = ielkText ) and ( event = ielEdited )) or

(( TImageEnView(Sender).Layers[layer].Kind <> ielkText ) and ( event = ielCreated )) then

TImageEnView(Sender).LayersMerge( 0, layer );

end;

end;

procedure TMainForm.FormCreate(Sender: TObject);

var

I: Integer;

begin

fClosePageControlHandler := TClosePageControlHandler.Create( pgcEditor );

fCurrentRetouchMode := iermSmudge;

fCurrentBrushFill := iebfSolid;

Application.OnHint := UpdateStatusCaption;

for i := 0 to pnlToolbar.ControlCount - 1 do

if pnlToolbar.Controls[i] is TSpeedButton then

TSpeedButton( pnlToolbar.Controls[i] ).Flat := True;

pnlToolbar.ParentBackground := False;

pnlStatus.ParentBackground := False;

end;

procedure TMainForm.miFileNewClick(Sender: TObject);

begin

CreateEditorTab('New' + IntToStr(pgcEditor.PageCount + 1) + '.jpg', true);

end;

procedure TMainForm.miFileOpenClick(Sender: TObject);

begin

if OpenImageEnDialog1.Execute then

CreateEditorTab(OpenImageEnDialog1.FileName, false);

end;

procedure TMainForm.miFileCloseClick(Sender: TObject);

begin

if pgcEditor.ActivePage <> nil then

TCloseTabSheet( pgcEditor.ActivePage ).Close;

end;

procedure TMainForm.miExitClick(Sender: TObject);

begin

Close;

end;

procedure TMainForm.miCutClick(Sender: TObject);

begin

ActiveImageEnView().Proc.CutToClipboard(iecpSelection);

end;

procedure TMainForm.miCopyClick(Sender: TObject);

begin

ActiveImageEnView().Proc.CopyToClipboard(iecpAuto);

UpdateButtonStatus();

end;

procedure TMainForm.miPasteClick(Sender: TObject);

begin

ActiveImageEnView().Deselect;

ActiveImageEnView().Proc.PasteFromClipboard;

end;

procedure TMainForm.UpdateButtonStatus(UpdateMI: Boolean = False);

var

v: boolean;

iev: TImageEnView;

{}

procedure \_EnableAll(MI: TMenuItem; v: boolean);

var

i: Integer;

begin

for i := 0 to MI.Count - 1 do

MI.Items[i].Enabled := v;

end;

{}

begin

iev := GetImageEnView( pgcEditor.ActivePageIndex );

v := iev <> nil;

btnSave .Enabled := v;

btnCut .Enabled := v;

btnCopy .Enabled := v;

btnPaste .Enabled := v and iev.Proc.CanPasteFromClipboard();

btnZoom100 .Enabled := v;

btnZoomIn .Enabled := v;

btnZoomOut .Enabled := v;

trkZoom .Enabled := v;

if v then

trkZoom .Position := Trunc(iev.Zoom \* 10);

btnDefault .Enabled := v;

btnSelectZoom .Enabled := v;

btnRectSelect .Enabled := v;

btnEllipticalSelect .Enabled := v;

btnLassoSelect .Enabled := v;

if v then

updBrush .Position := iev.CloneTool.BrushSize;

btnBrushTool .Enabled := v;

btnEraserTool .Enabled := v;

btnColorPicker .Enabled := v;

btnBrushColor .Enabled := v;

if v then

btnBrushColor .SelectedColor := iev.BrushTool.BrushColor;

btnUndo .Enabled := v and iev.Proc.CanUndo;

btnRedo .Enabled := v and iev.Proc.CanRedo;

miFileOpen .Enabled := v;

miFileClose .Enabled := v;

miFileSave .Enabled := v;

miFileSaveAs .Enabled := v;

miUndo .Enabled := v and iev.Proc.CanUndo;

miRedo .Enabled := v and iev.Proc.CanRedo;

miCut .Enabled := v;

miCopy .Enabled := v;

miPaste .Enabled := v and iev.Proc.CanPasteFromClipboard();

miCropToSel .Enabled := v and iev.Selected;

\_EnableAll( miImageMenu, v );

\_EnableAll( miToolsMenu, v );

miAddText .Enabled := v and iev.Selected;

\_EnableAll( miViewMenu, v );

if UpdateMI then

MouseButtonClick(Self);

UpdateStatusCaption(nil);

end;

procedure TMainForm.UpdateStatusCaption(Sender: TObject);

var

cap: string;

begin

cap := Application.Hint;

if ( cap = '' ) and ( GetImageEnView( pgcEditor.ActivePageIndex ) <> nil ) then

cap := format( '%s (%d%%)', [ ExtractFileName( Ac-tiveImageEnView().IO.Params.FileName ), Round( Ac-tiveImageEnView().Zoom )] );

lblStatus.Caption := ' ' + cap;

end;

procedure TMainForm.miAboutClick(Sender: TObject);

var

AboutForm: TAboutForm;

begin

AboutForm := TAboutForm.create(nil);

try

AboutForm.ShowModal;

finally

AboutForm.free;

end;

end;

procedure TMainForm.miPasteToRectClick(Sender: TObject);

begin

ActiveImageEnView().Proc.PasteFromClipboard(iecpSelection);

end;

procedure TMainForm.trkZoomChange(Sender: TObject);

begin

ActiveImageEnView().Zoom := trkZoom.Position / 10;

end;

procedure TMainForm.miFileSaveAsClick(Sender: TObject);

begin

SaveImageEnDialog1.Filename := ActiveImageEn-View().IO.Params.FileName;

SaveImageEnDialog1.AttachedImageEnIO := ActiveImageEn-View().IO;

if SaveImageEnDialog1.Execute then

begin

ActiveImageEnView().IO.Params.FileName := SaveImageEnDialog1.FileName;

SaveCurrentEditor();

end;

end;

procedure TMainForm.miFileSaveClick(Sender: TObject);

begin

if ( ExtractFilePath( ActiveImageEnView().IO.Params.Filename ) = '' ) or

( ExtractFileExt( ActiveImageEnView().IO.Params.Filename ) = '' ) then

miFileSaveAsClick(nil)

else

SaveCurrentEditor();

end;

procedure TMainForm.SaveCurrentEditor();

begin

ActiveImageEnView().IO.SaveToFile( ActiveImageEn-View().IO.Params.Filename );

ActiveImageEnView().IEBitmap.Modified := false;

end;

procedure TMainForm.miBackgroundClick(Sender: TObject);

var

ColorDialog: TColorDialog;

begin

ColorDialog := TColorDialog.Create(nil);

try

ColorDialog.Color := ActiveImageEnView().Background;

if ColorDialog.Execute then

begin

ActiveImageEnView().Background := ColorDialog.Color;

end;

finally

ColorDialog.Free;

end;

end;

procedure TMainForm.miNegativeClick(Sender: TObject);

begin

ActiveImageEnView().Proc.Negative;

end;

procedure TMainForm.miGrayscaleClick(Sender: TObject);

begin

ActiveImageEnView().Proc.ConvertToGray;

end;

procedure TMainForm.miVerticalflipClick(Sender: TObject);

begin

ActiveImageEnView().Proc.Flip(fdVertical);

end;

procedure TMainForm.miHorizontalflipClick(Sender: TObject);

begin

ActiveImageEnView().Proc.Flip(fdHorizontal);

end;

procedure TMainForm.miUndoClick(Sender: TObject);

begin

ActiveImageEnView().Proc.Undo( True );

MainForm.UpdateButtonStatus();

end;

procedure TMainForm.MouseButtonClick(Sender: TObject);

begin

if GetImageEnView( pgcEditor.ActivePageIndex ) = nil then

exit;

if btnRectSelect.down then

begin

ActiveImageEnView().MouseInteractGeneral := [miSelect];

end

else

if btnEllipticalSelect.down then

begin

ActiveImageEnView().MouseInteractGeneral := [miSelectCir-cle];

end

else

if btnDefault.down then

begin

ActiveImageEnView().MouseInteractGeneral := [miScroll];

end

else

if btnLassoSelect.Down then

begin

ActiveImageEnView().Deselect;

ActiveImageEnView().MouseInteractGeneral := [miSelectLas-so];

end

else

if btnSelectZoom.down then

begin

ActiveImageEnView().MouseInteractGeneral := [miSe-lectZoom];

end

else

if btnCropTool.down then

begin

ActiveImageEnView().MouseInteractGeneral := [ miCropTool ];

end

else

if btnSmudgeTool.down then

begin

ActiveImageEnView().MouseInteractGeneral := [ miRetouch-Tool ];

if Sender = btnSmudgeTool then

fCurrentRetouchMode := iermSmudge;

ActiveImageEnView().RetouchTool.RetouchMode := fCurren-tRetouchMode;

end

else

if btnCloneTool.down then

begin

ActiveImageEnView().MouseInteractGeneral := [ miCloneTool ];

end

else

if btnFillTool.down then

begin

ActiveImageEnView().MouseInteractGeneral := [ miColorFill ];

end

else

if btnBrushTool.down then

begin

if ActiveImageEnView().BrushTool.BrushImage.IsEmpty then

ActiveImageEn-View().BrushTool.BrushImage.LoadFromResource( HInstance, DEMORES\_PNG\_BRUSH\_IMAGE, RT\_RCDATA, ioPNG );

ActiveImageEnView().MouseInteractGeneral := [ miBrushTool ];

if ( Sender = btnBrushTool ) or ( fCurrentBrushFill in [ ieb-fEraser, iebfSmartEraser ]) then

fCurrentBrushFill := iebfSolid;

ActiveImageEnView().BrushTool.BrushFill := fCurrentBrush-Fill;

end

else

if btnEraserTool.down then

begin

ActiveImageEnView().MouseInteractGeneral := [ miBrushTool ];

if ( Sender = btnEraserTool ) or not ( fCurrentBrushFill in [ iebfEraser, iebfSmartEraser ]) then

fCurrentBrushFill := iebfEraser;

ActiveImageEnView().BrushTool.BrushFill := fCurrentBrush-Fill;

end

else

if btnRotateTool.down then

begin

ActiveImageEnView().MouseInteractGeneral := [ miRotateTool ];

end

else

if btnShapeTool.down then

begin

ActiveImageEnView().MouseInteractLayers := [ mlCreate-ShapeLayers ];

end

else

if btnColorPicker.down then

begin

ActiveImageEnView().MouseInteractGeneral := [ miColorPick-er ];

ActiveImageEnView().FillTool.ColorSelectActions := [ iec-cBrushColor, ieccFillColor ];

ActiveImageEnView().OnUserInteraction := ImageEnViewU-serInteraction;

end;

end;

procedure TMainForm.miCropToSelClick(Sender: TObject);

begin

ActiveImageEnView().Proc.CropSel();

ActiveImageEnView().Deselect();

end;

procedure TMainForm.miRedoClick(Sender: TObject);

begin

ActiveImageEnView().Proc.Redo( True );

MainForm.UpdateButtonStatus();

end;

procedure TMainForm.BrushClick(Sender: TObject);

begin

fCurrentBrushFill := TIEBrushFill( TMenuItem( Sender ).Tag );

if fCurrentBrushFill in [ iebfEraser ] then

btnEraserTool.Down := True

else

btnBrushTool.Down := True;

MouseButtonClick( nil );

end;

procedure TMainForm.btnZoom100Click(Sender: TObject);

begin

if Round( ActiveImageEnView().Zoom ) = 100 then

ActiveImageEnView().Fit()

else

ActiveImageEnView().Zoom := 100;

end;

procedure TMainForm.btnZoomInClick(Sender: TObject);

begin

ActiveImageEnView().ZoomIn();

end;

procedure TMainForm.btnZoomOutClick(Sender: TObject);

begin

ActiveImageEnView().ZoomOut();

end;

function TMainForm.ActiveImageEnView(): TImageEnView;

begin

Result := GetImageEnView( pgcEditor.ActivePageIndex );

if Result = nil then

raise Exception.create( 'Изображение отсутствует.' );

end;

procedure TMainForm.BevelPaint(Sender: TObject);

const

Bevel\_Color = $00A0A0A0;

begin

with TPaintBox(Sender).Canvas do

begin

Pen.Color := Bevel\_Color;

MoveTo( TPaintBox(Sender).Width div 2, 0 );

LineTo( TPaintBox(Sender).Width div 2, TPaint-Box(Sender).Height );

end;

end;

function TMainForm.GetImageEnView(Index: Integer): TImageEnView;

begin

Result := nil;

if ( Index >= 0 ) and ( Index < pgcEditor.PageCount ) and

( pgcEditor.Pages[Index].ControlCount > 0 ) then

Result := pgcEditor.Pages[Index].Controls[0] as TImageEn-View;

end;

procedure TMainForm.HelpClick(Sender: TObject);

begin

ShellExe-cute(0,PChar('Open'),Pchar('Help.chm'),nil,nil,SW\_SHOW);

end;

procedure TMainForm.FormCloseQuery(Sender: TObject; var CanClose: Boolean);

var

I: Integer;

begin

CanClose := True;

for I := pgcEditor.PageCount - 1 downto 0 do

if not TCloseTabSheet( pgcEditor.Pages[i] ).Close then

begin

CanClose := False;

Break;

end;

end;

procedure TMainForm.FormShow(Sender: TObject);

{}

procedure CreateEditorTabFromFile(const Name: string; const FilePath: string);

var

iev: TImageEnView;

begin

iev := CreateEditorTab(Name, True);

iev.IO.LoadFromFile(FilePath);

iev.SetScale(350, 100, ieuMillimeters);

iev.Update();

end;

{}

begin

LoadingForm.ShowModal;

CreateEditorTabFromFile('Bridge.jpg', 'Изображения\Bridge.jpg');

CreateEditorTabFromFile('Plants.jpg', 'Изображения\Plants.jpg');

CreateEditorTabFromFile('Bird.jpg', 'Изображения\Bird.jpg');

UpdateButtonStatus();

end;

procedure TMainForm.btnBrushColorChange(Sender: TObject);

begin

ActiveImageEnView().BrushTool.BrushColor := btnBrush-Color.SelectedColor;

ActiveImageEnView().FillTool.ColorFillValue := btnBrushCol-or.SelectedColor;

end;

procedure TMainForm.miAddShapeMenuClick(Sender: TObject);

var

shp, i: Integer;

begin

miArrowDown .Tag := ord( iesArrowDown );

miArrowLeft .Tag := ord( iesArrowLeft );

miArrowRight .Tag := ord( iesArrowRight );

miArrowUp .Tag := ord( iesArrowUp );

miCloud .Tag := ord( iesCloud );

miEllipse .Tag := ord( iesEllipse );

miExplosion .Tag := ord( iesExplosion );

miHeart .Tag := ord( iesHeart );

miLightningLeft.Tag := ord( iesLightningLeft );

miRectangle .Tag := ord( iesRectangle );

miSpeechBubble .Tag := ord( iesSpeechBubbleLeftOutShort );

miStar5 .Tag := ord( iesStar5 );

miTriangle .Tag := ord( iesTriangle );

miShapeBorder.Tag := -99;

miSelectColor.Tag := -99;

sepSH .Tag := -99;

if GetImageEnView( pgcEditor.ActivePageIndex ) = nil then

exit;

shp := -1;

if ActiveImageEnView().MouseInteractLayers = [ mlCreate-ShapeLayers ] then

shp := ord( fCurrentShape );

for i := 0 to miAddShapeMenu.Count - 1 do

miAddShapeMenu.Items[i].Checked := miAddShape-Menu.Items[i].tag = shp;

miShapeBorder.Checked := fCurrentBorderSize > 0;

end;

procedure TMainForm.AddShapeClick(Sender: TObject);

begin

fCurrentShape := TIEShape( TMenuItem(Sender).Tag );

btnShapeTool.Down := True;

MouseButtonClick( nil );

end;

procedure TMainForm.MenuClick(Sender: TObject);

var

z, i: Integer;

iev: TImageEnView;

begin

if sender = miViewMenu then

begin

z := 0;

iev := GetImageEnView( pgcEditor.ActivePageIndex );

if iev <> nil then

begin

if Round( iev.Zoom ) = Round( iev.GetIdealZoom ) then

z := -99

else

z := Round( iev.Zoom );

end;

for i := 0 to miViewMenu.Count - 1 do

if miViewMenu[i].tag <> 0 then

miViewMenu.Items[i].Checked := miViewMenu.Items[i].tag = z;

end;

UpdateButtonStatus();

end;

procedure TMainForm.miShapeBorderClick(Sender: TObject);

begin

if fCurrentBorderSize = 1 then

fCurrentBorderSize := 0

else

fCurrentBorderSize := 1;

end;

procedure TMainForm.miAddTextClick(Sender: TObject);

var

txt: string;

begin

txt := 'Введите текст';

if InputQuery('Добавить текст', 'Текст:', txt ) then

ActiveImageEnView().Proc.TextOut( ActiveImageEn-View().SelX1, ActiveImageEnView().SelY1, txt, FontDia-log1.Font );

end;

procedure TMainForm.SelectFont1Click(Sender: TObject);

begin

FontDialog1.Execute();

end;

procedure TMainForm.miBrushToolsClick(Sender: TObject);

var

bfMode, i: Integer;

begin

miSolidBrush.Tag := ord( iebfSolid );

miEraser.Tag := ord( iebfEraser );

if GetImageEnView( pgcEditor.ActivePageIndex ) = nil then

exit;

bfMode := -1;

if ActiveImageEnView().MouseInteractGeneral = [ miBrushTool ] then

bfMode := ord( ActiveImageEnView().BrushTool.BrushFill );

for I := 0 to miBrushTools.Count - 1 do

if miBrushTools[i] <> sepBR then

miBrushTools.Items[i].Checked := miBrushTools.Items[i].tag = bfMode;

miFillTool.Checked := ActiveImageEn-View().MouseInteractGeneral = [ miColorFill ];

end;

procedure TMainForm.miEDGrayScaleClick(Sender: TObject);

begin

ActiveImageEnView().Proc.EdgeDetect\_Sobel();

end;

procedure TMainForm.miFillToolClick(Sender: TObject);

begin

btnFillTool.Down := True;

MouseButtonClick(nil)

end;

procedure TMainForm.miEditingToolsMenuClick(Sender: TObject);

var

iev: TImageEnView;

begin

iev := GetImageEnView( pgcEditor.ActivePageIndex );

if iev = nil then

exit;

miRotateTl .Checked := miRotateTool in iev.MouseInteractGeneral;

miCropTl .Checked := ( miCropTool in iev.MouseInteractGeneral ) and ( iev.CropTool.Mode = iectmRECTANGLE );

end;

procedure TMainForm.miRetouchToolsClick(Sender: TObject);

var

rtMode, i: Integer;

begin

miSmudge .Tag := ord( iermSmudge );

miBlur .Tag := ord( iermBlur );

miSharpen .Tag := ord( iermSharpen );

miSmooth .Tag := ord( iermSmooth );

miPixelize .Tag := ord( iermPixelize );

miWave .Tag := ord( iermWave );

if GetImageEnView( pgcEditor.ActivePageIndex ) = nil then

exit;

miClone.Checked := ActiveImageEn-View().MouseInteractGeneral = [ miCloneTool ];

rtMode := -1;

if ActiveImageEnView().MouseInteractGeneral = [ miRetouch-Tool ] then

rtMode := ord( ActiveImageEn-View().RetouchTool.RetouchMode );

for I := 0 to miRetouchTools.Count - 1 do

if ( miRetouchTools.Items[i] <> miClone ) and ( miRetouch-Tools[i] <> sepRT ) then

miRetouchTools.Items[i].Checked := miRetouch-Tools.Items[i].tag = rtMode;

end;

procedure TMainForm.miRetouchClick(Sender: TObject);

begin

if sender = miClone then

btnCloneTool.Down := True

else

begin

btnSmudgeTool.Down := True;

fCurrentRetouchMode := TIERetouchMode( TMenuItem( Sender ).Tag );

end;

MouseButtonClick( nil );

end;

procedure TMainForm.pgcEditorChange(Sender: TObject);

begin

UpdateButtonStatus(True);

end;

procedure TMainForm.miPixelize2Click(Sender: TObject);

begin

ActiveImageEnView().Proc.Pixelize();

end;

procedure TMainForm.miPixelsClick(Sender: TObject);

begin

IEGlobalSettings().DefaultMeasureUnit := TIEUnits( TMenu-Item( Sender) .Tag );

end;

procedure TMainForm.miEditingToolClick(Sender: TObject);

begin

if Sender = miRotateTl then

begin

btnRotateTool.Down := True;

end

else

if Sender = miCropTl then

begin

btnCropTool.Down := True;

ActiveImageEnView().CropTool.Mode := iectmRECTANGLE;

end

else

MouseButtonClick( nil );

end;

procedure TMainForm.miSelectColorClick(Sender: TObject);

begin

btnBrushColor.PromptForColor(True);

end;

procedure TMainForm.miSmooth2Click(Sender: TObject);

begin

ActiveImageEnView().Proc.SymmetricNearestNeighbour();

end;

procedure TMainForm.miSelectMenuClick(Sender: TObject);

begin

if GetImageEnView( pgcEditor.ActivePageIndex ) = nil then

exit;

miRectangularSelect.Checked := miSelect in ActiveImageEn-View().MouseInteractGeneral;

miCircularSelect .Checked := miSelectCircle in Ac-tiveImageEnView().MouseInteractGeneral;

miLassoSelect .Checked := miSelectLasso in Ac-tiveImageEnView().MouseInteractGeneral;

miSelZoom .Checked := miSelectZoom in Ac-tiveImageEnView().MouseInteractGeneral;

miScrollView .Checked := miScroll in ActiveImageEn-View().MouseInteractGeneral;

end;

procedure TMainForm.SelectClick(Sender: TObject);

begin

if Sender = miRectangularSelect then

btnRectSelect.Down := True

else

if Sender = miCircularSelect then

btnEllipticalSelect.Down := True

else

if Sender = miLassoSelect then

btnLassoSelect.Down := True

else

if Sender = miSelZoom then

btnSelectZoom.Down := True

else

if Sender = miScrollView then

btnDefault.Down := True;

MouseButtonClick( nil );

end;

procedure TMainForm.updBrushChanging(Sender: TObject; var AllowChange: Boolean);

begin

ActiveImageEnView().CloneTool .BrushSize := up-dBrush.Position;

ActiveImageEnView().RetouchTool.BrushSize := up-dBrush.Position;

ActiveImageEnView().BrushTool .BrushSize := up-dBrush.Position;

end;

procedure TMainForm.ZoomClick(Sender: TObject);

var

z: Integer;

begin

z := TMenuItem( Sender ).Tag;

if z > 1 then

ActiveImageEnView().Zoom := z

else

ActiveImageEnView().Fit();

end;

end.

unit About;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, Sys-tem.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, Vcl.StdCtrls,

Vcl.Imaging.pngimage;

type

TAboutForm = class(TForm)

Panel1: TPanel;

Author: TLabel;

GGPK: TLabel;

GGPKLink: TLabel;

Logo: TImage;

LogoText: TImage;

procedure GGPKLinkClick(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

AboutForm: TAboutForm;

implementation

uses

ShellAPI;

{$R \*.dfm}

procedure TAboutForm.GGPKLinkClick(Sender: TObject);

begin

ShellExecute(Handle, 'open', PChar('http://ggpk.by/'), nil, nil, SW\_MAXIMIZE);

end;

end.