# ПОЛНЫЙ НАБОР ИСХОДНЫХ ТЕКСТОВ СЕРВЕРА

## ОГЛАВЛЕНИЕ

[ПОЛНЫЙ НАБОР ИСХОДНЫХ ТЕКСТОВ СЕРВЕРА 1](#_Toc485545014)

[ОГЛАВЛЕНИЕ 1](#_Toc485545015)

[Main, образ экрана 1](#_Toc485545016)

[unit Main; 1](#_Toc485545017)

[unit Common1; 11](#_Toc485545018)

[unit SrviceLibrary1; 12](#_Toc485545019)

[unit Sensor01; 14](#_Toc485545020)

[SensorGroup01, образ экрана 20](#_Toc485545021)

[unit SensorGroup01; 20](#_Toc485545022)

[SensorGroup02, образ экрана 22](#_Toc485545023)

[unit SensorGroup02; 23](#_Toc485545024)

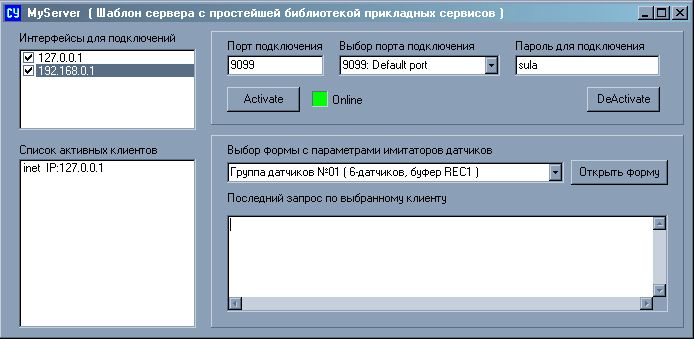
[unit Horizon01; 24](#_Toc485545025)

[SensorHorizon01, образ экрана 33](#_Toc485545026)

[unit SensorHorizon01; 33](#_Toc485545027)

[unit Service01; 34](#_Toc485545028)

## Main, образ экрана



## unit Main;

// ==========================================================================

// TCP-СЕРВЕР И БИБЛИОТЕКА ПРИКЛАДНЫХ СЕРВИСОВ

// ==========================================================================

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, CheckLst, ComCtrls, ExtCtrls,

// Юниты обслуживания сервера

IdComponent, IdThreadMgr, IdThreadMgrDefault,

IdBaseComponent, IdTCPServer,

// Юниты обслуживания сервера, добавляемые в ручную

IdSocketHandle, IdStack, IdGlobal, IdResourceStrings,

// Прикладная часть

Common1, SrviceLibrary1,

SensorGroup01, SensorGroup02, SensorHorizon01;

type

TServerForm = class(TForm)

TCPServer: TIdTCPServer;

ThreadMgrDefault: TIdThreadMgrDefault;

chLboxIPs : TCheckListBox;

lstboxAllClients: TListBox;

Panel1: TPanel;

edPort: TEdit;

Button1: TButton;

cbboxViewPorts: TComboBox;

StaticText1: TStaticText;

Button2: TButton;

edPSW: TEdit;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

Panel2: TPanel;

Label5: TLabel;

Label6: TLabel;

Memo1: TMemo;

Label7: TLabel;

CbBoxSensorGroup: TComboBox;

btnOpenSensorGroup: TButton;

Label8: TLabel;

procedure TCPServerDisconnect(AThread: TIdPeerThread);

procedure TCPServerExecute(AThread: TIdPeerThread);

procedure TCPServerConnect(AThread: TIdPeerThread);

procedure cbboxViewPortsChange(Sender: TObject);

procedure Button1Click(Sender: TObject);

procedure Button2Click(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure lstboxAllClientsClick(Sender: TObject);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure btnOpenSensorGroupClick(Sender: TObject);

private

{ Private declarations }

// По номеру порта получить описание соответствующей службы

function PortDescription(const PortNumber: integer): string;

// Сформировать список IP-адресов сетевых карт

procedure PopulateIPAddresses();

// Обновление подключений сервера к Socket на выбранный порт.

function UpdateBindings(RqListIP : TCheckListBox;

RqPort : string) : boolean;

// Получить количество активных клиентов (коннектов).

function GetClientCount() : integer;

// Выполнить Disconnect для указанного клиета

procedure DisconnectClient(AThread: TIdPeerThread);

// Запрос на принудительное отключение всех клиентов

function RqDisconnectAllClients() : boolean;

// Обновить список агентов клиентов на сервере

procedure UpdateClientsList (RqList : TListBox;

RqReport : TMemo);

// Активация и деактивация сервера.

function ServerActivate(RqActivate : boolean) : boolean;

public

{ Public declarations }

end;

var

ServerForm: TServerForm;

implementation

{$R \*.dfm}

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

// Реплики и команды выдаваемые сервером

const cRepTimeOut = 20; // Тайаут для рандеву сервера и клиента

cFieldSep = ':'; // Разделитель полей в командах и квитанциях

cRepYES = 'YES'; // Код позитивной квитанции от сервера

cRepNOT = 'NOT'; // Код негативной квитанции от сервера

cCmdEND = 'END'; // Код команды отключения клиента

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

// Команды выдаваемые клиентом

const cCmdConnect = 'CNN'; // Код команды подключения к серверу

cCmdDisConnect = 'DCN'; // Код команды отключения от сервера

cCmdLink = 'LNK'; // Код команды запроса на подтверждение коннекта

cCmdService = 'SRV'; // Код команды к сервисам обслуживания клиента

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

// Запросы выдаваемые клиентом

const cCmdGET = 'GET'; // Запрос на блок данных

cCmdSET = 'SET'; // Запрос на конфигурацию сервера или сервисов

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

// Агент клиента на сервере

type TAgent = class(TObject)

AThread : TIdPeerThread; // Нить сервера, которая обслуживает клиента

IP : string; // IP - адрес клиента

Port : integer; // Port - клиента

Name : string; // Имя клиента

Password : string; // Пароль клиента

ClientMsg : string; // Сообщение от клиента серверу

ServerMsg : string; // Сообщение сервера клиенту

end;

// ==========================================================================

// СЕГМЕНТ СЕРВИСНЫХ ОПЕРАЦИЙ

// ==========================================================================

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

// 24.11.2016

// Вырезать из текста команды очередное поле

// Пример 1

// На входе Cmd = 'field1:field2:field3'

// На выходе Result = 'field1', Cmd = 'field2:field3'

// Пример 2

// На входе Cmd = 'field3'

// На выходе Result = 'field3', Cmd = ''

function CutNextCmdField(var Cmd : string) : string;

var wPos : integer;

wStr : string;

begin

wStr := Trim(Cmd);

Result := wStr;

wPos := pos(cFieldSep, wStr);

if wPos > 0

then begin

Result := copy(wStr, 1, wPos - 1);

if (Length(wStr) > wPos)

then Cmd := copy(wStr, wPos + 1, Length(wStr))

end

else Cmd := '';

end;

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

// 21.11.2016

// По номеру порта получить описание соответствующей (Berkeley standard) службы

// Используются юниты: IdStack

function TServerForm.PortDescription(const PortNumber: integer): string;

var ST : TStrings;

begin

// Создать объект, описывающий один стандартный (определенный IANA) порт,

// прочитанный из файла windows\system32\drivers\etc\services

ST := GStack.WSGetServByPort(PortNumber);

// Проитать имя TCP - службы из данного объекта

with ST do

try

Result := '';

if Count > 0

then begin

Result := IntToStr(PortNumber) + ': ' + Strings[0];

end;

finally

Free;

end;

end;

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

// 21.11.2016

// Сформировать список IP-адресов сетевых карт в chboxIPs,

// а также сформировать в cbboxViewPorts список номеров и

// описаний стандартных служб на этих портах.

// Используются юниты: IdGlobal, IdStack, IdResourceStrings

procedure TServerForm.PopulateIPAddresses();

const DefaultPort = '9099';

var Ind : integer;

begin

with chLboxIPs do

begin

// Сформировать список IP-адресов сетевых карт

Clear;

Items := GStack.LocalAddresses; // IP-адреса сетевых карт (TStrings)

Items.Insert(0, '127.0.0.1'); // Добавить внутренний IP-адрес

end;

// Сформировать описания для стандартных портов

cbboxViewPorts.Items.BeginUpdate;

cbboxViewPorts.Items.Add(DefaultPort + ': Default port');

try

// cbboxViewPorts.Items.Add(RSBindingAny);

for Ind := 0 to IdPorts.Count - 1 do

// Прочитать службу очередного TCP порта

cbboxViewPorts.Items.Add(PortDescription(Integer(IdPorts[Ind])));

finally

cbboxViewPorts.Items.EndUpdate;

end;

// Установить значение порта по умолчанию

cbboxViewPorts.ItemIndex := 0;

edPort.Text := DefaultPort;

end;

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

// 21.11.2016

// Изменить номер порта в edtPort

procedure TServerForm.cbboxViewPortsChange(Sender: TObject);

// Выделить номер порта из строки

function GetPort(AString : String) : String;

begin

Result := AString;

if pos(':', AString) > 0

then Result := copy(AString, 1, pos(':', AString) - 1);

end;

begin

edPort.Text := GetPort(cbboxViewPorts.Items.Strings[cbboxViewPorts.ItemIndex]);

end;

// ==========================================================================

// СЕГМЕНТ ОПЕРАЦИЙ ПО ИНИЦИАЛИЗАЦИИ СЕРВЕРА

// ==========================================================================

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

// 22.11.2016

// Обновление подключений сервера к Socket на выбранный порт.

function TServerForm.UpdateBindings(RqListIP : TCheckListBox;

RqPort : string) : boolean;

var

Binding : TIdSocketHandle; // Ссылка на объект подключений

Port : integer; // Порт для подключений сервера

BindCount : integer; // Счетчик выполненных подключений

Ind : integer;

begin

Result := False;

// Входной контроль состояния сервера

// Операции с bindings допускаются только при неактивном сервере

if TCPServer.Active then Exit;

// Входной контроль номера порта

Port := -1;

try

Port := StrToInt(Trim(RqPort));

except

MessageDlg('Порт должен быть положительным целым числом',

mtError, [mbOk], 0);

end;

if Port < 0 then Exit;

// Обновление bindings

if not RqListIP.Count > 0 then Exit;

BindCount := 0;

try

// Очистить bindings

TCPServer.Bindings.Clear;

for Ind := 0 to RqListIP.Count-1 do

if RqListIP.Checked[Ind]

then begin

// Добавить очередное подключение

Binding := TCPServer.Bindings.Add;

Binding.IP := RqListIP.Items.Strings[Ind];

Binding.Port := Port;

Inc(BindCount);

end;

if BindCount > 0

then Result := True

else MessageDlg('Не выбраны IP для подключениий сервера.',

mtError, [mbOk], 0);;

except

MessageDlg('Не могу обновить порт подключениий сервера.',

mtError, [mbOk], 0);

end;

end;

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

// 22.11.2016

// Получить количество активных клиентов (коннектов).

function TServerForm.GetClientCount() : integer;

var List : TList;

begin

Result := - 1;

try

// Получить ссылку на список нитей подключенных клиентов

List := TCPServer.Threads.LockList;

if Assigned(List) then Result := List.Count;

finally

// Разблокировать список

TCPServer.Threads.UnlockList;

end;

end;

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

// 22.11.2016

// Активация и деактивация сервера.

function TServerForm.ServerActivate(RqActivate : boolean) : boolean;

var Deactivate : boolean;

begin

Result := True;

if RqActivate

then begin

// Активация сервера

if TCPServer.Active then Exit;

// Обновление подключений сервера на выбранный порт.

if not UpdateBindings(chLboxIPs, edPort.Text) then Exit;

// Попробуем активировать

try

TCPServer.Active := True;

except

Result := False;

MessageDlg('Ошибка АКТИВАЦИИ сервера.',

mtError, [mbOk], 0);

end;

end

else begin

// Деактивация сервера

if not TCPServer.Active then Exit;

// Проверить наличие подключенных клиентов

Deactivate := True;

if GetClientCount() > 0

then Deactivate := RqDisconnectAllClients();

// Начать деактивацию

if Deactivate

then begin

try

TCPServer.Active := False;

except

Result := False;

MessageDlg('Ошибка при ДЕАКТИВАЦИИ сервера.',

mtError, [mbOk], 0);

end;

end;

end;

end;

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

// Обновить список агентов на сервере

procedure TServerForm.UpdateClientsList (RqList : TListBox;

RqReport : TMemo);

var wList : TList;

AThread : TIdPeerThread;

wAgent : TAgent;

Ind : integer;

begin

RqList.Clear;

RqReport.Clear;

try

// Получить ссылку на список нитей для подключенных клиентов

wList := TCPServer.Threads.LockList;

if Assigned(wList)

then begin

for Ind := 0 to wList.Count -1

do begin

AThread := TIdPeerThread(wList.Items[Ind]);

wAgent := TAgent(AThread.Data);

if Assigned(wAgent)

then RqList.Items.AddObject(wAgent.Name

+ ' IP:'

+ wAgent.IP,

wAgent);

end;

end;

finally

// Разблокировать список

TCPServer.Threads.UnlockList;

end;

end;

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

procedure TServerForm.TCPServerConnect(AThread: TIdPeerThread);

var wAgent : TAgent;

wMsg : string;

wCmd : string;

wName : string;

wPSW : string;

begin

wMsg := '';

wMsg := AThread.Connection.ReadLn('',cRepTimeOut);

wCmd := CutNextCmdField(wMsg);

wName := CutNextCmdField(wMsg);

wPSW := CutNextCmdField(wMsg);

if (wCmd <> cCmdConnect) or (wName = '') or (wPSW = '')

then begin

// Отправить клиенту негативную квитанцию

AThread.Connection.WriteLn(cRepNOT + cFieldSep

+ 'Некорректная команда подключения');

Exit;

end;

if wPSW = edPSW.Text // Проверить пароль на подключение

then begin

// Создать объект клиентского агента на сервере

wAgent := TAgent.Create;

// Связать агента и нить сервера взаимными ссылками

AThread.Data := wAgent;

wAgent.AThread := AThread;

// Заполнить системные поля агента ( IP и Port клиента )

wAgent.IP := AThread.Connection.Socket.Binding.PeerIP;

wAgent.Port := AThread.Connection.Socket.Binding.PeerPort;

wAgent.Name := wName;

wAgent.Password := wPSW;

wAgent.ClientMsg := 'Запрос на подключение от ' + wName;

// Сохранить и отправить клиенту позитивную квитанцию

wAgent.ServerMsg := cRepYES + cFieldSep

+ 'Успешное подключение к серверу';

AThread.Connection.WriteLn(wAgent.ServerMsg);

// Обновить список агентов на сервере

UpdateClientsList(lstboxAllClients, Memo1);

end

else begin

// Отправить клиенту негативную квитанцию

AThread.Connection.WriteLn(cRepNOT + cFieldSep

+ 'Неправильный пароль');

AThread.Connection.Disconnect;

end;

end;

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

// 22.11.2016

// Выполнить Disconnect для указанного клиета

procedure TServerForm.DisconnectClient(AThread: TIdPeerThread);

var Agent : TAgent;

begin

if not Assigned(AThread) then Exit;

Agent := TAgent(AThread.Data);

if Assigned(Agent)

then begin

AThread.Data := nil;

Agent.Free;

end;

AThread.Connection.WriteLn(cCmdEND);

AThread.Connection.Disconnect;

end;

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

// 22.11.2016

// Запрос на принудительное отключение всех клиентов

function TServerForm.RqDisconnectAllClients() : boolean;

var List : TList;

AThread : TIdPeerThread;

Ind : integer;

begin

Result := False;

if MessageDlg('На сервере присутствуют подключенные клиенты.'

+ #13#10 + ' Отключить их принудительно?',

mtConfirmation, [mbYes, mbNo], 0) = mrYes

then begin

try

// Получить ссылку на список нитей подключенных клиентов

List := TCPServer.Threads.LockList;

if Assigned(List)

then begin

for Ind := 0 to List.Count -1

do begin

AThread := TIdPeerThread(List.Items[Ind]);

DisconnectClient(AThread);

end;

end;

finally

// Разблокировать список

TCPServer.Threads.UnlockList;

end;

// Обновить список агентов на сервере

UpdateClientsList(lstboxAllClients, Memo1);

Result := True;

end;

end;

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

// ==========================================================================

// ОБСЛУЖИВАНИЕ ЗАЯВОК КЛИЕНТА

// ==========================================================================

// 24.11.2016

// Обработчик события "Клиент выдал заявку на обслуживание"

procedure TServerForm.TCPServerExecute(AThread: TIdPeerThread);

var wAgent : TAgent;

wMsgSRC : string;

wMsg : string;

wSysCmd : string;

wUserCmd : string;

wUCmdPrm : string;

wRep : string;

wRequest : TRequest; // Запрос к диспетчеру прикладных сервисов

begin

// Получить ссылку на агента

wAgent := TAgent(AThread.Data);

// Прочитать полный текст заявки от клиента

wMsgSRC := AThread.Connection.ReadLn;

wMsg := wMsgSRC;

// Выделить в заявке системную командную часть

wSysCmd := CutNextCmdField(wMsg);

// Сохранить полный текст заявки от клиента

if wSysCmd <> cCmdLink

then if Assigned(wAgent)then wAgent.ClientMsg := 'Запрос : ' + wMsgSRC;

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

// Запрос от клиента на подтверждение коннекта

if wSysCmd = cCmdLink

then begin

// Выдать клиенту позитивную квитанцию

wRep := cRepYES + cFieldSep + 'Сервер подтверждает коннект';

AThread.Connection.WriteLn(wRep);

Exit;

end;

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

// Запрос от клиента на дисконнект

if wSysCmd = cCmdDisConnect

then begin

// Выдать клиенту позитивную квитанцию

wRep := cRepYES + cFieldSep + 'Сервер принял запрос на дисконнект';

AThread.Connection.WriteLn(wRep);

// Выполнить дисконнект клиента

DisconnectClient(AThread);

// Обновить список агентов на сервере

UpdateClientsList(lstboxAllClients, Memo1);

Exit;

end;

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

// ===========================================

// Запрос от клиента на получение сервиса

if wSysCmd = cCmdService

then begin

// Выделить в заявке код команды пользователя

wUserCmd := CutNextCmdField(wMsg);

// Выделить в заявке параметры команды пользователя

wUCmdPrm := CutNextCmdField(wMsg);

// Подготовить запрос на прикладное обслуживание

wRequest.RqCmd := wUCmdPrm;

// Выполнить прикладное обслуживание

if UserDispatch(wRequest)

then begin

// Отправить клиенту позитивную квитанцию

wRep := cRepYES;

AThread.Connection.WriteLn(wRep);

// Если клиент запросил на блок данных ( cCmdGET )

if wUserCmd = cCmdGET

then begin

// Отправить клиенту результат обслуживания

AThread.Connection.WriteBuffer(wRequest.Buffer^,

wRequest.BufSize,

True);

end;

end

else begin

// Отправить клиенту негативную квитанцию

wRep := cRepNOT;

AThread.Connection.WriteLn(wRep);

end;

// Сохранить последнее сообщение клиенту от сервера

if Assigned(wAgent)then wAgent.ServerMsg := 'Квитанция : ' + wRep;

end;

// ===========================================

end;

// ==========================================================================

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

// 24.11.2016

// Обработчик события - отключение клиента

procedure TServerForm.TCPServerDisconnect(AThread: TIdPeerThread);

begin

// Обновить список агентов на сервере

UpdateClientsList(lstboxAllClients, Memo1);

end;

// ==========================================================================

// ИНИЦИИРОВАНИЕ / ТЕРМИНИРОВАНИЕ

// ==========================================================================

procedure TServerForm.FormCreate(Sender: TObject);

begin

PopulateIPAddresses();

end;

// ==========================================================================

// УПРАВЛЕНИЕ СЕРВЕРОМ

// ==========================================================================

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

// Активация сервера

procedure TServerForm.Button1Click(Sender: TObject);

begin

ServerActivate(True);

if TCPServer.Active then StaticText1.Color := clLime;

end;

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

// Деактивация сервера

procedure TServerForm.Button2Click(Sender: TObject);

begin

ServerActivate(False);

if not TCPServer.Active then StaticText1.Color := clBtnFace;

end;

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

// Отобразить последнюю операцию выбранного клиента

procedure TServerForm.lstboxAllClientsClick(Sender: TObject);

var Ind : integer;

wAgent : TAgent;

begin

Ind := lstboxAllClients.ItemIndex;

wAgent := TAgent(lstboxAllClients.Items.Objects[Ind]);

Memo1.Clear;

if Assigned(wAgent)

then begin

Memo1.Lines.Add(wAgent.ClientMsg);

Memo1.Lines.Add(wAgent.ServerMsg);

end;

end;

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

// Обслживание попыток терминировать сервер при подключенных клиентах

procedure TServerForm.FormClose(Sender: TObject; var Action: TCloseAction);

begin

if (GetClientCount() > 0) or (TCPServer.Active)

then begin

Action := caNone;

MessageDlg('Вы не можете завершить приложение'

+ #13#10 + 'при наличии активных клиентов или'

+ #13#10 + 'пока сервер активиен( Online )...',

mtInformation, [mbOk], 0);

end;

end;

// ==========================================================================

// РАБОТА С ФОРМАМИ ИМИТАТОРОВ ДАТЧИКОВ

// ==========================================================================

// Открыть выбранную форму для работы с имитаторами датчиков

procedure TServerForm.btnOpenSensorGroupClick(Sender: TObject);

begin

case CbBoxSensorGroup.ItemIndex of

0 : SensorGroup01Form.Show;

1 : SensorGroup02Form.Show;

2 : SensorHorizonForm.Show;

end;

end;

// ==========================================================================

// КОНЕЦ

// ==========================================================================

end.

## unit Common1;

interface

uses

Windows, Forms;

// ==========================================================================

// БУФЕРА ДЛЯ ПРИКЛАДНОЙ ЧАСТИ

// ==========================================================================

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

const REC1MaxIndx = 5;

// Структура запмси в буфере результатов выполнения команды 'GET:REC1'

type TREC1 = packed record

Id : string[8];

Arr : array[0..REC1MaxIndx] of extended;

end;

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

const REC2MaxIndx = 3;

// Структура запмси в буфере результатов выполнения команды 'GET:REC2'

type TREC2 = packed record

Id : string[8];

Arr : array[0..REC2MaxIndx] of extended;

end;

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

// Структура запмси в буфере результатов выполнения команды 'GET:REC3'

type TREC3 = packed record

Id : string[8];

Pith : extended;

Roll : extended;

end;

// ==========================================================================

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

// Директория старта приложения

var ApplicationDirectory : string;

// ==========================================================================

// БУФЕРА РЕЗУЛЬТАТОВ ВЫПОЛНЕНИЯ ПРИКЛАДНЫХ СЕРВИСОВ

// ==========================================================================

var REC1 : TREC1; // Буфер результатов команды 'REC1'

REC2 : TREC2; // Буфер результатов команды 'REC2'

REC3 : TREC3; // Буфер результатов команды 'REC3'

// ==========================================================================

implementation

// ==========================================================================

// ==========================================================================

// КОНЕЦ

// ==========================================================================

end.

## unit SrviceLibrary1;

interface

uses SysUtils, Common1,

SensorGroup01, SensorGroup02, SensorHorizon01;

// ==========================================================================

// СТРУКТУРА ЗАПРОСА К ПРИКЛАДНЫМ СЕРВИСАМ

// ==========================================================================

type TRequest = record

// Входные параметры

RqCmd : string; // Код комманды на обслуживание

// Выходные параметры

Buffer : pointer; // Адрес буфера или nil

BufSize : integer; // Размер буфера в байтах

end;

// ==========================================================================

// БУФЕРА РЕЗУЛЬТАТОВ ВЫПОЛНЕНИЯ ПРИКЛАДНЫХ СЕРВИСОВ

// ==========================================================================

// см. unit Common1

// ==========================================================================

// ДИСПЕТЧЕР ПРИКЛАДНЫХ СЕРВИСОВ

// ==========================================================================

// Диспетчер прикладных сервисов

function UserDispatch(var Request : TRequest) : boolean;

// ==========================================================================

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

implementation

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

// ==========================================================================

// ==========================================================================

// ДИСПЕТЧЕР И ПРИКЛАДНЫЕ СЕРВИСЫ

// ==========================================================================

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

// Выполнить команду 'REC1'

procedure RunREC1();

var Ind : integer;

begin

Randomize;

// Полная очистка буфера

FillChar(REC1, SizeOf(REC1), #0);

// Запишем идентификатор буфера

REC1.Id := 'REC1';

// Запишем текущие значения

for Ind := Low(REC1.Arr) to High(REC1.Arr)

do begin

if Ind > High(SensorsArr01) then Exit;

if Assigned(SensorsArr01[Ind])

then REC1.Arr[Ind] := SensorsArr01[Ind].Value;

end;

end;

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

// Выполнить команду 'REC2'

procedure RunREC2();

var Ind : integer;

begin

Randomize;

// Полная очистка буфера

FillChar(REC2, SizeOf(REC2), #0);

// Запишем идентификатор буфера

REC2.Id := 'REC2';

// Запишем текущие значения

for Ind := Low(REC2.Arr) to High(REC2.Arr)

do begin

if Ind > High(SensorsArr02) then Exit;

if Assigned(SensorsArr02[Ind])

then REC2.Arr[Ind] := SensorsArr02[Ind].Value;

end;

end;

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

// Выполнить команду 'REC3'

procedure RunREC3();

begin

// Полная очистка буфера

FillChar(REC3, SizeOf(REC3), #0);

// Запишем идентификатор буфера

REC3.Id := 'REC3';

// Запишем текущие значения

if Assigned(Horizon)

then begin

REC3.Pith := Horizon.Pitch;

REC3.Roll := Horizon.Roll;

end;

end;

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

// Диспетчер прикладных сервисов

function UserDispatch(var Request : TRequest) : boolean;

begin

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

// Начальные установки

Result := False;

Request.Buffer := nil;

Request.BufSize := 0;

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

// Диспретчирование

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

// Команда 'REC1'

if UpperCase(Request.RqCmd) = 'REC1'

then begin

// Выполнение прикладного сервиса

RunREC1();

// Отчет прикладного сервиса по запросу

Request.Buffer := Addr(REC1);

Request.BufSize := SizeOF(REC1);

Result := True;

Exit;

end;

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

// Команда 'REC2'

if UpperCase(Request.RqCmd) = 'REC2'

then begin

// Выполнение прикладного сервиса

RunREC2();

// Отчет прикладного сервиса по запросу

Request.Buffer := Addr(REC2);

Request.BufSize := SizeOF(REC2);

Result := True;

Exit;

end;

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

// Команда 'REC3'

if UpperCase(Request.RqCmd) = 'REC3'

then begin

// Выполнение прикладного сервиса

RunREC3();

// Отчет прикладного сервиса по запросу

Request.Buffer := Addr(REC3);

Request.BufSize := SizeOF(REC3);

Result := True;

Exit;

end;

end;

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

// ==========================================================================

// КОНЕЦ

// ==========================================================================

end.

## unit Sensor01;

// Данный юнит описывает класс для динамического создания

// объектов имитации сенсоров (датчиков).

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, ComCtrls, ExtCtrls;

// ================================================================

// Класс для объектов имитации сенсоров (датчиков)

type TSensor = class(TObject)

private

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

fPanel : TPanel; // Динамическая несущая панель сенсора

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

fLbIndx : TLabel; // Подпись индекса сенсора

fLbTitle : TLabel; // Подпись названия сенсора

fSTxtIndx : TStaticText; // Индекс сенсора

fEdTitle : TEdit; // Название сенсора

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

fLbMin : TLabel; // Поспись минимального значения сенсора

fLbValue : TLabel; // Подпись текущего значения сенсора

fLbMax : TLabel; // Поспись максимального значения сенсора

fEdMin : TEdit; // Минимальное значение сенсора

fSTxtValue : TStaticText; // Текущее значение сенсора

fEdMax : TEdit; // Максимальное значение сенсора

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

fTrackBar : TTrackBar; // Регулятор значений сенсора

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

fChBoxNPr : TCheckBox; // Выключатель дополнительного шума

fEdNPr : TEdit; // Процент дополнительного шума

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

// ОБРАБОТЧИКИ СОБЫТИЙ НА КОМПОНЕНТАХ

procedure EdMinChange(Sender: TObject);

procedure EdMaxChange(Sender: TObject);

procedure TrackBarChange(Sender: TObject);

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

// ОБСЛУЖИВАНИЕ СВОЙСТВ

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

procedure SetIndx(Indx : integer);

function GetIndx() : integer;

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

procedure SetTitle(Title : string);

function GetTitle() : string;

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

procedure SetMin(Min : integer);

function GetMin() : integer;

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

procedure SetMax(Max : integer);

function GetMax() : integer;

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

function GetValue() : integer;

public

constructor Create(RqWinControl : TWinControl; XB,YB : integer);

procedure Free;

property Indx : integer read GetIndx write SetIndx;

property Title : string read GetTitle write SetTitle;

property Min : integer read GetMin write SetMin;

property Max : integer read GetMax write SetMax;

property Value : integer read GetValue;

end;

// ================================================================

// ================================================================

implementation

// ================================================================

// ================================================================

// ================================================================

// ОБЩИЙ ПРОЦЕДУРНЫЙ СЕРВИС

// ================================================================

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

// Преобразование RqEdit.Text в целое число со знаком

function TryEditToInt(RqEdit : TEdit; var Value : integer) : boolean;

begin

Result := False;

try

if Trim(RqEdit.Text) = ''

then Value := 0

else begin

Value := StrToInt(Trim(RqEdit.Text));

RqEdit.Color := clWindow;

end;

Result := True;

except

RqEdit.Color := RGB(255,240,240);

ShowMessage('Не могу преобразовать в целое число со знаком');

end;

end;

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

// Программно создать компонент, являющийся потомком TControl.

// Функция CreateControl запрашивает ссылку на class (тип) того

// компонента, которого необходимо создать в контейнере RqOwner

// (Например, форме Form1).

function CreateControl(ControlClass : TControlClass;

RqOwner : TWinControl;

const ComponentName : string;

X, Y, W, H : Integer): TControl;

begin

Result := ControlClass.Create(RqOwner);

with Result do

begin

Parent := RqOwner; // Ссылка на объект контейнер

Name := ComponentName; // Переопределить имя компонента

SetBounds(X, Y, W, H); // Наследуется из TWinControl

Visible := True; // Требование отображать компонент

end;

end;

// ================================================================

// ИНИЦИИРОВАНИЕ / ТЕРМИНИРОВАНИЕ

// ================================================================

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

// Создать объект сенсора

constructor TSensor.Create(

RqWinControl : TWinControl; // Ссылка на форму или другой контейнер

XB, YB : integer); // Left и Top на контейнере

begin

inherited Create;

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

// Создать несущую панель для данного сенсора

fPanel:= TPanel(CreateControl(TPanel, RqWinControl, '',

XB,YB,240,150));

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

// Создать компоненты на несущей панели данного сенсора

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

// ИНДЕКС ДАТЧИКА

fLbIndx := TLabel(CreateControl(TLabel, fPanel, '',

12,8,40,20));

fLbIndx.Caption := 'Индекс';

fSTxtIndx := TStaticText(CreateControl(TStaticText, fPanel, '',

12, 26, 40, 20));

fSTxtIndx.Alignment := taCenter;

fSTxtIndx.AutoSize := False;

fSTxtIndx.BorderStyle := sbsSunken;

fSTxtIndx.Font.Style := [fsBold];

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

// НАЗВАНИЕ ДАТЧИКА

fLbTitle := TLabel(CreateControl(TLabel, fPanel, '',

56,8,40,20));

fLbTitle.Caption := 'Название датчика';

fEdTitle := TEdit(CreateControl(TEdit, fPanel, '',

56, 26, 170, 20));

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

// МИНИМАЛЬНОЕ ЗНАЧЕНИ

fEdMin := TEdit(CreateControl(TEdit, fPanel, '',

12,66,70,20));

fEdMin.Text := '-100';

fEdMin.OnChange := EdMinChange;

fLbMin := TLabel(CreateControl(TLabel, fPanel, '',

12,48,70,20));

fLbMin.Caption := 'Min';

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

// ТЕКУЩЕЕ ЗНАЧЕНИЕ

fSTxtValue := TStaticText(CreateControl(TStaticText, fPanel, '',

84,66,70,20));

fSTxtValue.Alignment := taCenter;

fSTxtValue.AutoSize := False;

fSTxtValue.BorderStyle := sbsSunken;

fSTxtValue.Font.Style := [fsBold];

fSTxtValue.Caption := '0';

fLbValue := TLabel(CreateControl(TLabel, fPanel, '',

84,48,70,20));

fLbValue.Caption := 'Value';

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

// МАКСИМАЛЬНОЕ ЗНАЧЕНИЕ

fEdMax := TEdit(CreateControl(TEdit, fPanel, '',

156,66,70,20));

fEdMax.Text := '100';

fEdMax.OnChange := EdMaxChange;

fLbMax := TLabel(CreateControl(TLabel, fPanel, '',

156,48,70,20));

fLbMax.Caption := 'Max';

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

// РЕГУЛЯТОР ЗНАЧЕНИЙ

fTrackBar := TTrackBar(CreateControl(TTrackBar, fPanel, '',

4, 88, 230, 24));

fTrackBar.Min := -100;

fTrackBar.Max := 100;

fTrackBar.Position := 0;

fTrackBar.OnChange := TrackBarChange;

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

// ДОПОЛНИТЕЛЬНЫЙ СЛУЧАЙНЫЙ ШУМ

fChBoxNPr := TCheckBox(CreateControl(TCheckBox, fPanel, '',

12, 118, 170, 20));

fChBoxNPr.Checked := True;

fChBoxNPr.Caption := 'Добавить шум в процентах :';

fEdNPr := TEdit(CreateControl(TEdit, fPanel, '',

180,118,40,20));

fEdNPr.Text := '10';

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

// Инициализировать генератор случайных чисел

Randomize;

end;

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

// Удалить объект сенсора

procedure TSensor.Free;

begin

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

fLbIndx.Free;

fLbTitle.Free;

fSTxtIndx.Free;

fEdTitle.Free;

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

fLbMin.Free;

fLbValue.Free;

fLbMax.Free;

fEdMin.Free;

fSTxtValue.Free;

fEdMax.Free;

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

fTrackBar.Free;

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

fChBoxNPr.Free;

fEdNPr.Free;

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

fPanel.Free;

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

inherited Free;

end;

// ================================================================

// ОБРАБОТЧИКИ СОБЫТИЙ НА КОМПОНЕНТАХ СЕНСОРА

// ================================================================

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

// Изменение минимального значения сенсора

procedure TSensor.EdMinChange(Sender: TObject);

var Value : integer;

begin

if TryEditToInt(fEdMin, Value)

then fTrackBar.Min := Value;

end;

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

// Изменение максимального значения сенсора

procedure TSensor.EdMaxChange(Sender: TObject);

var Value : integer;

begin

if TryEditToInt(fEdMax, Value)

then fTrackBar.Max := Value;

end;

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

// Изменение текущего значения сенсора

procedure TSensor.TrackBarChange(Sender: TObject);

begin

fSTxtValue.Caption := IntToStr(fTrackBar.Position);

end;

// ================================================================

// ОБСЛУЖИВАНИЕ СВОЙСТВ

// ================================================================

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

// Запись индекса сенсора (для индексации сенсоров в массивах)

procedure TSensor.SetIndx(Indx : integer);

begin

fSTxtIndx.Tag := Indx;

fSTxtIndx.Caption := IntToStr(Indx);

end;

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

// Чтение индекса сенсора (для индексации сенсоров в массивах)

function TSensor.GetIndx() : integer;

begin

Result := fSTxtIndx.Tag;

end;

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

// Запись названия сенсора

procedure TSensor.SetTitle(Title : string);

begin

fEdTitle.Text := Title;

end;

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

// Чтение названия сенсора

function TSensor.GetTitle() : string;

begin

Result := fEdTitle.Text;

end;

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

// Запись минимального значения сенсора

procedure TSensor.SetMin(Min : integer);

begin

if Min < fTrackBar.Max

then begin

fEdMin.Enabled := False;

fTrackBar.Min := Min;

fTrackBar.Position := (fTrackBar.Max - fTrackBar.Min) div 2;

fEdMin.Text := IntToStr(Min);

fEdMin.Enabled := True;

end;

end;

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

// Чтение минимального значения сенсора

function TSensor.GetMin() : integer;

begin

Result := fTrackBar.Min;

end;

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

// Запись максимального значения сенсора

procedure TSensor.SetMax(Max : integer);

begin

if Max > fTrackBar.Min

then begin

fEdMax.Enabled := False;

fTrackBar.Max := Max;

fTrackBar.Position := (fTrackBar.Max - fTrackBar.Min) div 2;

fEdMax.Text := IntToStr(Max);

fEdMax.Enabled := True;

end;

end;

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

// Чтение максимального значения сенсора

function TSensor.GetMax() : integer;

begin

Result := fTrackBar.Max;

end;

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

// Чтение текущего значения сенсора

function TSensor.GetValue() : integer;

var Value : integer;

begin

Result := fTrackBar.Position;

if fChBoxNPr.Checked

then begin

if TryEditToInt(fEdNPr, Value)

then Result := Result + Random(Abs(Round(Result \* Value / 100)));

end;

end;

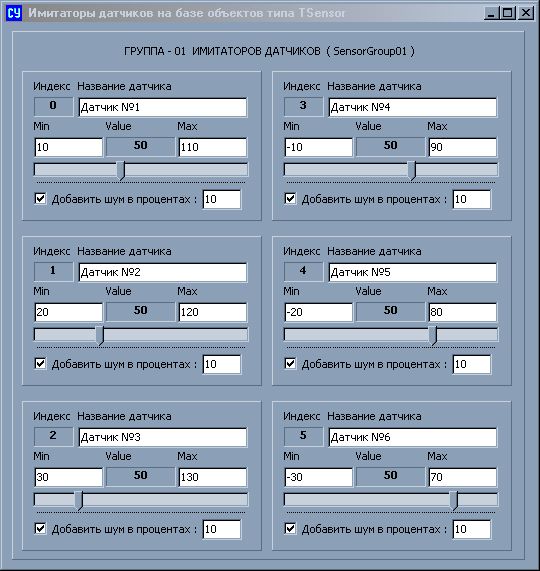
// ================================================================

// КОНЕЦ

// ================================================================

end.

## SensorGroup01, образ экрана



## unit SensorGroup01;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, ComCtrls, ExtCtrls,

Sensor01;

type

TSensorGroup01Form = class(TForm)

Panel1: TPanel;

Label1: TLabel;

procedure FormDestroy(Sender: TObject);

procedure FormCreate(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

SensorGroup01Form: TSensorGroup01Form;

// ================================================================

// ИНТЕРФЕС ДАННЫХ ГРУППЫ СЕНСОРОВ

// ================================================================

// Максимальный индекс сенсора в массиве сенсоров

const Max01SensorIndex = 5;

// Массив сенсоров

var SensorsArr01 : array [0..Max01SensorIndex] of TSensor;

// ================================================================

// ================================================================

implementation

{$R \*.dfm}

// ================================================================

// ================================================================

// ================================================================

// ПОЛНОЕ ОПИСАНИЕ ГРУППЫ СЕНСОРОВ

// ================================================================

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

// Описание начальных установок для конкретного сенсора в группе

type TSensorsDesc = record

XB : integer; // Left - положение сенсора в его контейнере

YB : integer; // Top - положение сенсора в его контейнере

Title : string; // Название для сенсора

Min : integer; // Минимальное значение для сенсора

Max : integer; // Максимальное значение для сенсора

end;

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

// Описание группы сенсоров

const SensorsDesc : array [0..Max01SensorIndex] of TSensorsDesc =

( // Первая колонка

( XB : 10; YB : 40; Title : 'Датчик №1'; Min : 10; Max : 110 ),

( XB : 10; YB : 40 + 165; Title : 'Датчик №2'; Min : 20; Max : 120),

( XB : 10; YB : 40 + 330; Title : 'Датчик №3'; Min : 30; Max : 130),

// Вторая колонка

( XB : 260; YB : 40; Title : 'Датчик №4'; Min : -10; Max : 90),

( XB : 260; YB : 40 + 165; Title : 'Датчик №5'; Min : -20; Max : 80),

( XB : 260; YB : 40 + 330; Title : 'Датчик №6'; Min : -30; Max : 70)

);

// ================================================================

// ИНИЦИИРОВАНИЕ / ТЕРМИНИРОВАНИЕ

// ================================================================

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

// Создать группу из шести сенсоров

procedure TSensorGroup01Form.FormCreate(Sender: TObject);

var Ind : integer;

begin

for Ind := Low(SensorsArr01) to High(SensorsArr01)

do begin

if not Assigned(SensorsArr01[Ind])

then begin

SensorsArr01[Ind] := TSensor.Create(Panel1,

SensorsDesc[Ind].XB,

SensorsDesc[Ind].YB);

with SensorsArr01[Ind]

do begin

Indx := Ind;

Title := SensorsDesc[Ind].Title;

Min := SensorsDesc[Ind].Min;

Max := SensorsDesc[Ind].Max;

end;

end;

end;

// Edit1.Text := IntToStr(AllocMemSize);

end;

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

// Ликвидировать группу из шести сенсоров

procedure TSensorGroup01Form.FormDestroy(Sender: TObject);

var Ind : integer;

begin

for Ind := Low(SensorsArr01) to High(SensorsArr01)

do begin

if Assigned(SensorsArr01[Ind])

then begin

SensorsArr01[Ind].Free;

SensorsArr01[Ind] := nil;

end;

end;

end;

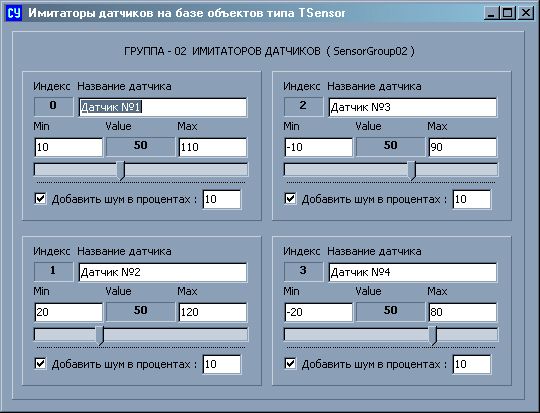
// ================================================================

// КОНЕЦ

// ================================================================

end.

## SensorGroup02, образ экрана



## unit SensorGroup02;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, ComCtrls, ExtCtrls,

Sensor01;

type

TSensorGroup02Form = class(TForm)

Panel1: TPanel;

Label1: TLabel;

procedure FormDestroy(Sender: TObject);

procedure FormCreate(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

SensorGroup02Form: TSensorGroup02Form;

// ================================================================

// ИНТЕРФЕС ДАННЫХ ГРУППЫ СЕНСОРОВ

// ================================================================

// Максимальный индекс сенсора в массиве сенсоров

const Max02SensorIndex = 3;

// Массив сенсоров

var SensorsArr02 : array [0..Max02SensorIndex] of TSensor;

// ================================================================

// ================================================================

implementation

{$R \*.dfm}

// ================================================================

// ================================================================

// ================================================================

// ПОЛНОЕ ОПИСАНИЕ ГРУППЫ СЕНСОРОВ

// ================================================================

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

// Описание начальных установок для конкретного сенсора в группе

type TSensorsDesc = record

XB : integer; // Left - положение сенсора в его контейнере

YB : integer; // Top - положение сенсора в его контейнере

Title : string; // Название для сенсора

Min : integer; // Минимальное значение для сенсора

Max : integer; // Максимальное значение для сенсора

end;

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

// Описание группы сенсоров

const SensorsDesc : array [0..Max02SensorIndex] of TSensorsDesc =

( // Первая колонка

( XB : 10; YB : 40; Title : 'Датчик №1'; Min : 10; Max : 110 ),

( XB : 10; YB : 40 + 165; Title : 'Датчик №2'; Min : 20; Max : 120),

// Вторая колонка

( XB : 260; YB : 40; Title : 'Датчик №3'; Min : -10; Max : 90),

( XB : 260; YB : 40 + 165; Title : 'Датчик №4'; Min : -20; Max : 80)

);

// ================================================================

// ИНИЦИИРОВАНИЕ / ТЕРМИНИРОВАНИЕ

// ================================================================

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

// Создать группу из шести сенсоров

procedure TSensorGroup02Form.FormCreate(Sender: TObject);

var Ind : integer;

begin

for Ind := Low(SensorsArr02) to High(SensorsArr02)

do begin

if not Assigned(SensorsArr02[Ind])

then begin

SensorsArr02[Ind] := TSensor.Create(Panel1,

SensorsDesc[Ind].XB,

SensorsDesc[Ind].YB);

with SensorsArr02[Ind]

do begin

Indx := Ind;

Title := SensorsDesc[Ind].Title;

Min := SensorsDesc[Ind].Min;

Max := SensorsDesc[Ind].Max;

end;

end;

end;

// Edit1.Text := IntToStr(AllocMemSize);

end;

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

// Ликвидировать группу из шести сенсоров

procedure TSensorGroup02Form.FormDestroy(Sender: TObject);

var Ind : integer;

begin

for Ind := Low(SensorsArr02) to High(SensorsArr02)

do begin

if Assigned(SensorsArr02[Ind])

then begin

SensorsArr02[Ind].Free;

SensorsArr02[Ind] := nil;

end;

end;

end;

// ================================================================

// КОНЕЦ

// ================================================================

end.

## unit Horizon01;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, ExtCtrls, StdCtrls, ComCtrls,

Math;

type THorizon = class(Tobject)

private

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

fPanel : TPanel; // Динамическая несущая панель

fImg : TImage;

fXB : integer;

fYB : integer;

fRect : TRect;

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

fTrackRoll : TTrackBar;

fSTxtRoll : TStaticText;

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

fTrackPitch : TTrackBar;

fSTxtPitch : TStaticText;

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

fLbRoll : TLabel;

fLbPitch : TLabel;

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

// Описание контура самолета в полярных координатах

fPolarArr : array of record

L : extended; // Длина вектора

A : extended; // Угол в радианах

end;

// Рабочий массив опорных точек контура самолета

fAirCraft : array of TPoint;

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

// Обработчик изменения текущего значения крена

procedure TrackRollChange(Sender: TObject);

// Обработчик изменения текущего значения тангажа

procedure TrackPitchChange(Sender: TObject);

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

// Построить контур самолета в полярных координатах

procedure DsAirCraftToPolar();

// Очистка индикатора крена

procedure ClearRoll();

// Очистка индикатора тангажа

function ClearPitch() : TRect;

// Прорисовать декартовы оси

procedure ShowAxis();

// Прорисовка угловых маркеров на шкале крена

procedure ShowRollMarks();

// Прорисовка тангажа

procedure ShowPitch(RqPitch : extended);

// Прорисовка контура самолета с учетом крена

procedure ShowAirCraft(RqAngle : extended);

// Полная прорисовка авиагоризонта

procedure ShowHorizon();

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

procedure SetRoll(RqRoll : integer);

function GetRoll() : integer;

procedure SetPitch(RqPitch : integer);

function GetPitch() : integer;

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

public

constructor Create(

RqWinControl : TWinControl; // Ссылка на форму или другой контейнер

XB, YB : integer);

procedure Free();

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

// Угол крена в градусах (-90 .. 90 )

property Roll : integer read GetRoll write SetRoll;

// Угол тангажа в градусах (-89 .. 89 )

property Pitch : integer read GetPitch write SetPitch;

end;

// =========================================================================

// =========================================================================

implementation

// =========================================================================

// =========================================================================

// Параметры шкалы крена подобраны для наилучшей визуализации

const Radius1 = 200; // Радиус шкалы крена

XZero = Radius1; // Left - шкалы крена

YZero = Radius1; // Top - шкалы крена

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

// Описание контура самолета в декартовых координатах

const MaxIndAirCraft = 12;

const DsAirCraft : array[0..MaxIndAirCraft] of TPoint =

(

(X : - Radius1; Y : 0),

(X : - Radius1 + 30; Y : 4),

(X : - 30; Y : 4),

(X : 0; Y : -20),

(X : + 30; Y : 4),

(X : Radius1 - 30; Y : 4),

(X : Radius1; Y : 0),

(X : Radius1 - 30; Y : -4),

(X : + 30; Y : -4),

(X : 0; Y : -28),

(X : - 30; Y : -4),

(X : - Radius1 + 30; Y : -4),

(X : - Radius1; Y : 0)

);

// ================================================================

// ОБЩИЙ ПРОЦЕДУРНЫЙ СЕРВИС

// ================================================================

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

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

// Программно создать компонент, являющийся потомком TControl.

// Функция CreateControl запрашивает ссылку на class (тип) того

// компонента, которого необходимо создать в контейнере RqOwner

// (Например, форме Form1).

function CreateControl(ControlClass : TControlClass;

RqOwner : TWinControl;

const ComponentName : string;

X, Y, W, H : Integer): TControl;

begin

Result := ControlClass.Create(RqOwner);

with Result do

begin

Parent := RqOwner; // Ссылка на объект контейнер

Name := ComponentName; // Переопределить имя компонента

SetBounds(X, Y, W, H); // Наследуется из TWinControl

Visible := True; // Требование отображать компонент

end;

end;

// ================================================================

// ИНИЦИИРОВАНИЕ / ТЕРМИНИРОВАНИЕ

// ================================================================

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

constructor THorizon.Create(

RqWinControl : TWinControl; // Ссылка на форму или другой контейнер

XB, YB : integer);

begin

inherited Create;

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

// Создать несущую панель для данного сенсора

fPanel:= TPanel(CreateControl(TPanel, RqWinControl, '',

XB,YB,512,512));

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

fImg := TImage(CreateControl(TImage, fPanel, '',

4,4,440,440));

fXB := 20;

fYB := 20;

fRect := Rect(fXB,fYB, fXB + 2 \* Radius1,fYB + 2 \* Radius1);

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

// РЕГУЛЯТОР КРЕНА

fTrackRoll := TTrackBar(CreateControl(TTrackBar, fPanel, '',

50, 470, 440, 32));

fTrackRoll.Min := -90;

fTrackRoll.Max := 90;

fTrackRoll.Position := 0;

fTrackRoll.OnChange := TrackRollChange;

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

// ТЕКУЩЕЕ ЗНАЧЕНИЕ КРЕНА

fSTxtRoll := TStaticText(CreateControl(TStaticText, fPanel, '',

10, 472, 40, 16));

fSTxtRoll.Alignment := taCenter;

fSTxtRoll.AutoSize := False;

fSTxtRoll.BorderStyle := sbsSunken;

fSTxtRoll.Font.Style := [fsBold];

fSTxtRoll.Caption := '0';

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

// РЕГУЛЯТОР ТАНГАЖА

fTrackPitch := TTrackBar(CreateControl(TTrackBar, fPanel, '',

464, 32, 32, 410));

fTrackPitch.Orientation := trVertical;

fTrackPitch.Width := 32;

fTrackPitch.Height := 410;

fTrackPitch.Min := -88;

fTrackPitch.Max := 88;

fTrackPitch.Position := 0;

fTrackPitch.OnChange := TrackPitchChange;

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

// ТЕКУЩЕЕ ЗНАЧЕНИЕ ТАНГАЖА

fSTxtPitch := TStaticText(CreateControl(TStaticText, fPanel, '',

464, 16, 40, 16));

fSTxtPitch.Alignment := taCenter;

fSTxtPitch.AutoSize := False;

fSTxtPitch.BorderStyle := sbsSunken;

fSTxtPitch.Font.Style := [fsBold];

fSTxtPitch.Caption := '0';

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

fLbRoll := TLabel(CreateControl(TLabel, fPanel, '',

64,456,58,14));

fLbRoll.AutoSize := True;

fLbRoll.Caption := 'Угол крена в градусах';

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

fLbPitch := TLabel(CreateControl(TLabel, fPanel, '',

456,440,38,14));

fLbPitch.AutoSize := True;

fLbPitch.Caption := 'Тангаж';

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

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

//

DsAirCraftToPolar();

ShowHorizon();

end;

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

procedure THorizon.Free();

begin

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

SetLength(fPolarArr, 0);

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

fLbPitch.Free;

fLbRoll.Free;

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

fSTxtPitch.Free;

fTrackPitch.Free;

fSTxtRoll.Free;

fTrackRoll.Free;

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

fImg.Free;

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

fPanel.Free;

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

inherited Free;

end;

// ================================================================

// ОБРАБОТЧИКИ СОБЫТИЙ НА КОМПОНЕНТАХ

// ================================================================

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

// Изменение текущего значения крена

procedure THorizon.TrackRollChange(Sender: TObject);

begin

fSTxtRoll.Caption := IntToStr(fTrackRoll.Position);

ShowHorizon();

end;

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

// Изменение текущего значения тангажа

procedure THorizon.TrackPitchChange(Sender: TObject);

begin

fSTxtPitch.Caption := IntToStr( -fTrackPitch.Position);

ShowHorizon();

end;

// ================================================================

// ПРИВАТНЫЕ МЕТОДЫ

// ================================================================

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

// Построить контур самолета в полярных координатах

procedure THorizon.DsAirCraftToPolar();

var wAngle : extended;

Ind : integer;

begin

SetLength(fPolarArr, Length(DsAirCraft));

for Ind := Low(fPolarArr) to High(fPolarArr)

do begin

// Вычислим длину вектора

fPolarArr[Ind].L := sqrt(DsAirCraft[Ind].X \* DsAirCraft[Ind].X +

DsAirCraft[Ind].Y \* DsAirCraft[Ind].Y);

// Вычислим угол вектора в радианах

if fPolarArr[Ind].L < 1e-14

then wAngle := 0

else wAngle := arcsin(Abs(DsAirCraft[Ind].Y) / fPolarArr[Ind].L);

// Выполним коррекцию угла по квадрантам

if (DsAirCraft[Ind].X >= 0 )

then begin // I и IV квадранты

if (DsAirCraft[Ind].Y >=0 )

then fPolarArr[Ind].A := wAngle // I квадрант

else fPolarArr[Ind].A := - wAngle; // IV квадрант

end

else begin // II и III квадранты

if (DsAirCraft[Ind].Y >= 0 )

then fPolarArr[Ind].A := Pi - wAngle // II квадрант

else fPolarArr[Ind].A := Pi + wAngle; // III квадрант

end;

end;

end;

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

// Очистка индикатора крена

procedure THorizon.ClearRoll();

begin

with fImg.Canvas

do begin

Brush.Style := bsSolid;

Brush.Color := clWindow;

// Очистка индикатора

Ellipse(fRect);

end;

end;

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

// Очистка индикатора тангажа

function THorizon.ClearPitch() : TRect;

begin

Result := Rect(fXB + 4, fYB + 4,

fXB + 2 \* Radius1 - 4,

fYB + 2 \* Radius1 - 4);

with fImg.Canvas

do begin

Brush.Style := bsSolid;

Brush.Color := clWindow;

// Очистка индикатора

Ellipse(Result);

end;

end;

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

procedure THorizon.ShowHorizon();

begin

// Очистка индикатора крена

ClearRoll();

// Прорисовка шкалы крена

ShowRollMarks();

// Прорисовка тангажа

ShowPitch(fTrackPitch.Position);

// Прорисовка крена

ShowAirCraft(fTrackRoll.Position);

end;

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

// Прорисовка угловых маркеров на шкале крена

procedure THorizon.ShowRollMarks();

const GdRd = Pi/180;

var wSin, wCos : extended;

wPn1 : Tpoint;

wAngle : extended;

Ind : integer;

begin

with fImg.Canvas

do begin

Pen.Width := 1;

Pen.Color := clBlack;

Font.Color := clBlack;

end;

// Прорисовать маркеры градусов

Ind := 0;

repeat

wAngle := Ind \* GdRd;

SinCos(wAngle, wSin, wCos);

wPn1.X := Round(Radius1 \* wCos);

wPn1.Y := Round(Radius1 \* wSin);

with fImg.Canvas

do begin

if (Ind mod 10) = 0

then Pen.Width := 3

else Pen.Width := 1;

// Рисовать маркер

Ellipse(fXB + Radius1 + wPn1.X - 2, fYB + Radius1 - (wPn1.Y - 2),

fXB + Radius1 + wPn1.X + 2, fYB + Radius1 - (wPn1.Y + 2));

end;

Ind := Ind + 2;

until (Ind > 359);

end;

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

// Прорисовать оси

procedure THorizon.ShowAxis();

const GdRd = Pi/180;

var wY, wYm, wYp, Ind : integer;

begin

with fImg.Canvas

do begin

Pen.Width := 1;

Pen.Color := clBlack;

Brush.Style := bsClear;

// Горизонтальная ось

MoveTo(fXB, fYB + Radius1);

LineTo(fXB + 2 \* Radius1, fYB + Radius1);

// Вертикальная ось

MoveTo(fXB + Radius1, fYB);

LineTo(fXB + Radius1, fYB + 2 \* Radius1);

// Прорисовка маркеров тангажа

Ind := 2;

repeat

if Ind >= 50

then begin

Pen.Color := clRed;

Font.Color := clRed;

end;

wY := Round((Radius1 - 4) \* Sin(Ind \* GdRd));

wYm := fYB + Radius1 - wY;

wYp := fYB + Radius1 + wY;

if Ind mod 10 <> 0

then begin

MoveTo(fXB + Radius1 - 2, wYm);

LineTo(fXB + Radius1 + 2, wYm);

MoveTo(fXB + Radius1 - 2, wYp);

LineTo(fXB + Radius1 + 2, wYp);

end

else begin

MoveTo(fXB + Radius1 - 4, wYm);

LineTo(fXB + Radius1 + 4, wYm);

TextOut(fXB + Radius1 + 6,

wYm - TextWidth('0') -1,

IntToStr(-Ind));

MoveTo(fXB + Radius1 - 4, wYp);

LineTo(fXB + Radius1 + 4, wYp);

TextOut(fXB + Radius1 + 6,

wYp - TextWidth('0') -1,

'+' + IntToStr(Ind));

end;

Ind := Ind + 2;

until (Ind > fTrackPitch.Max - 18);

end;

end;

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

// Прорисовка тангажа

procedure THorizon.ShowPitch(RqPitch : extended);

const GdRd = Pi/180;

var wRect : TRect;

wAngle1 : extended;

wAngle2 : extended;

wSin, wCos : extended;

wPn1, wPn2 : Tpoint;

begin

with fImg.Canvas

do begin

Pen.Width := 1;

Pen.Color := clBlack;

Font.Color := clBlack;

end;

if RqPitch >=0

then begin // тангаж (+)

// Угол начала отсечения

wAngle1 := Pi + (RqPitch \* GdRd); // III

// Угол конца отсечения

wAngle2 := - (RqPitch \* GdRd); // IV

end

else begin // тангаж (-)

// Угол начала отсечения

wAngle1 := Pi - (- RqPitch) \* GdRd; // II

// Угол конца отсечения

wAngle2 := (- RqPitch) \* GdRd; // I

end;

// Координаты точек начала отсечения хорды

SinCos(wAngle1, wSin, wCos);

wPn1.X := Round(Radius1 \* wCos);

wPn1.Y := Round(Radius1 \* wSin);

// Координаты точек конца отсечения хорды

SinCos(wAngle2, wSin, wCos);

wPn2.X := Round(Radius1 \* wCos);

wPn2.Y := Round(Radius1 \* wSin);

// Прорисовка тангажа

with fImg.Canvas

do begin

// Очистка индикатора тангажа

wRect := ClearPitch();

// Установить цвет земли

Brush.Color := RGB(200,250,200);

// Прорисовать горизонт и землю

Chord(wRect.Left, wRect.Top, wRect.Right, wRect.Bottom,

fXB + XZero + wPn1.X,

fYB + YZero - wPn1.Y,

fXB + XZero + wPn2.X,

fYB + YZero - wPn2.Y);

end;

end;

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

// Прорисовка контура самолета с учетом крена

procedure THorizon.ShowAirCraft(RqAngle : extended);

const GdRd = Pi/180;

var wSin, wCos : extended;

wAngle : extended;

wPn1 : Tpoint;

Ind : integer;

begin

ShowAxis();

SetLength(fAirCraft, Length(DsAirCraft));

for Ind := Low(fAirCraft) to High(fAirCraft)

do begin

// Угол крена в радианах

wAngle := RqAngle \* GdRd + fPolarArr[Ind].A;

// Переход в Декартовы координаты

SinCos(wAngle, wSin, wCos);

wPn1.X := Round(fPolarArr[Ind].L \* wCos);

wPn1.Y := Round(fPolarArr[Ind].L \* wSin);

// Подготовка массива для Polyline

fAirCraft[Ind].X := fXB + XZero + wPn1.X;

fAirCraft[Ind].Y := fYB + YZero - wPn1.Y;

end;

// Прорисовка контура самолета с учетом крена

with fImg.Canvas

do begin

Pen.Width := 1;

Pen.Color := clBlack;

Font.Color := clBlack;

Polyline(fAirCraft);

end;

SetLength(fAirCraft, 0);

end;

// ================================================================

// ПОДДЕРЖКА PROPERTY

// ================================================================

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

// Установка угла крена

procedure THorizon.SetRoll(RqRoll : integer);

begin

if (RqRoll >= fTrackRoll.Min) and (RqRoll <= fTrackRoll.Max)

then begin

fTrackRoll.Position := RqRoll;

// ShowHorizon();

end;

end;

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

// Чтение угла крена

function THorizon.GetRoll() : integer;

begin

Result := fTrackRoll.Position;

end;

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

// Установка угла тангажа

procedure THorizon.SetPitch(RqPitch : integer);

begin

if (RqPitch >= fTrackPitch.Min) and (RqPitch <= fTrackPitch.Max)

then begin

fTrackPitch.Position := RqPitch;

// ShowHorizon();

end;

end;

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

// Чтение угла тангажа

function THorizon.GetPitch() : integer;

begin

Result := - fTrackPitch.Position;

end;

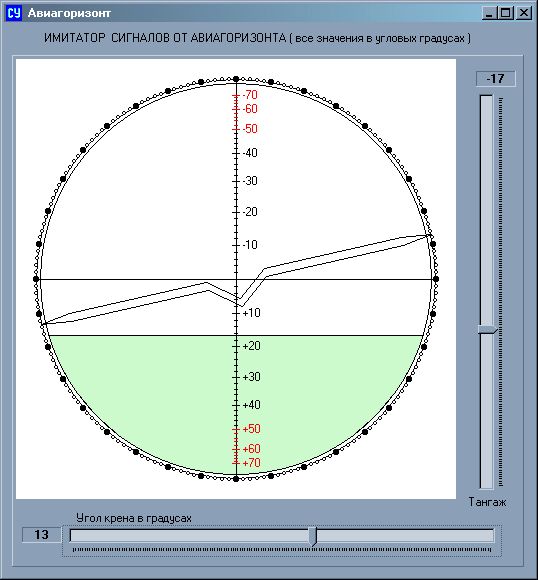
// ================================================================

// КОНЕЦ

// ================================================================

end.

## SensorHorizon01, образ экрана



## unit SensorHorizon01;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, ExtCtrls, StdCtrls, ComCtrls,

//

Horizon01;

type

TSensorHorizonForm = class(TForm)

Label1: TLabel;

procedure FormCreate(Sender: TObject);

procedure FormDestroy(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

SensorHorizonForm : TSensorHorizonForm;

var Horizon : THorizon;

// =========================================================================

// =========================================================================

implementation

{$R \*.dfm}

// =========================================================================

// =========================================================================

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

procedure TSensorHorizonForm.FormCreate(Sender: TObject);

begin

if not Assigned(Horizon)

then begin

Horizon := THorizon.Create(SensorHorizonForm, 8, 32);

Horizon.Roll := 0;

Horizon.Pitch := 0;

end;

end;

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

procedure TSensorHorizonForm.FormDestroy(Sender: TObject);

begin

if Assigned(Horizon)

then begin

Horizon.Free;

Horizon := nil;

end;

end;

// =========================================================================

// КОНЕЦ

// =========================================================================

end.

## unit Service01;

// НАБОР СЕРВИСОВ ДЛЯ ОБСЛУЖИВАНИЯ РАБОТ С ВЕКТОРАМИ

// И РАЗЛИЧНЫХ ОПЕРАЦИЙ НА IMAGE

// РЕДАКЦИЯ ОТ 06.02.2017

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,

Dialogs;

// =========================================================================

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

// Описание двухмерного вектора

type TVector2D = record

X : extended;

Y : extended;

end;

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

// Описание трехмерного вектора

type TVector3D = record

X : extended;

Y : extended;

Z : extended;

end;

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

// Описание двухмерного вектора в декартовых и полярных координатах

type TVector2DP = record

X : extended;

Y : extended;

Len : extended;

Ang : extended;

end;

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

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

// Суммирование двух векторов типа TVector2D

function SumVector2D(const V1, V2 : TVector2D): TVector2D;

// Вычитание (V1 - V2) векторов типа TVector2D

function SubVector2D(const V1, V2 : TVector2D): TVector2D;

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

// Суммирование двух векторов типа TVector3D

function SumVector3D(const V1, V2 : TVector3D): TVector3D;

// Вычитание (V1 - V2) векторов типа TVector3D

function SubVector3D(const V1, V2 : TVector3D): TVector3D;

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

// Переход от декартовых координат к полярным

procedure DecartToPolar (var V : TVector2DP); register;

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

// Переход от полярных координат к декартовым

procedure PolarToDecart(var V : TVector2DP); register;

// =========================================================================

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

// Контроль попадания точки RqX,RqY внутрь RqRect

function IsXYInRect(const RqRect : TRect; RqX, RqY : integer): boolean;

// Контроль попадания точки RqPoint внутрь RqRect

// см. PtInRect

function IsPointInRect(const RqRect : TRect; RqPoint : TPoint): boolean;

// Контроль попадания RqRect внутрь RqMainRect

function IsRectInMainRect(const RqMainRect, RqRect : TRect): boolean;

// Контроль пустого RqRect (один пиксель считается пустым Rect)

// см. IsRectEmpty

function IsEmptyRect(const RqRect : TRect): boolean;

// Контроль пересечения двух RqRect (один пиксель считается не пустым Rect)

// см. IntersectRect

function IsCrossRect(const RqRect1, RqRect2 : TRect): boolean;

// Контроль равенства габаритов TBmp габаритам RqRect

function IsBmpEqualRect(RqBmp : TBitMap; RqRect : TRect) : boolean;

// Переместить RqRect на начальные координаты RqX, RqY

function MoveRectangle (RqRect : TRect; RqX, RqY : integer) : TRect;

// Сохранить в Bitmap область из FromBitMap указанную FomRect

function SaveBitMap(FromBitMap, Bitmap : TBitmap;

FomRect : TRect) : boolean;

// Восстановить из Bitmap область в ToBitMap указанную начальными координатами

function RestoreBitMap(BitMap : TBitmap; // BitMap источник

ToBitMap : TBitmap; // BitMap приемник

Xb, Yb : integer) // Начальные координаты в приемнике

: boolean;

// =========================================================================

// =========================================================================

implementation

// =========================================================================

// =========================================================================

// НАБОР СЕРВИСОВ ДЛЯ ОБСЛУЖИВАНИЯ РАБОТ С ВЕКТОРАМИ

// =========================================================================

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

// Суммирование двух векторов типа TVector2D

// 06.02.2017

function SumVector2D(const V1, V2 : TVector2D): TVector2D;

begin

Result.X := V1.X + V2.X;

Result.Y := V1.Y + V2.Y;

end;

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

// Вычитание (V1 - V2) векторов типа TVector2D

// 06.02.2017

function SubVector2D(const V1, V2 : TVector2D): TVector2D;

begin

Result.X := V1.X - V2.X;

Result.Y := V1.Y - V2.Y;

end;

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

// Суммирование двух векторов типа TVector3D

// 06.02.2017

function SumVector3D(const V1, V2 : TVector3D): TVector3D;

begin

Result.X := V1.X + V2.X;

Result.Y := V1.Y + V2.Y;

Result.Y := V1.Z + V2.Z;

end;

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

// Вычитание (V1 - V2) векторов типа TVector3D

// 06.02.2017

function SubVector3D(const V1, V2 : TVector3D): TVector3D;

begin

Result.X := V1.X - V2.X;

Result.Y := V1.Y - V2.Y;

Result.Y := V1.Z - V2.Z;

end;

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

// 12.02.2017

// Переход от декартовых координат к полярным

procedure DecartToPolar (var V : TVector2DP); register;

asm

FINIT // Сброс регистров и флагов сопроцессора (FPU)

// Стек регистров FPU после выполнения команды

FLD TVector2DP.Y [EAX] // (V.Y);

FLD TVector2DP.X [EAX] // (V.X); (V.Y);

// Вычисление фазы в радианах

FLD ST(1) // (V.Y); (V.X); (V.Y)

FLD ST(1) // (V.X); (V.Y); (V.X); (V.Y);

FPATAN // ArcTan((V.X); (V.Y)); (V.X); (V.Y);

FSTP TVector2DP.Ang [EAX] // (V.X); (V.Y)

// Вычисление модуля

FLD ST(0) // (V.X); (V.X); (V.Y);

FMUL // (V.X \* V.X); (V.Y);

FXCH ST(1) // (V.Y); (V.X \* V.X);

FLD ST(0) // (V.Y); (V.Y); (V.X \* V.X)

FMUL // (V.Y \* V.Y); (V.X \* V.X);

FADD // ((V.Y \* V.Y) + (V.X \* V.X));

FSQRT // sqrt((V.Y \* V.Y) + (V.X \* V.X));

FSTP TVector2DP.Len [EAX] // [empty]

FWAIT

end;

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

// 12.02.2017

// Переход от полярных координат к декартовым

procedure PolarToDecart(var V : TVector2DP); register;

asm

FINIT // Сброс регистров и флагов сопроцессора (FPU)

// Стек регистров FPU после выполнения команды

FLD TVector2DP.Len [EAX] // (V.Len)

FLD TVector2DP.Ang [EAX] // (V.Len)

FSINCOS // (cos(V.Ang)); (sin(V.Ang)); (V.Len)

FMUL ST, ST(2) // (V.Len \* cos(V.Ang)); (sin(V.Ang)); (V.Len)

FSTP TVector2DP.X [EAX] // (sin(V.Ang)); (V.Len)

FMUL // (V2P.Len \* sin(V2P.Ang))

FSTP TVector2DP.Y [EAX] // [empty]

FWAIT

end;

// =========================================================================

// НАБОР СЕРВИСОВ ДЛЯ ОБСЛУЖИВАНИЯ РАЗЛИЧНЫХ ОПЕРАЦИЙ НА IMAGE

// =========================================================================

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

// 29.01.2017

// Контроль попадания точки RqX,RqY внутрь RqRect

function IsXYInRect(const RqRect : TRect; RqX, RqY : integer): boolean;

begin

Result := (RqX >= RqRect.Left) and (RqX <= RqRect.Right) and

(RqY >= RqRect.Top) and (RqY <= RqRect.Bottom);

end;

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

// Контроль попадания точки RqPoint внутрь RqRect

// см. PtInRect

// 29.01.2017

function IsPointInRect(const RqRect : TRect; RqPoint : TPoint): boolean;

begin

Result := (RqPoint.X >= RqRect.Left) and

(RqPoint.X <= RqRect.Right) and

(RqPoint.Y >= RqRect.Top) and

(RqPoint.Y <= RqRect.Bottom);

end;

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

// Контроль попадания RqRect внутрь RqMainRect

// 29.01.2017

function IsRectInMainRect(const RqMainRect, RqRect : TRect): boolean;

begin

Result := IsXYInRect(RqMainRect, RqRect.Left, RqRect.Top) and

IsXYInRect(RqMainRect, RqRect.Right, RqRect.Bottom);

end;

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

// Контроль пустого RqRect (один пиксель считается пустым Rect)

// см. IsRectEmpty

// 29.01.2017

function IsEmptyRect(const RqRect : TRect): boolean;

begin

Result := IsRectEmpty(RqRect);

end;

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

// Контроль пересечения двух RqRect (один пиксель считается не пустым Rect)

// см. IntersectRect

// 29.01.2017

function IsCrossRect(const RqRect1, RqRect2 : TRect): boolean;

begin

Result := IsXYInRect(RqRect1, RqRect2.Left, RqRect2.Top) or

IsXYInRect(RqRect1, RqRect2.Right, RqRect2.Bottom) or

IsXYInRect(RqRect1, RqRect2.Right, RqRect2.Top) or

IsXYInRect(RqRect1, RqRect2.Left, RqRect2.Bottom) or

IsXYInRect(RqRect2, RqRect1.Left, RqRect1.Top) or

IsXYInRect(RqRect2, RqRect1.Right, RqRect1.Bottom) or

IsXYInRect(RqRect2, RqRect1.Right, RqRect1.Top) or

IsXYInRect(RqRect2, RqRect1.Left, RqRect1.Bottom);

end;

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

// Контроль равенства габаритов TBmp габаритам RqRect

// 29.01.2017

function IsBmpEqualRect(RqBmp : TBitMap; RqRect : TRect) : boolean;

begin

Result := False;

if Assigned(RqBmp)

then begin

if RqBmp.Height <> (RqRect.Bottom - RqRect.Top) + 1 then Exit;

if RqBmp.Width <> (RqRect.Right - RqRect.Left) + 1 then Exit;

Result := True;

end;

end;

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

// Переместить RqRect на начальные координаты RqX, RqY

// 29.01.2017

function MoveRectangle (RqRect : TRect; RqX, RqY : integer) : TRect;

begin

Result := Rect(RqX, RqY,

RqX + (RqRect.Right - RqRect.Left),

RqY + (RqRect.Bottom - RqRect.Top ));

end;

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

// Сохранить в Bitmap область из FromBitMap указанную FomRect

// 29.01.2017

function SaveBitMap(FromBitMap, Bitmap : TBitmap;

FomRect : TRect) : boolean;

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

// SysUtils:

// PByteArray = ^TByteArray;

// TByteArray = array[0..32767] of Byte;

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

var

RowBeg, ColBeg : Integer;

RowEnd, ColEnd : Integer;

PFrom, PTo : PByteArray;

RowFrom, RowTo : Integer;

ColFrom, ColTo : Integer;

begin

Result := False;

if not (FromBitMap.PixelFormat = pf24bit) then Exit;

// Контроль геометрии наложения FomRec

// 1) FomRect за пределами FromBitMap

if FomRect.Top >= FromBitMap.Height then Exit;

if FomRect.Left >= FromBitMap.Width then Exit;

// 2) Нормировка диапазона строк

RowBeg := FomRect.Top;

if RowBeg < 0 then RowBeg := 0;

RowEnd := FomRect.Bottom;

if RowEnd > FromBitMap.Height - 1 then RowEnd := FromBitMap.Height - 1;

// 3) Нормировка диапазона столбцов

ColBeg := FomRect.Left;

if ColBeg < 0 then ColBeg := 0;

ColEnd := FomRect.Right;

if ColEnd > FromBitMap.Width - 1 then ColEnd := FromBitMap.Width;

// 4) Проверка на пустой FomRec

if RowEnd < RowBeg then Exit;

if ColEnd < ColBeg then Exit;

// Начинаем копирование

try

Bitmap.PixelFormat := pf24bit;

Bitmap.Height := RowEnd - RowBeg + 1;

Bitmap.Width := ColEnd - ColBeg + 1;

RowTo := 0;

for RowFrom := RowBeg to RowEnd do

begin

// Представим строку BitMap через указатель на массив байтов

PFrom := FromBitMap.ScanLine[RowFrom];

PTo := Bitmap.ScanLine[RowTo];

// Побайтовое копирование. Порядок байтов в пикселе pf24bit : B,G,R

ColTo := 0;

for ColFrom := ColBeg \* 3 to ColEnd \* 3

do begin

PTo^[ColTo] := PFrom^[ColFrom];

Inc(ColTo);

end;

Inc(RowTo);

end;

Result := True;

except

ShowMessage('Не могу сохранить BitMap');

end;

end; // of function

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

// Восстановить из Bitmap область в ToBitMap указанную начальными координатами

// 29.01.2017

function RestoreBitMap(BitMap : TBitmap; // BitMap источник

ToBitMap : TBitmap; // BitMap приемник

Xb, Yb : integer) // Начальные координаты в приемнике

: boolean;

var

RowBeg, ColBeg : Integer;

RowEnd, ColEnd : Integer;

PFrom, PTo : PByteArray;

RowFrom, RowTo : Integer;

ColFrom, ColTo : Integer;

begin

Result := False;

if not (ToBitmap.PixelFormat = pf24bit) then Exit;

// Контроль геометрии наложения FomRec

// 1) Xb, Yb за пределами ToBitMap

if Yb >= ToBitmap.Height then Exit;

if Xb >= ToBitmap.Width then Exit;

// 2) Нормировка диапазона строк

RowBeg := Yb;

if Yb < 0

then begin

RowBeg := 0;

RowEnd := BitMap.Height - 1;

end

else RowEnd := Yb + BitMap.Height - 1;

if RowEnd > ToBitMap.Height - 1 then RowEnd := ToBitMap.Height - 1;

// 3) Нормировка диапазона столбцов

ColBeg := Xb;

if Xb < 0

then begin

ColBeg := 0;

ColEnd := BitMap.Width - 1;

end

else ColEnd := Xb + BitMap.Width - 1;

if ColEnd > ToBitMap.Width - 1 then ColEnd := ToBitMap.Width;

// 4) Проверка на пустой ToRec

if RowEnd < RowBeg then Exit;

if ColEnd < ColBeg then Exit;

// Начинаем копирование

try

RowFrom := 0;

for RowTo := RowBeg to RowEnd do

begin

// Представим каждую строку BitMap как

// указатель на массив байтов

PFrom := BitMap.ScanLine[RowFrom];

PTo := ToBitmap.ScanLine[RowTo];

// побайтовое копирование изображения

ColFrom := 0;

for ColTo := ColBeg \* 3 to ColEnd \* 3

do begin

PTo^[ColTo] := PFrom^[ColFrom];

Inc(ColFrom);

end;

Inc(RowFrom);

end;

Result := True;

except

ShowMessage('Не могу восстановить BitMap');

end;

end; // of function

// =========================================================================

// КОНЕЦ

// =========================================================================

end.