СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc137400236)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc137400237)

[Анализ существующих решений 5](#_Toc137400238)

[Inno Setup 5](#_Toc137400239)

[Install Aware 6](#_Toc137400240)

[NSIS 6](#_Toc137400241)

[Заключение 7](#_Toc137400242)

[2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 8](#_Toc137400243)

[Изучение возможностей языка Delphi 8](#_Toc137400244)

[Тестирование 11](#_Toc137400245)

[Отладка 11](#_Toc137400246)

[ЗАКЛЮЧЕНИЕ 12](#_Toc137400247)

[Список использованных источников 13](#_Toc137400248)

[Приложение А 14](#_Toc137400249)

# ВВЕДЕНИЕ

Цель работы: разработать программу обработки типизированных файлов с кодом на языке Object Pascal, для ввода и вывода данных использовать компоненты Delphi.

В первом разделе курсовой работы выполнен анализ технического задания, выделены функции, которыми должно обладать разрабатываемое приложение, а также сформулированы требования к нему, произведена постановка задачи на проектирование. Во втором разделе выполняется анализ задачи проектирования, анализируется вид исходных данных, в зависимости от чего проектируется визуальный интерфейс и приводится словесный алгоритм работы программы. В третьем разделе рассматриваются алгоритмы разработанных процедур и функций, а также приводятся их описания. К реализованной программе разработана инструкция пользователя, которая приведена в четвертом разделе. Алгоритмы процедур приведены в приложении А. Листинг программы содержится в приложении Б.

Программа разработана с использованием среды визуального программирования Delphi 11 и языка Object Pascal.

Данный курсовой проект был разработан в среде Delphi11 с базовым языком программирования Object Pascal. Среди множества языков Pascal является наиболее универсальным и легко изучаемым языком. При этом его удобно использовать для создания сложных приложений, базами данных, приложения масштаба предприятия. Delphi так же прост в обучении, как и Visual Basic, но при этом средства языка программирования Pascal позволяет использовать все современные возможности объектно-ориентированного программирования.

Delphi — это среда быстрой разработки, в которой в качестве языка программирования используется язык Delphi. Язык Delphi — строго типизированный объектно-ориентированный язык, в основе которого лежит хорошо знакомый программистам Object Pascal.

Delphi прост и логичен. Основные конструкции языка четко выделяются в программе, что способствует хорошему восприятию написанного кода. Графический интерфейс Delphi отлично продуман, среда разработки предоставляет доступ только к тем участкам кода, с которыми необходимо работать, скрывая основную часть программы, которая создается автоматически самой разработкой.

Для выполнения данной курсовой работы необходимо разработать алгоритм решения поставленного задания, правильно указав последовательное выполнение действий.

# 1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

В современном мире развитие информационных технологий является неотъемлемой частью жизни людей. В связи с этим возрастает спрос на удобные и функциональные программные продукты. Одним из инстру­ментов, упрощающих жизнь разработчиков и пользователей, являются инсталляторы. Инсталлятор — это программа, предназначенная для установки другой программы или сервиса на компьютер пользователя.

В данной работе рассмотрен процесс создания инсталлятора на языке программирования Delphi. Delphi — это язык программирования, разрабатываемый компанией Embarcadero Technologies. Он относится к семей­ству языков Pascal и позволяет создавать быстрые и эффективные приложения.

Цель данной работы — ознакомление с процессом создания инсталлятора на языке Delphi, а также реализация практической части работы в виде создания инсталлятора для простого приложения. В процессе работы будут рассмотрены основные принципы работы с инсталляторами, используемые инструменты и библиотеки, необходимые для создания инсталлятора на языке Delphi.

В рамках работы автором была разработана программа установки простого приложения, которая может быть использована при создании инсталляторов для других программ.

Анализ существующих решений

Существует множество инструментов для создания инсталляторов на языке Delphi. Рассмотрим некоторые из них.

Inno Setup

Inno Setup - это бесплатный инструмент для создания установщиков на языке Delphi. Он имеет графический интерфейс и поддерживает множество функций, таких как установка файлов, создание ярлыков, запуск команд и многое другое. Inno Setup также имеет мощную систему сценариев, позволяющую программистам создавать собственные установщики с использованием Delphi-скриптов. Внешний вид инсталлятора представлен на рисунке 1.

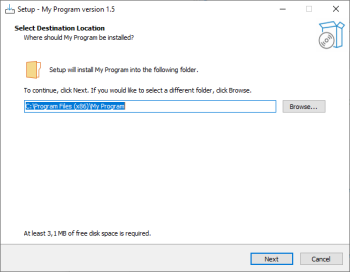


Рисунок 1. Внешний вид инсталлятора.

Install Aware

InstallAware - это полнофункциональный инструмент для создания инсталляторов на языке Delphi. Он предлагает множество функций, таких как создание пользовательских сообщений, установка файлов, создание ярлыков, создание реестровых ключей и многое другое. InstallAware также поддерживает мультиязычность и может создавать установщики для нескольких операционных систем. Внешний вид инсталлятора представлен на рисунке 2.

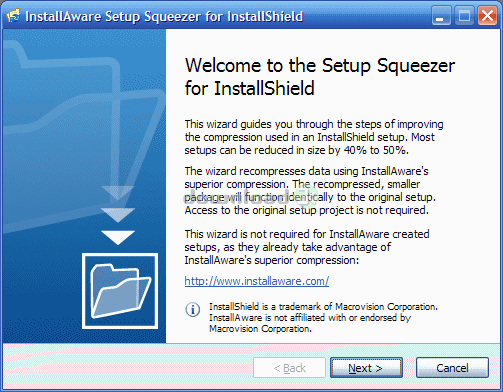


Рисунок 2. Внешний вид инсталлятора.

NSIS

NSIS - это другой бесплатный инструмент для создания инсталляторов на языке Delphi. Он имеет мощную систему сценариев на языке скриптов, которая позволяет создавать установщики с различными функциями, такими как установка файлов, создание ярлыков и т.д. NSIS также может создавать компактные установщики, что позволяет быстро загрузить установщик с веб-сайта. Внешний вид инсталлятора представлен на рисунке 3.

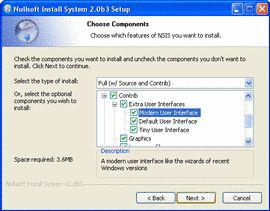


Рисунок 3. Внешний вид инсталлятора.

## Заключение

В данном разделе мы рассмотрели некоторые из лучших инструментов для создания установщиков на языке Delphi. Они все поддерживают множество функций и имеют свои преимущества и недостатки. Несмотря на это, Inno Setup, InstallAware и NSIS являются абсолютно бесплатными и отличными выборами для создания инсталляторов на языке Delphi.

# 2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

**Задание:** Разработка программного средства для инсталляции и деинсталляции программного обеспечения.

Изучение возможностей языка Delphi

Для создания инсталлятора был выбран язык программирования Delphi, так как данный язык имеет множе­ство инструментов для работы с пользовательским интерфейсом и файловой системой.

Одной из главных особенностей языка Delphi является его объектно-ориентированная природа. Это поз­воляет создавать гибкий и удобный код с возможностью повторного использования.

Для работы с пользовательским интерфейсом в Delphi используется компонентный подход, который позво­ляет быстро и просто создавать различные элементы интерфейса. Кроме того, в языке Delphi есть множество готовых компонентов, которые можно использовать для создания интерфейса без написания дополнительного кода.

Для работы с файловой системой в Delphi используются библиотеки, которые содержат множество классов и методов для работы с файлами и папками. Это позволяет создать удобный и функциональный инсталлятор для любого приложения.

Одной из особенностей языка Delphi является его высокая производительность. Это достигается за счет оптимизации компилятора и использования многопоточности.

В заключение можно отметить, что язык Delphi имеет множество возможностей для создания функционального и удобного инсталлятора. Он позволяет быстро создавать пользовательский интерфейс, работать с файловой системой и обеспечивает высокую производительность.

Начнем разработку с создания интерфейса программы. Разместим на форме необходимые нам в работе компоненты, такие как TButton, TGroupBox, TListBox, TLabel, TEdit. Подпишем необходимые поля ввода данных, добавим компоненты TFileOpenDialog, TFileSaveDialog. Они понадобятся нам для вывода диалогов выбора файлов для открытия и сохранения. Внешний вид инсталлятора представлен на рисунке 4.

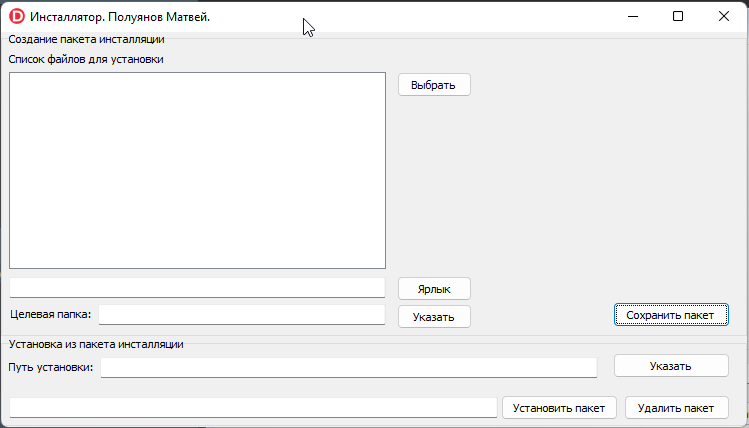


Рисунок 4. Внешний вид инсталлятора после запуска приложения.

Добавим обработчик для кнопки. Текст обработчика:

procedure TForm1.Button4Click(Sender: TObject);

begin

if OpenDialog1.Execute then

begin

ListBox1.Items.Clear;

ListBox1.Items:=OpenDialog1.Files;

end;

end;

Этот обработчик загрузит в компонент ListBox имена файлов которые необходимо проинсталлировать в систему. Следующее поле ввода с кнопкой предлагает выбрать имя файла для которого необходимо создать ярлык на рабочем столе по окончании процесса инсталляции.

Текст процедуры которая создает ярлык:

procedure MakeLNK(AppName:string);

var

SL: IShellLink;

PF: IPersistFile;

LnkName: WideString;

begin

OleCheck(CoCreateInstance(CLSID\_ShellLink, nil, CLSCTX\_INPROC\_SERVER,

IShellLink, SL));

{ IShellLink implementers are required to implement IPersistFile }

PF := SL as IPersistFile;

OleCheck(SL.SetPath(PChar(AppName))); // set link path to proper file

{ create a path location and filename for link file }

LnkName := GetFolderLocation('Desktop') + '\' +

ChangeFileExt(ExtractFileName(AppName), '.lnk');

PF.Save(PWideChar(LnkName), True); // save link file

end;

На следующем шаге необходимо указать папку в которую необходимо установить приложение.

Процедура выбора папки отсутствует в Delphi в стандартном наборе компонент, её текст:

Function OpenFolder(form1: tform): string;

var

TitleName : string;

lpItemID : PItemIDList;

BrowseInfo : TBrowseInfo;

DisplayName: array [0 .. MAX\_PATH] of char;

TempPath : array [0 .. MAX\_PATH] of char;

begin

FillChar(BrowseInfo, sizeof(TBrowseInfo), #0);

BrowseInfo.hwndOwner := form1.handle;

BrowseInfo.pszDisplayName := @DisplayName;

TitleName := 'Выберите папку ';

BrowseInfo.lpszTitle := PChar(TitleName);

BrowseInfo.ulFlags := BIF\_RETURNONLYFSDIRS;

lpItemID := SHBrowseForFolder(BrowseInfo);

if lpItemID <> nil then

begin

SHGetPathFromIDList(lpItemID, TempPath);

// ShowMessage(TempPath);

GlobalFreePtr(lpItemID);

end;

OpenFolder := TempPath;

end;

Внешний вид приложения готового к созданию инсталлятора представлен на рисунке 5.

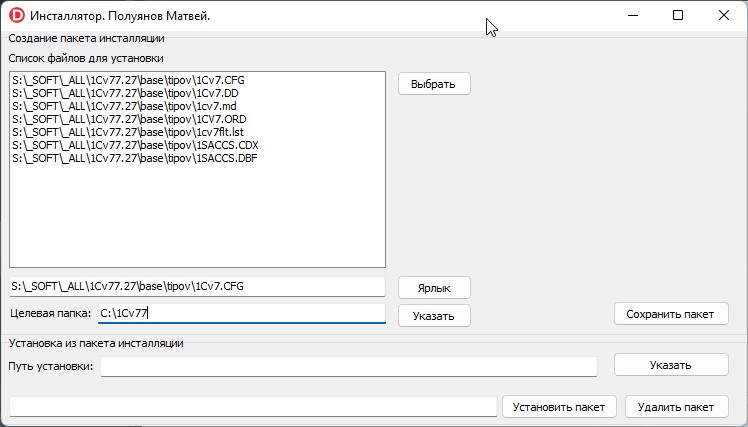


Рисунок 5. Внешний вид приложения готового к созданию инсталлятора.

После нажатия кнопки Сохранить пакет программа сформирует список файлов для инсталляции, заархивирует их, создаст файл необходимый для их распаковки и создания ярлыка. По окончании работы инсталлятора необходимые файлы будут помещены в указанную папку.

Для установки программного обеспечения необходимо указать путь для установки и указать пакет инсталляции с инструкциями для их установки. Внешний вид инсталлятора с выбранным для инсталляции пакетом и путем установки представлен на рисунке 6.

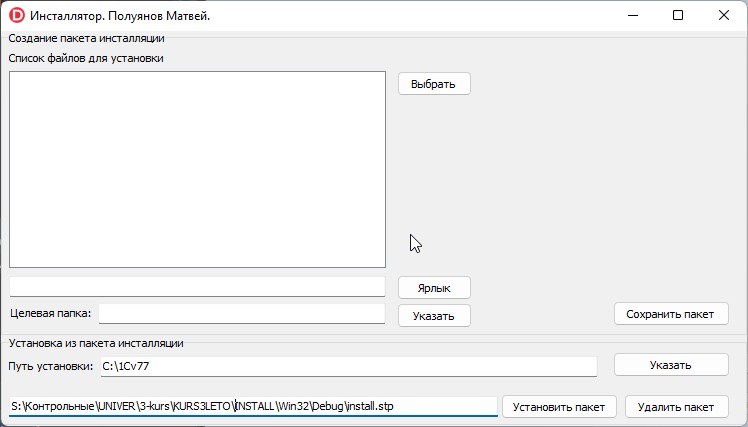


Рисунок 6. Внешний вид инсталлятора с выбранным для инсталляции пакетом и путем установки.

При нажатии на кнопку установить пакет, будет создана папка пути установки, в неё будут распакованы необходимые файлы и создан ярлык на рабочем столе для выбранного на этапе создания пакета файла.

Для удаления пакета необходимо указать путь к папке, в которую была установлена программа и нажать кнопку Удалить пакет.

В результате работы был создан установочный пакет на языке Delphi, который позволяет автоматически устанавливать приложение и настраивать его работу с помощью необходимых ключей реестра.

Инсталлятор был протестирован на нескольких компьютерах с различными конфигурациями и операционными системами и успешно выполнил все заданные функции.

После создания инсталлятора необходимо провести тестирование и отладку, чтобы гарантировать корректную его работу на всех системах. Тестирование проводится на различных конфигурациях, операционных системах и приложениях.

Тестирование

Первый этап тестирования — это функциональное тестирование, в котором проверяется соответствие поведения инсталлятора требованиям. Необходимо проверить, что инсталлятор:

* корректно отображает информацию в процессе установки;
* правильно выбирает целевую папку установки;
* корректно копирует файлы;
* правильно создает ярлыки на рабочем столе и в меню Пуск;
* правильно регистрирует приложение в системе;
* удаляет все созданные файлы и ярлыки при удалении приложения.

Во время функционального тестирования также проводятся тесты на безопасность и стабильность работы инсталлятора.

После функционального тестирования проводится совместимостное тестирование. В рамках этого тестиро­вания проверяется работоспособность инсталлятора на различных операционных системах и аппаратных конфигурациях.

Отладка

При тестировании инсталлятора могут проявляться ошибки и неполадки. Для их устранения проводится отладка.

Первым шагом отладки является выявление, в каких условиях возникает ошибка. Это поможет сузить круг поиска проблемы и сосредоточить усилия на ее решении. Для этого необходимо записывать все ошибки, ко­торые появляются в процессе тестирования инсталлятора.

Далее, необходимо провести дополнительное тестирование для воспроизведения ошибки. Это поможет точнее определить причину ошибки и, возможно, обнаружить другие связанные неполадки.

После выявления причины ошибки можно приступать к ее устранению. Необходимо исправить код и провести повторное тестирование, чтобы убедиться, что ошибка была устранена.

# ЗАКЛЮЧЕНИЕ

В заключении данной курсовой работы был рассмотрен процесс создания инсталлятора на языке Delphi.

В результате выполнения работы был разработан установщик для простого приложения. В процессе созда­ния были рассмотрены основные этапы разработки, а также пройдены все необходимые шаги по созданию полноценного инсталлятора.

Инсталлятор может быть использован для установки приложения на любом компьютере, что дает возмож­ность ускорения процесса установки приложений и облегчения работы с пользовательской программной про­дукцией.

Проведенное исследование и разработка с использованием языка Delphi дает возможность разработчикам создавать более эффективные и удобные инсталляторы для своих приложений.

Таким образом, данная работа стала хорошим примером использования Delphi для разработки программных установщиков, а также дает возможность использования полученных знаний и опыта при разработке подобных проектов.

# Список использованных источников

1. Кузнецов А. А. Создание установщика приложений на языке Delphi. [Электронный ресурс] // Хабр. Режим доступа: <https://habr.com/ru/post/84600/>
2. Smith J. Delphi XE2 Foundations Part 1 (Installing and Configuring Delphi XE2). [Электронный ресурс] // Embarcadero Technologies. Режим доступа: [https://www.embarcadero.com/ru/white-papers/delphi-xe2- foundations-part-1](https://www.embarcadero.com/ru/white-papers/delphi-xe2-foundations-part-1)
3. Иванов В.И. Проектирование и разработка приложений в среде Delphi. М.: Издательский центр "Акаде­мия", 2012. 368 с.
4. Jones Т. Delphi in a Nutshell. Sebastopol, СА: O’Reilly Media, 2000. 400 с.
5. Петров В.Д. Разработка многопользовательских приложений с помощью языка Delphi. СПб.: БХВ- Петербург, 2014. 448 с.
6. Валентин Озеров «Советы по Delphi», 1999
7. Зуев Е.А. Программирование на языке Turbo Pascal 6.0,7.0. – М.: Радио и связь, Веста, 1993.
8. Фаронов В.В. Turbo Pascal 7.0. Начальный курс. - М.: Нолидж, 2000.
9. Фаронов В.В. «DELPHI. Программирование на языке высокого уровня». – Питер, 2005.
10. Дятченко Д.А. Методические указания к выполнению курсовой работы по дисциплине «Программирование и алгоритмические языки»– Северодонецк, 2005.

# Приложение А

**Текст программы.**

unit UMain;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, ShellAPI;

type

TForm1 = class(TForm)

GroupBox1: TGroupBox;

Button1: TButton;

GroupBox2: TGroupBox;

Button2: TButton;

Edit1: TEdit;

Label1: TLabel;

ListBox1: TListBox;

Edit3: TEdit;

Button3: TButton;

Button4: TButton;

OpenDialog1: TOpenDialog;

SaveDialog1: TSaveDialog;

Label2: TLabel;

Edit4: TEdit;

Button5: TButton;

Edit2: TEdit;

Label3: TLabel;

Button6: TButton;

Button7: TButton;

procedure Button4Click(Sender: TObject);

procedure Button3Click(Sender: TObject);

procedure Button1Click(Sender: TObject);

procedure Button5Click(Sender: TObject);

procedure Button2Click(Sender: TObject);

procedure Button6Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

implementation

{$R \*.dfm}

uses ComObj, ActiveX, ShlObj, Registry;

const

{ Registry key where Folder information is kept }

SFolderKey = '\Software\Microsoft\Windows\CurrentVersion\' +

'Explorer\Shell Folders';

function GetFolderLocation(const FolderType: string): string;

{ Retrieves from registry path to folder indicated in FolderType }

begin

with TRegistry.Create do

try

RootKey := HKEY\_CURRENT\_USER;

if not OpenKey(SFolderKey, False) then

{ open key where shell folder information is kept. }

raise ERegistryException.CreateFmt('Folder key "%s" not found',

[SFolderKey]);

{ Get path for specified folder }

Result := ReadString(FolderType);

if Result = '' then

raise ERegistryException.CreateFmt('"%s" item not found in registry',

[FolderType]);

CloseKey;

finally

Free;

end;

end;

procedure MakeLNK(AppName:string);

var

SL: IShellLink;

PF: IPersistFile;

LnkName: WideString;

begin

OleCheck(CoCreateInstance(CLSID\_ShellLink, nil, CLSCTX\_INPROC\_SERVER,

IShellLink, SL));

{ IShellLink implementers are required to implement IPersistFile }

PF := SL as IPersistFile;

OleCheck(SL.SetPath(PChar(AppName))); // set link path to proper file

{ create a path location and filename for link file }

LnkName := GetFolderLocation('Desktop') + '\' +

ChangeFileExt(ExtractFileName(AppName), '.lnk');

PF.Save(PWideChar(LnkName), True); // save link file

end;

function CreateDirEx(Dir: string): boolean;

var

i, L : integer;

CurDir: string;

begin

if ExcludeTrailingBackslash(Dir) = '' then

exit;

Dir := IncludeTrailingBackslash(Dir);

L := length(Dir);

for i := 1 to L do

begin

CurDir := CurDir + Dir[i];

if Dir[i] = '\' then

begin

if not DirectoryExists(CurDir) then

if not CreateDir(CurDir) then

exit;

end;

end;

result := true;

end;

function CopyFile(FromPath, ToPath: string): integer;

var

F1 : file;

F2 : file;

NumRead : integer;

NumWritten: integer;

Buf : pointer;

BufSize: longint;

Totalbytes: longint;

TotalRead : longint;

begin

result := 0;

Assignfile(F1, FromPath);

Assignfile(F2, ToPath);

Reset(F1, 1);

Totalbytes := Filesize(F1);

Rewrite(F2, 1);

// BufSize := 16384;

GetMem(Buf, Bufsize);

TotalRead := 0;

repeat

BlockRead(F1, Buf^, Bufsize, NumRead);

inc(TotalRead, NumRead);

BlockWrite(F2, Buf^, NumRead, NumWritten);

Application.ProcessMessages;

until (NumRead = 0) or (NumWritten <> NumRead);

if (NumWritten <> NumRead) then

begin

// ошибка

result := -1;

end;

Closefile(F1);

Closefile(F2);

end;

Procedure ListFileDir(Path: string; FileList: TStrings; mask: string);

var

SR: TSearchRec;

begin

if FindFirst(Path + mask, faAnyFile, SR) = 0 then

begin

repeat

if (SR.Attr <> faDirectory) then

begin

FileList.Add(Path + SR.Name);

end;

until FindNext(SR) <> 0;

FindClose(SR);

end;

end;

Function OpenFolder(form1: tform): string;

var

TitleName : string;

lpItemID : PItemIDList;

BrowseInfo : TBrowseInfo;

DisplayName: array [0 .. MAX\_PATH] of char;

TempPath : array [0 .. MAX\_PATH] of char;

begin

FillChar(BrowseInfo, sizeof(TBrowseInfo), #0);

BrowseInfo.hwndOwner := form1.handle;

BrowseInfo.pszDisplayName := @DisplayName;

TitleName := 'Выберите папку ';

BrowseInfo.lpszTitle := PChar(TitleName);

BrowseInfo.ulFlags := BIF\_RETURNONLYFSDIRS;

lpItemID := SHBrowseForFolder(BrowseInfo);

if lpItemID <> nil then

begin

SHGetPathFromIDList(lpItemID, TempPath);

// ShowMessage(TempPath);

GlobalFreePtr(lpItemID);

end;

OpenFolder := TempPath;

end;

function ExecAndWait(aCmd: string; WaitTimeOut: cardinal = INFINITE): cardinal;

var

si : STARTUPINFO;

pi : PROCESS\_INFORMATION;

res: BOOL;

r : cardinal;

begin

with si do

begin

cb := sizeof(si);

lpReserved := nil;

lpDesktop := nil;

lpTitle := PChar('External program "' + aCmd + '"');

dwFlags := 0;

cbReserved2 := 0;

lpReserved2 := nil;

end;

res := CreateProcess(nil, PChar(aCmd), nil, nil, false, 0, nil, nil, si, pi);

if res then

WaitForSingleObject(pi.hProcess, WaitTimeOut);

GetExitCodeProcess(pi.hProcess, r);

result := r;

end;

procedure TForm1.Button1Click(Sender: TObject);

var

s,p,pk:string;

i:integer;

f,fi:textfile;

begin

p:=extractfilepath(paramstr(0))+'\'+'TMP';

CreateDirEx(p);

Savedialog1.InitialDir:=p+'\';

// тут мы собственно выполняем все действия по формированию пакета

assignfile(f,extractfilepath(paramstr(0))+'\install.stp');

rewrite(f);

assignfile(fi,extractfilepath(paramstr(0))+'\install.lst');

rewrite(fi);

Writeln(f,extractfilename(edit3.text));

//Writeln(f,edit4.text);

for I := 0 to ListBox1.Items.Count-1 do

begin

Copyfile(ListBox1.Items[i],p+'\'+extractfilename(ListBox1.Items[i]));

Writeln(f,extractfilename(ListBox1.Items[i]));

Writeln(fi,extractfilepath(paramstr(0))+'TMP\'+extractfilename(ListBox1.Items[i]));

end;

// запаковать и сложить в папку

//ExecAndWait(extractfilepath(paramstr(0)+'\make.cmd'));

ShellExecute(Form1.Handle, 'open', PChar(extractfilepath(paramstr(0))+'\make.cmd'), PChar(''), nil, SW\_HIDE);

closefile(f);

closefile(fi);

//end;

// помещаем штсталлятор вцелевую папку

CreateDirEx(edit4.text);

//Copyfile(extractfilepath(paramstr(0))+'\install.exe',edit4.text+'\install.exe');

Copyfile(extractfilepath(paramstr(0))+'\install.rar',edit4.text+'\install.rar');

Copyfile(extractfilepath(paramstr(0))+'\install.stp',edit4.text+'\install.stp');

Copyfile(extractfilepath(paramstr(0))+'\rar.exe',edit4.text+'\rar.exe');

Copyfile(extractfilepath(paramstr(0))+'\restore.cmd',edit4.text+'\restore.cmd');

ShowMessage('Пакет создан!');

end;

procedure TForm1.Button2Click(Sender: TObject);

var

p, pd,s,ls:string;

f, fi:textfile;

begin

if OpenDialog1.Execute then

begin

EDit1.TExt:=OpenDialog1.FileName;

p:=extractfilepath(Edit2.Text);

ShellExecute(Form1.Handle, 'open', PChar(extractfilepath(paramstr(0))+'\restore.cmd'), PChar(''), nil, SW\_HIDE);

CreateDirEx(Edit1.Text);

assignfile(fi,Edit1.Text);

reset(fi);

ReadLn(fi,ls);

while not eof(fi) do

begin

ReadLn(fi,s);

Copyfile(extractfilepath(paramstr(0))+'\'+s,edit2.text+'\'+s);

end;

Closefile(fi);

MakeLNK(EDit2.TExt+'\'+ls);

ShowMessage('Пакет установлен!');

end

else Edit1.TExt:='';

end;

procedure TForm1.Button3Click(Sender: TObject);

begin

if OpenDialog1.Execute then

begin

EDit3.TExt:=OpenDialog1.FileName;

end

else Edit3.TExt:='';

end;

procedure TForm1.Button4Click(Sender: TObject);

begin

if OpenDialog1.Execute then

begin

ListBox1.Items.Clear;

ListBox1.Items:=OpenDialog1.Files;

end;

end;

procedure TForm1.Button5Click(Sender: TObject);

begin

Edit4.TExt:= OpenFolder(form1);

end;

procedure TForm1.Button6Click(Sender: TObject);

begin

Edit2.TExt:= OpenFolder(form1);

end;

end.