Введение.

Кому предназначена статья?

«Будешь читать литературу для «чайников» - станешь «чайником» - программистская мудрость. Моя статья написана для начинающих программистов (НЕ ДЛЯ «ЧАЙНИКОВ»!!!), ее цель – избавить их от долгих бесполезных кликаний мышью по кнопкам, пытаясь «вслепую» создать более-менее удобный инсталлятор для своей программы. Сразу предупреждаю: в статье не будет сложных примеров скриптов, заумных фраз и обилия терминов. Здесь собран только самый необходимый минимум, который сделает процесс написания инсталлятора простым и понятным.

Зачем нужен Inno Setup?

Итак, после долгих дней, месяцев или даже лет кропотливой работы, вы все-таки написали рабочую программу. Да, возможно, она работает не совсем так, как вам хочется, но главное – ОНА РАБОТАЕТ! Теперь дело за малым: создать самораспаковывающийся архив и опубликовать ваше творение во всемирной паутине, чтобы человечество могло по достоинству оценить ваши труды. Но тут возникает вопрос: а что, если ваша программа имеет несколько вариантов установки (например, стандартная, минимальная и выборочная), а еще хочется после установки программы поместить ее иконку на рабочий стол пользователя, добавить ее в автозапуск… А может быть после инсталляции пользователю следует перезагрузить компьютер… А может ему вообще не следует устанавливать вашу программу на свой компьютер, если он не наделен правами администратора? В принципе, многое можно реализовать при помощи тех же самораспаковывающихся архивов, но как долго вам придется с этим разбираться и не уйдет ли на создание инсталлятора времени больше, чем на создание самой программы? И не увеличится ли после этого в разы ее размер?

Вот тут на помощь программисту и приходят готовые программы-инсталляторы, одним из которых является Inno Setup – с его помощью можно быстро создать удобный и простой инсталлятор.

Совет.

Для того, чтобы найти в статье ответ на интересующий Вас вопрос, автор советует прочесть ее всю. Это не займет много времени, зато Вы наверняка найдете то, что искали, совершенно в неожиданном месте – не потому, что статья не последовательна, а потому, что все секции Inno Setup связаны между собой настолько тесно, что очень трудно бывает отнести пример выполнения какой-либо задачи к описанию конкретной секции.

Принятые обозначения

Если имеется в виду Inno Setup, в тексте он будет так и обозначен **– Inno Setup**, либо **инсталлятор Inno Setup**.

Если же имеется в виду скрипт (код), написанный программистом для Inno Setup, он будет называться **скриптом**.

Программа-инсталлятор, полученная в результате обработки (компиляции) скрипта Inno Setup'ом, будет именоваться **инсталлятором**.

Здесь и далее в статье сверху некоторых директив ;таким образом будет приведен комментарий к ним.

Создание инсталлятора с помощью Мастера

При выборе пункта меню Файл->Новый Inno Setup предложит вам создать инсталляцию с помощью мастера (Inno Setup Script Wizard). В большинстве случаев полученная с его помощью инсталляция вас не устроит, поэтому ее все равно нужно будет дорабатывать. Однако я не советую вам вообще пренебрегать мастером, т.к. он существенно облегчает создание хорошего инсталлятора, беря на себя всю «рутинную» работу.

Итак, для начала создадим простейший инсталлятор с помощью мастера Inno Setup Script Wizard. Для этого не нужно в окне приветствия ставить галочку против строки «Создать пустой сценарий».

Строки, обязательные для заполнения, подписаны жирным шрифтом. В скобках написаны **[Имя секции]**- директива. Директивам будут присвоены введенные значения.

В окне **Application Information** (Информация о приложении) следует ввести:
Имя приложения (**[Setup]**-AppName);
Версия приложения (**[Setup]**-AppVerName);
Авторское право (**[Setup]**-AppPublisher);
Домашняя страница (**[Setup]**-AppPublisherURL **[Setup]**-AppSupportURL **[Setup]**-AppUpdatesURL);

В окне **Application Directory** (Размещение приложения):
Базовый путь установки (**[Setup]**-DefaultDirName):
при выборе **Program Files Directory** по умолчанию программа будет установлена в папку **Program Files**, путь к которой буден взят из значения константы {pf};
при выборе **Custom** программа будет установлена в папку, расположение которой нужно будет указать ниже.
Папка приложения (**[Setup]**-DefaultGroupName) - имя группы меню Пуск по умолчанию
При снятом флажке *Разрешить Изменение Пути Пользователем* инсталятор не будет отображать страницу выбора папки. В этом случае папка по умолчанию не изменится (**[Setup]**-DisableDirPage=yes).
При выставленном флажке *Приложение Не Нуждается В Каталоге* папка приложения создаваться не будет, выбор последней предлагаться не будет, программа будет установлена в ту же папку, что и Windows (**[Setup]**-CreateAppDir=no).

В окне **Application Files**:
Главный файл приложения (**[Files]**- Source) – здесь будет прописан полный путь к главному файлу вашего приложения (главный файл в секции **[Files]** скрипта записывается первым). Также будут установлена папка, куда приложение устанавливается и необходимые (по мнению мастера) флаги, которые можно будет поменять или удалить.
Галочка против строки **Разрешить запуск приложения после установки** означает, что после установки инсталлятор предложит пользователю (по желанию) сразу запустить программу. В этом случае в скрипт добавится секция **[Run]** с указанием на запуск установленной программы.
В дополнение к главному файлу приложения можно добавить как угодно много дополнительных файлов, а также папок с вложенными в них файлами. Для этого достаточно щелкнуть на **Добавить файл(ы)** или **Добавить папку** и указать путь к ним в появившемся окне. После добавления установку файлов можно редактировать и удалить (при выделении файла из списка станут доступными кнопки **Правка** и **Удалить**).
При нажатии кнопки **Правка** откроется окно, в котором будет представлен достаточно широкий выбор параметров установки файла или папки. Рассмотрим его детально.
Опция **Включая подкаталоги** актуальна, понятно, для папок и говорит сама за себя.
В разделе **Путь** из выпадающего меню следует **Указать базовый каталог**, т.е. каталог, куда инсталлятор должен будет установить выбранный файл или папку. Это может быть:
Application Directory – папка приложения (выбирается по умолчанию, равен константе {app});
Program Files Directory – папка Program Files (равен константе {pf});
Common Files Directory – папка Common Files (равен константе {cf});
Windows Directory – папка Windows (равен константе {win});
Windows System Directory – системная папка Windows (равен константе {sys});
Setup Source Directory – папка инсталлятора (равен константе {src});
System Drive Root Directory – системный диск, на котором установлена Windows (равен константе {sd});
Common Startup Folder – папка Автозагрузка меню Пуск (равен константе {commonstartup});
User Startup Folder – папка Автозагрузка меню Пуск для текущего пользователя (равен константе {userstartup});
[Custom] – здесь можно указать другой каталог для установки файла (папки).
Ниже можно **Определить подкаталог** для устанавливаемого файла (папки) – в этом случае инсталлятор создаст подкаталог в каталоге устанавливаемой программы и установит файл (папку) туда. В скрипте это будет выглядеть так:
Source: "C:\Program.EXE"; DestDir: "{app}\KatalogProgram";

В окне **Application Icons**:
В строке **Папка меню Пуск** задается имя устанавливаемой программы – так оно будет отображаться в меню Пуск.
Значение флажков, расположенных ниже, понятно. А вот что при их установке добавится в скрипт:

Разрешить пользователю изменять имя папки меню Пуск:
DisableProgramGroupPage=yes
И как следствие этого флажок на следующей строчке станет недоступным, а в скрипт добавится:
AllowNoIcons=no (значение по умолчанию – не будет отображаться в скрипте)
Разрешить отмену создания папки меню Пуск:
AllowNoIcons=yes

Совет: следующие параметры инсталляции легче задавать вручную. Как именно это сделать, не прибегая к помощи мастера, будет рассмотрено позже, а пока просто вкратце перечислим, что значат установленные флажки в следующих строках.

Создавать ярлык Интернет в папке меню Пуск (в меню Пуск в папке устанавливаемой программы будет добавлена ссылка на сайт, указанный в AppPublisherURL);

Создавать значок Uninstall в папке меню Пуск (устанавливаемую программу можно будет удалить из меню Пуск);

Другое:

Разрешить пользователю создавать значки Рабочего Стола (пользователь может отказаться от установки значка программы на его рабочем столе);

Разрешить пользователю создание значков быстрого запуска (по желанию пользователь может добавить значок быстрого запуска программы, который будет располагаться в левом углу панели задач, справа от кнопки меню Пуск).

В окне **Application Documentation**:

Здесь следует указать путь к Файлу, задать **Информационный файл, выводимый перед установкой** и **Информационный файл, выводимый после установки**. Желательно, чтобы это были текстовые файлы (txt).

Следующее окно – **Финиш**. На нем работа мастера завершается. Думаете, написание инсталлятора на этом завершается? Ах, как вы ошибаетесь. С помощью мастера мы создали остов, фундамент будущего инсталлятора, то есть избавили себя от нудной работы. Впереди – самое интересное: сделать ваш инсталлятор действительно удобным, простым и красивым. В путь!

Секция [Setup]

Основные параметры секции [Setup].

Ниже приведена секция **[Setup]** с самыми необходимыми директивами. Это – минимальный набор для написания инсталятора.

```
[Setup]
; Имя программы
AppName=Programma
; Версия программы
AppVerName= Programma ver.1.0
; Определяет папку, в которую инсталятор предложит установить программу по умолчанию
DefaultDirName={pf}\ Programma 1.0
; Имя группы меню Пуск по умолчанию, которое используется на странице Выбора программной группы мастера
DefaultGroupName= Programma 1.0
; Следующая строка задает базовое имя для получаемого файла установки
OutputBaseFilename= Programma Setup 1.0
; Папка, в которой будут создаваться файлы установки относительно папки скрипта.
OutputDir=c:\Proba\
; Рабочая папка скрипта (откуда брать программулину, для которой пишется инсталлятор)
SourceDir=C:\
; Определяет, что флажок "Не создавать значков" на странице выбора программной группы будет отображаться
```

AllowNoIcons=yes
; инсталятор всегда будет показывать папку для установки в списке настроек на странице Всё готово к установке
AlwaysShowDirOnReadyPage=yes

;Эти директивы используются для определения содержимого вкладки "Поддержка" диалога Установка и удаление программ на Панели управления в Windows 2000/XP
AppPublisher=My Company, Inc.
AppPublisherURL=http://www.mycompany.com/
AppVersion=1.5
; При отрицательном значении (no), инсталятор не будет создавать пункта программы в диалоге Установка и удаление программ
CreateUninstallRegKey=yes
; При значении auto инсталятор выдаст сообщение "Папка: … уже существует. Вы хотите продолжить установку в эту папку?" ("The directory ... already exists. Would you like to install to that directory anyway?")
DirExistsWarning=auto
; Условия лицензии
LicenseFile=license.txt
; При значении yes, установка отобразит страницу для ввода информации о пользователе. Полученные значения будут в константах {userinfoname} и {userinfoorg}
UserInfoPage=yes

Некоторые директивы в этом примере указаны со значениями, принятыми по умолчанию – это значит, что их можно было не указывать.

Перезагрузка компьютера после удачной установки или удаления приложения

Для секции [Setup]:
UninstallRestartComputer=yes – компьютер будет перезагружен после удачного удаления программы, даже если в этом нет необходимости
AlwaysRestart=yes **–** компьютер будет перезагружен после удачной установки программы, даже если в этом нет необходимости
RestartIfNeededByRun=yes – компьютер будет перезагружен только если в этом нуждаются файлы, запускаемые из секции **[Run]**

Также инсталятор всегда будет предлагать пользователю перезагрузить компьютер, если для файлов секции **[Files]** применяются флаги **restartreplace** или **uninsrestartdelete**.

В секции **[Components]** за перезагрузку отвечает флаг **restart** – если при установке пользователь выберет компонент с этим флагом, после успешной установки инсталлятор предложит ему перезагрузить компьютер. Этот флаг следует использовать, если в секции **[Files]** имеются файлы с флагом **restartreplace**.

Использовать настройки предыдущей установки

Все директивы, перечисленные ниже, имеют данные значения по умолчанию, так что их указывать нет необходимости. Указываются они только в том случае, когда вам не нужно, чтобы инсталятор использовал какие-либо настройки предыдущей установки – тогда директивам присваивается значение no.
Инсталятор поищет в реестре, устанавливалось ли данное приложение на данный компьютер раньше, и если да, то по умолчанию предложит:

UsePreviousAppDir=yes – папку, которую пользователь выбрал при прошлой установке
UsePreviousGroup=yes – группу меню Пуск, которую указал пользователь при прошлой установке
UsePreviousSetupType=yes – тип и компоненты предыдущей установки
UsePreviousTasks=yes – задания, выбранные пользователем при предыдущей установке

UsePreviousUserInfo=yes – имя пользователя, название организации и серийный номер, которые ввел пользователь при предыдущей установке

Использование значений по умолчанию

DefaultDirName={pf}\MyProg – папка, в которую инсталятор предложит установить приложение по умолчанию. Это значение пользователь может изменить, если это предусмотрено инсталлятором

DefaultGroupName=MyProg – имя программы в меню Пуск, которое предложит инсталятор по умолчанию

Для того, чтобы работали приведенные далее директивы, нужно включить запрос имени пользователя и организации:

UserInfoPage=yes

DefaultUserInfoName=Lamer – имя пользователя, которое предложит инсталятор по умолчанию

DefaultUserInfoOrg=LamerLand – название организации, которое предложит инсталятор по умолчанию

«Косметические»

WizardImageFile=Image.bmp – картинка на приветствие (должна находиться в каталоге, обозначенном как SourceDir, либо к ней должен быть прописан полный путь).

WizardImageStretch=no – картинку лучше не растягивать под отведенный ей в окне инсталлятора размер

WizardSmallImageFile=SmallImage.bmp – маленькая картинка, во время инсталляции будет находиться в правом верхнем углу.

С помощью следующих директив можно изменить цвет фона за картинками:

WizardImageBackColor= clTeal

WizardSmallImageBackColor=clNavy

Значения цветов могут быть следующими:

clRed (красный)
clYellow (желтый)
clOlive (оливковый)
clLime (ярко зеленый)
clGreen (зеленый)
clTeal (салатовый)
clAqua (зелено-голубой)
clBlue (ярко-синий)
clNavy (синий)
clPurple (фиолетовый)
clFuchsia (розовый)
clMaroon (коричневый)
clBlack (черный)
clGray (серый)
clSilver (серебряный)
clWhite (белый)

Поработаем с фоновым окном инсталлятора.

WindowVisible=yes – включить поддержку фонового окна инсталлятора.

AppCopyright=Copyright © 2004 Zuka-Buka, Inc. - выводит сообщение об авторских правах в правом нижнем углу фонового окна инсталятора

BackColor=clLime – начальный цвет градиентной заливки фонового окна

BackColor2=clOlive – конечный цвет градиентной заливки фонового окна

BackColorDirection= lefttoright – направление градиентной заливки

BackSolid=no – это значение по умолчанию (градиентная заливка включена). При значении yes цвет фона будет сплошным и равным BackColor.

WindowShowCaption=no – инсталятор будет по-настоящему полноэкранным, фоновое окно закроет собой панель задач.

WindowStartMaximized=yes – фоновое окно инсталлятора будет развернуто на весь экран, но не перекроет панель задач (значение по умолчанию).
WindowResizable=no – пользователь не сможет поменять размер фонового окна инсталлятора, когда оно развернуто.

Прочие «косметические» директивы:
FlatComponentsList=yes – задает внешний вид checkbox'ов (yes – «плоские», no – «3D»).
SetupIconFile=Ikonka.ico – задает иконку для полученного инсталлятора
ShowComponentSizes=yes – показывать размер устанавливаемых компонентов (значение по умолчанию). Удобная опция, лучше не отключать ее.
ShowTasksTreeLines=yes – инсталятор соединит линиями главное задание и относящиеся к нему подзадания.

Деинсталятор

Ниже приведены примеры использования директив деинсталлятора с краткими пояснениями. Для долее детальной информации обратитесь к русскому файлу справки (раздел Секция **[Setup]**)
Приведенные ниже директивы полезно использовать, если ваша программа является обновлением или дополнением к какой-либо другой программе.

CreateUninstallRegKey=no – не создавать запись в меню *Установка и удаление программ* Панели управления
Uninstallable=no – не создавать деинсталлятор
UninstallLogMode=append – по возможности дополнить деинсталлятор – в этом случае при удалении отменяются все изменения, внесенные различными инсталяторами
UpdateUninstallLogAppName=no – при дополнении деинсталлятора его заголовок не изменится

А эти директивы являются «косметическими» - определяют внешний вид деинсталлятора:

UninstallDisplayIcon= "C:\Ikonka1.ico" – значок для деинсталлятора в диалоге *Установка и удаление программ* Панели управления
UninstallDisplayName=Удалить Proga 1.0 – название программы в диалоге *Установка и удаление программ* Панели управления
UninstallFilesDir= "C:\UninstProga" – папка, в которую будет помещен деинсталятор
UninstallIconFile= "C:\Ikonka2.ico" – иконка, которая будет отображаться в правом верхнем углу окна *Процесс удаления*
UninstallStyle=classic – стиль деинсталятора

Остальные параметры секции **[Setup]** подробно описаны в **ISetup.hlp** – файле справки Inno Setup.

Примеры использования

Отключить приветствие

DisableStartupPrompt=no
При значении yes, инсталлятор не будет отображать сообщение "Эта программа установит… на ваш компьютер…" ("This will install... Do you wish to continue?). Игнорируется когда UseSetupLdr=no.

Изменение приложения из панели управления

При использовании директивы AppModifyPath в Windows 2000/XP на странице *Установка и удаление программ* Панели управления против заголовка программы рядом с кнопкой **Удалить** добавится кнопка **Изменить**. Нажав на нее, пользователь сможет установить понадобившиеся ему компоненты программы, и для этого ему не придется переустанавливать приложение целиком. Сейчас рассмотрим скрипт, который просто покажет работу этой директивы.

Задача: приложение состоит из трех .exe-файлов: Matrix.exe, 2X.exe и Plus.exe. Для него возможны два типа установки: полная и выборочная (если выбраны все 3 компонента программы, установка полная, если не все – выборочная). С помощью директивы AppModifyPath дадим возможность пользователю устанавливать нужные ему компоненты приложения.

Решение:

В этом примере местонахождение программы-инсталлятора будет определено в секции **[Code]**. При установке программы в реестре системы пользователя создастся ключ, с помощью которого в дальнейшем определится путь к инсталлятору.

```
[Setup]
AppName=Vibor
AppVerName=Vibor 1.0
DefaultDirName={pf}\Vibor 1.0
DefaultGroupName=Vibor 1.0
;Имя полученного инсталятора
OutputBaseFilename=Vibor 1.0
SourceDir=C:\
AllowNoIcons=yes
;Во избежание каких-либо недоразумений отключим использование настроек предыдущей установки:
;не использовать папку, которую выбрал пользователь в прошлый раз
UsePreviousAppDir=no
;не использовать группу меню Пуск, выбранную пользователем при предыдущей установке
UsePreviousGroup=no
;не использовать тип установки, выбранный пользователем в прошлый раз
UsePreviousSetupType=no
;не использовать задания, которые выбрал пользователь при прошлой установке
UsePreviousTasks=no
AlwaysShowDirOnReadyPage=yes
; директива AppModifyPath определит местонахождение программы-инсталлятора с помощью описанной в секции [Code] константы MyConst
AppModifyPath="{code:MyConst}"


[Types]
Name: "polnaya"; Description: "Все компоненты"
;флаг iscustom означает, что именно этот тип установки является выборочным
Name: "viborochnaya"; Description: "Выбрать компоненты"; Flags: iscustom

;создаем список компонентов
[Components]
Name: "Matrix"; Description: "Программа Matrix 1.0"; Types: polnaya viborochnaya
Name: "2X"; Description: "Программа 2X"; Types: polnaya viborochnaya
Name: "Plus"; Description: "Программа сложения"; Types: polnaya viborochnaya

[Files]
Source: "Matrix.exe"; Components: Matrix; DestDir: "{app}"
Source: "2X.exe"; Components: 2X; DestDir: "{app}"
Source: "Plus.exe"; Components: Plus; DestDir: "{app}"

;создаем ключ реестра: корневой ключ HKEY_CURRENT_USER, раздел Software\Vibor, параметр InstallSettings со строковым значением, которое содержит полный путь к файлу инсталлятора ({src}\Vibor 1.0.exe)
[Registry]
Root: HKCU; Subkey: "Software\Vibor"; ValueType: String; ValueName: InstallSettings; ValueData: "{src}\Vibor 1.0.exe"; Flags: uninsdeletekey
```

;и небольшой код, в котором описывается константа MyConst – с ее помощью будет определен путь к инсталлятору, которым будет пользоваться директива AppModifyPath при изменении программы из Панели управления

[Code]

```
function MyConst(Default: String): String;
var
ResultStr: String;
  begin
    RegQueryStringValue(HKCU, 'Software\Vibor', 'InstallSettings', ResultStr);
    Result:=ResultStr;
  end;
```

Скрипт считывает из реестра значение параметра InstallSettings и присваивает его строковой переменной ResultStr, после чего она присваивается значению функции Result.

Директива AppModifyPath может по-разному вычислять местонахождение инсталлятора (не только с помощью записи в реестре) – применяйте методы на свой вкус.

Работа всех секций, приведенных в данном примере, будет подробнее рассмотрена ниже.

Если приложение не нуждается в директории

Допустим, что написанное вами приложение не нуждается в собственной директории, а входящие в него файлы должны установиться в разные уже существующие директории в системе пользователя. Рассмотрим задачу на это тему.

Задача: Приложение является надстройкой для Windows. Естественно, все его файлы должны быть установлены в папку Windows.

Решение:

Директива CreateAppDir отвечает за создание папки приложения. Если CreateAppDir=yes (по умолчанию), для приложения нужно будет обязательно указывать DefaultDirName. Если же CreateAppDir=no, папка для приложения создаваться не будет, выбор последней предлагаться не будет, константа {app} будет равна {win}, т.е. приложение установится в папку Windows. Если при этом включено создание деинсталлятора, он будет создан в системной папке Windows. Скрипт будет примерно следующий:

```
[Setup]
AppName=Proga
AppVerName=Proga 1.0
AppPublisher=My Company, Inc.
AppPublisherURL=http://www.mycompany.com
;задаем не создавать папки для приложения (оно будет установлено в папку Windows)
CreateAppDir=no
Compression=lzma
;деинсталлятор будет создан и помещен в системную папку
UninstallFilesDir="{app}\System"

[Files]
;флаг ignoreversion означает, что версии файлов будут игнорироваться
Source: "C:\File1.EXE"; DestDir: "{win}"; Flags: ignoreversion
Source: "C:\File2.EXE"; DestDir: "{win}"; Flags: ignoreversion
Source: "C:\File3.EXE"; DestDir: "{app}\System32"; Flags: ignoreversion
```

В данном скрипте специально используются две константы: {app} и {win}, для того, чтобы продемонстрировать, что при CreateAppDir=no они означают одно и то же.

Если же ваш инсталятор должен установить файлы не в папку Windows, то в DefaultDirName указывается корневой каталог, а в секции **[Files]** в DestDir после {app} указываются папки, в которые должны установиться файлы. Например, инсталятор

```
[Setup]
```

AppName=Proga
AppVerName=Proga 1.0
AppPublisher=My Company, Inc.
AppPublisherURL=http://www.mycompany.com
DefaultDirName={pf}
UsePreviousAppDir=no
UsePreviousGroup=no
Compression=lzma

[Files]
Source: "C:\File1.EXE"; DestDir: "{app}\Directory1"; Flags: ignoreversion
Source: "C:\File2.EXE"; DestDir: "{app}\Directory2"; Flags: ignoreversion
Source: "C:\File3.EXE"; DestDir: "{app}"; Flags: ignoreversion

установит файл File1.exe в папку C:\Program Files\Directory1, файл File2.exe в папку C:\Program Files\Directory2, файл File3.exe в папку C:\Program Files. Деинсталлятор будет расположен в папке C:\Program Files. Создание деинсталлятора можно отключить с помощью директивы Uninstallable (если эта директива равна no, деинсталлятор создаваться не будет – автоматическое удаление приложения будет невозможно).

Если пользователю нельзя менять путь установки приложения, используйте директиву DisableDirPage – при значении yes страница *Выбор папки назначения* отображаться не будет, а приложение установится в папку, заданную в DefaultDirName.

Если приложение является обновлением для установленной программы, разумнее будет воспользоваться директивой AppId – о ней подробнее написано ниже. Если же вы создали обновление для программы, уже установленной в системе пользователя, у вас нет возможности воспользоваться директивой AppId, и вы не знаете, где именно установлена эта программа на компьютере пользователя, ему придется самому указать путь к папке программы на странице *Выбор папки установки*. Хотя именно в этом случае местонахождение уже установленного приложения в системе пользователя можно найти программно – с помощью секции **[Code]**. В данной статье приведено несколько таких способов: ниже, при описании секции **[Ini]**, и выше, при описании директивы AppModifyPath секции **[Setup]**

Предупреждения при установке в существующую папку

За появление сообщения «Папка … уже существует. Вы хотите продолжить установку в эту папку?» отвечает директива DirExistsWarning. При DirExistsWarning=auto инсталятор выдаст это сообщение, если UsePreviousAppDir=no или установка производится в папку впервые. При UsePreviousAppDir=yes сообщение появится только при первой установке. Если DirExistsWarning=yes, сообщение будет появляться всегда, а если DirExistsWarning=no, то не будет появляться никогда.

Предупреждение при установке в несуществующую папку

За появление сообщения «Папка … не существует. Создать?» отвечает директива EnableDirDoesntExistWarning. Если EnableDirDoesntExistWarning=yes, сообщение будет появляться, а если EnableDirDoesntExistWarning=no, то не будет. Также стоит обратить внимание на то, что директива эта будет действовать только при DirExistsWarning=no.

Информация, отображаемая до и после установки

Информацию можно отобразить несколькими способами: в лицензионном соглашении, информации на странице самого инсталлятора или в открывающемся окне Блокнота. Текстовые файлы, используемые при этом, должны находиться либо в папке скрипта, либо к ним должен быть указан полный путь. Рассмотрим каждый из вариантов подробнее.

1. Лицензионное соглашение

LicenseFile=license.txt. Для файла лицензии лучше всего использовать текстовый файл. В файле лицензии указываются авторские права, правила распространения программы и другая подобная информация, но никак не руководство по установке программы и комментарии автора.

2. Информация перед установкой

InfoBeforeFile=MyInfoBefore.txt. Эта информация будет отображена перед страницей *Выбор папки установки* прямо в окне инсталлятора. Здесь можно поместить советы по установке программы и другую информацию, которая будет полезна пользователю для успешной инсталляции вашего приложения.

3. Информация после установки

InfoAfterFile=MyInfoAfter.txt. Эта информация будет отображена после успешной установки приложения прямо в окне инсталлятора.

4. Информация на странице Блокнота
А) отображается после удачной установки (перед страницей *Установка завершена*)

За запуск приложений после удачной установки отвечает секция **[Run]**. Скрипт может быть следующим:

```
[Run]
Filename: "{app}\ReadMe.txt"; Flags: shellexec skipifsilent
```

В этом случае файл ReadMe.txt запустится, хочет того пользователь или нет.

```
[Run]
Filename: "{app}\ReadMe.txt"; Description: "Хочу прочитать файл ReadMe"; Flags: postinstall shellexec skipifsilent
```

В случае использования флага **postinstall** файл ReadMe.txt отобразится только в том случае, если пользователь поставит галочку против строки «Хочу прочитать файл ReadMe».

Кроме того, файл, который должен будет отобразиться после удачной установки, можно включить и в секцию **[Files]**:

```
[Files]
Source: "C:\ReadMe.txt"; DestDir: "{app}"
```

В этом случае файл ReadMe.txt будет установлен в папку приложения, и пользователь сможет его прочитать позже.

Также эта задача может быть решена проще: с помощью флага **isreadme** секции **[Files]**. Но это возможно только в случае, когда ваш инсталятор не должен перезагрузить компьютер пользователя после успешной установки. В противном случае пользователь не сможет прочитать информацию сразу после установки программы.

Секция [Tasks]

Бывает так, что в ходе установки инсталятор должен выполнить какие-либо задания. Эти задания описываются в секции **[Tasks]**. Следует помнить, что сама по себе секция **[Tasks]** ничего не запускает и не создает. Чтобы задание действительно выполнилось, его надо вызвать из той секции, к которой оно относится. Для того чтобы стал понятен принцип работы этой секции, рассмотрим несколько практических задач.

Задания для создания ярлыков

Для задания для вынесения ярлыка на рабочий стол используем секции **[Tasks]** и **[Icons]**.

… … … … … … … … … … …
[Tasks]
Name: "DesktopIcon"; Description: "Вынести ярлык на рабочий стол"; Flags: unchecked
[Icons]
Name: "{userdesktop}\Program"; IconFilename: "C:\Ikonka.ico"; Filename: "{app}\Program.EXE"; Tasks: DesktopIcon
… … … … … … … … … … …

В секции **[Tasks]** задание описывается. В окне инсталлятора оно появится после удачной установки приложения, перед страницей *Установка завершена*, и будет иметь вид строки, значение которой определит параметр Description. Задание выполнится, если пользователь поставит флажок в checkbox'е в начале этой строки.

В секции **[Icons]** описывается, куда следует поместить ярлык, как ярлык будет озаглавлен и (по желанию) как он будет выглядеть, для какого именно файла он создается (подробнее работа этой секции будет рассмотрена далее). Параметр Tasks секции **[Icons]** вызывает задание, описанное в секции **[Tasks]**. Если этот параметр не указать, то ярлык будет вынесен на рабочий стол, даже если пользователь не поставил флажок против строки «Вынести ярлык на рабочий стол».

Так выглядит задание для добавления программы в автозапуск:

… … … … … … … … … … …
[Tasks]
Name: "AutoLaunch"; Description: "Добавить в автозапуск"; Flags: unchecked
[Icons]
Name: "{commonstartup}\Proga"; IconFilename: "C:\Ikonka.ico"; Filename: "{app}\Program.EXE"; Tasks: AutoLaunch
… … … … … … … … … … …

Таким же образом создаются задания для помещения ярлыков в папку Мои документы (в секции **[Icons]** используется константа {userdocs}), Избранное (в секции **[Icons]** используется константа {userfavorites}), а также для добавления имени программы в контекстное меню **Отправить** (в секции **[Icons]** используется константа {sendto}).

Для добавления ярлыка программы в меню Пуск задание создавать не придется – достаточно присвоить директиве AllowNoIcons секции **[Setup]** значение yes – тогда пользователь на странице *Выбор программной группы* может выбрать, помещать ли ярлык программы в меню Пуск. Куда именно будет размещен ярлык программы – в главное меню Пуск или какую-то его папку, нужно указать в секции **[Icons]** (об этом подробнее написано в разделе Секция **[Icons]**).

Для запуска каких-либо файлов секция **[Tasks]** не подходит – для этого существует секция **[Run]**.

Секция [Icons]

Секция **[Icons]** отвечает за создание ярлыков при установке программы. Следует помнить, что если ярлыки указаны только в секции **[Icons]**, они будут создаваться всегда, а если для ярлыка, описанного в секции **[Icons]**, задан предварительно описанный в секции **[Tasks]** параметр Tasks, то ярлык будет создан, только если пользователь пожелает его создать (т.е. выберет соответствующее задание при установке программы).

Примеры создания ярлыков

[Icons]
;Создаст группу программы в меню Пуск
Name: "{group}\Proga"; IconFilename: "C:\Ikonka.ico"; Filename: "{app}\Program.exe"
;Создаст ярлык программы на Рабочем столе.
Name: "{userdesktop}\Proga"; IconFilename: "C:\Ikonka.ico"; Filename: "{app}\ Program.exe"
;Добавит ярлык программы в контекстное меню **Отправить**

Name: "{sendto}\Proga"; IconFilename: "C:\Ikonka.ico"; Filename: "{app}\ Program.exe "
;Добавит ярлык программы в **Автозапуск**
Name: "{commonstartup}\Proga"; IconFilename: "C:\Ikonka.ico"; Filename: "{app}\ Program.exe "

Как видно из примеров, месторасположение ярлыка зависит от заданной в параметре Name константы.

Если куда-либо выносится ярлык для программы MS-DOS, пользователю будет приятно, если после выполнения программы окно закроется автоматически. Для этого используется флаг **closeonexit**:

[Icons]
Name: "{userdesktop}\Proga"; IconFilename: "C:\Ikonka.ico"; Filename: "{app}\Program.exe"; Flags: closeonexit

Всплывающую подсказку (хинт) для ярлыка на рабочем столе можно задать так:

[Icons]
Name: "{userdesktop}\Proga"; IconFilename: "C:\Ikonka.ico"; Filename: "{app}\ Program.exe"; Comment: "Это ярлык к программе Program"

Хинты будут отображаться только начиная с Windows Me. Более ранние версии Windows этот параметр не поддерживают. Также хинт не будет отображаться для ярлыков к программам MS-DOS.

Так создастся группа для программы в меню Пуск > Программы:

[Icons]
Name: "{group}\Proga"; IconFilename: "C:\Ikonka.ico"; Filename: "{app}\Program.exe"

Так – просто ярлык в меню Пуск > Программы:

[Icons]
Name: "{commonprograms}\Proga"; IconFilename: "C:\Ikonka.ico"; Filename: "{app}\Program.exe"

Так ярлык добавится в главное меню Пуск:

[Icons]
Name: "{userstartmenu}\Proga"; IconFilename: "C:\Ikonka.ico"; Filename: "{app}\Program.exe"

Таким же образом создаются ярлыки в папках Мои документы (в секции **[Icons]** используется константа {userdocs}), Избранное (в секции **[Icons]** используется константа {userfavorites}).

Подробнее об использованных в примерах константах, а также о тех константах, которые не были упомянуты, можно прочитать в русском файле справки **ISetup.hlp** (раздел **Константы**).

Ярлык деинсталлятора в меню Пуск можно поместить так:

[Icons]
;Создаст группу программы в меню Пуск, а в ней ярлык для запуска программы Program.exe
Name: "{group}\Proga"; IconFilename: "C:\Ikonka.ico"; Filename: "{app}\Program.exe"
;Создаст в той же группе ярлык для деинсталлятора программы Program.exe
Name: "{group}\UninstallProga"; Filename: "{uninstallexe}"

Параметру Filename можно было присвоить и значение {app}\unins000.exe – в данном случае эти значения равны.

Задать «горячую клавишу» для ярлыка можно так (в комментарии к ярлыку следует указать это сочетание клавиш):

[Icons]
Name: "{userdesktop}\Proga"; IconFilename: "C:\Ikonka.ico"; Filename: "{app}\Program.exe"; Comment: "Для запуска нажать ctrl+alt+k"; HotKey: "ctrl+alt+k"

Для того, чтобы программа при выборе ярлыка запускалась в свернутом окне, используется флаг **runminimized**:

[Icons]
Name: "{userdesktop}\Proga"; IconFilename: "C:\Ikonka.ico"; Filename: "{app}\ Program.exe”; Flags: runminimized

Секции [Components] и [Types]

В секции **[Types]** перечисляются типы установки программы. Следует помнить, что сама по себе секция **[Types]** только описывает возможные для данной программы типы установки, и должна комбинироваться с другими секциями.

В секции **[Components]** перечисляются компоненты, которые инсталлятор отобразит на странице *Выбор компонентов*. Комбинируется с секцией **[Types]**, причем если в секции **[Components]** перечислены компоненты, а секции **[Types]** не существует, типы установки будут созданы инсталлятором автоматически, и могут быть не такими, как вам хотелось бы.

Работу секций рассмотрим на примерах.

Выбор устанавливаемых компонентов

Задача: имеется набор компонентов. По желанию пользователь может установить все компоненты или выбрать нужные ему из списка.

Решение:

В секции **[Types]** перечисляются все типы установки с их описанием. Для данной задачи можно создать два типа, например **Полная установка** и **Выборочная установка**, причем не забудьте указать компилятору, какая именно из установок является выборочной флагом **iscustom**.

В секции **[Components]** перечисляются компоненты для установки с описанием и указанием типов установки (параметр Types), к которым они относятся.

В секции **[Files]** перечисляются файлы, которые собственно и будут установлены. Обязательно укажите, к какому компоненту относится каждый файл (параметр Components).

Многоточия в начале и конце примера означают, что вверху должна находиться секция **[Setup]** с необходимыми параметрами, а внизу могут идти другие секции (например, **[Tasks]** или **[Icons]**).

……
[Types]
Name: "polnaya"; Description: "Полная установка"
Name: "viborochnaya"; Description: "Выборочная установка"; Flags: iscustom
[Components]
Name: "Component1"; Description: "Описание: программа первая"; Types: polnaya viborochnaya
Name: " Component 2"; Description: "Описание: программа вторая"; Types: polnaya viborochnaya
Name: " Component 3"; Description: "Описание: программа третья"; Types: polnaya viborochnaya
[Files]
Source: "Program1.exe"; Components: Component1; DestDir: "{app}"
Source: " Program2.exe"; Components: Component2; DestDir: "{app}"
Source: " Program3.exe"; Components: Component3; DestDir: "{app}"
……

Создать группу компонентов

Бывает так, что при установке компоненты для удобства и наглядности нужно разделить на группы, и из каждой из групп пользователь может выбрать один или несколько компонентов.

Так создается группа компонентов:

[Components]
Name: "Gruppa"; Description: "Заголовок группы"; Types: polnaya

Name: "Gruppa\Component1"; Description: "Компонент 1"; Types: polnaya
Name: "Gruppa\Component2"; Description: "Компонент 2"; Types: polnaya

Если из двух компонентов группы должен быть выбран только один, для них используется флаг **exclusive**:

[Components]
Name: "Gruppa"; Description: "Заголовок группы"; Types: polnaya
Name: "Gruppa\Component1"; Description: "Компонент 1"; Types: polnaya; Flags: exclusive
Name: "Gruppa\Component2"; Description: "Компонент 2"; Types: polnaya; Flags: exclusive
[Files]
Source: "Program1.exe"; Components: Gruppa\Component1; DestDir: "{app}"
Source: "Program2.exe"; Components: Gruppa\Component2; DestDir: "{app}"

Лучше не объединять в группу компоненты, которые являются взаимоисключающими (имеют флаг **exclusive**) и не являются таковыми – пользователь наверняка запутается при выборе компонентов. Компоненты, из которых может быть выбран только один, лучше объединить в отдельную группу.

«Зафиксировать» выбор компонента

Если выбор компонента нельзя доверить пользователю – он должен быть выбран или не выбран в любом случае, используется флаг **fixed**:

[Components]
Name: "Component"; Description: "Компонент"; Types: polnaya viborochnaya; Flags: fixed

При использовании этого флага, какой бы тип установки ни выбрал пользователь, компонент с флагом **fixed** будет всегда выбран (или не выбран), и пользователь не сможет этого изменить.

Не выбирать автоматически компоненты группы

Если вы не хотите, чтобы при выборе заголовка группы автоматически выбирался принадлежащий ей компонент (обычно при выборе заголовка группы автоматически выбираются все ее компоненты), используется флаг **dontinheritcheck**:

[Components]
Name: "Gruppa"; Description: "Заголовок группы"; Types: polnaya
Name: "Gruppa\Component1"; Description: "Компонент 1"; Types: polnaya; Flags: dontinheritcheck
Name: "Gruppa\Component2"; Description: "Компонент 2"; Types: polnaya

Этот флаг не комбинируется с флагом **exclusive**.

Секция [Dirs]

Секция в основном используется для создания пустых папок, которые создаст инсталлятор помимо папки приложения.

Создадим подпапку в папке устанавливаемой программы:

[Dirs]
Name: "{app}\Directory"

Так папка создастся в папке Мои документы у текущего пользователя:

[Dirs]
Name: "{userdocs}\Directory"

Также создаваемые папки могут иметь атрибуты: «системная» (system), «только для чтения» (readonly) и «скрытая» (hidden). За атрибуты папки отвечает параметр Attribs, и если он не указан, то папка не будет иметь вообще никаких атрибутов.

[Dirs]
Name: "{app}\Directory"; Attribs: readonly hidden

Флаг **deleteafterinstall** указывает инсталлятору создать папку, но после успешной установки или если установка прервана удалить ее, если она пустая. Если в папке будут содержаться файлы, с помощью этого флага она не удалится.

Флаг **uninsalwaysuninstall** указывает деинсталлятору всегда пытаться удалять папку (если она пустая), даже если она уже существовала до начала установки приложения.

Флаг **uninsneveruninstall** указывает деинсталлятору не удалять папку при удалении программы – обычно при удалении программы деинсталлятор удаляет все папки, которые были созданы инсталлятором при установке (если они пустые).

Секция [Files]

В этой секции указываются все файлы, которые инсталлятор может установить на компьютер пользователя. Примеры использования секции **[Files]** в статье уже встречались. Рассмотрим принцип ее работы подробнее.

Установить файл в папку приложения

Установить файл Program.exe в папку приложения:

[Files]
Source: "C:\Program.exe"; DestDir: "{app}"

Установить файл Program.exe в подпапку Papka папки приложения:

[Files]
Source: "C:\Program.exe"; DestDir: "{app}\Papka"

Подпапку, указанную в DestDir (в данном случае Papka) инсталлятор создаст автоматически.

Задание компонентов

Рассматривая секции **[Components]** и **[Types]**, разбирался пример, как указать инсталлятору, к какому из компонентов или к какой группе компонентов относится файл:

[Types]
Name: "polnaya"; Description: "Полная установка"
Name: "viborochnaya"; Description: "Выборочная установка"; Flags: iscustom
[Components]
Name: "Component1"; Description: "Описание: программа первая"; Types: polnaya viborochnaya
[Files]
Source: "Program1.exe"; Components: Component1; DestDir: "{app}"

Если файл является файлом ReadMe

Если устанавливаемый файл является файлом ReadMe, инсталлятор после удачной установки приложения может предложить пользователю прочесть этот файл. Для этого используется флаг **isreadme** секции **[Files]**, но файл ReadMe пользователь сможет прочитать, только если после установки компьютер не будет перезагружен.

[Files]
Source: "ReadMe.txt"; DestDir: "{app}"; Flags: isreadme

Чтобы файл с именем Program.exe после установки стал называться Proga.exe:

[Files]
Source: "C:\Program.exe"; DestDir: "{app}"; DestName: "Proga.exe"

Атрибуты файлов

Также файлы секции **[Files]** могут иметь атрибуты: «системный» (system), «только для чтения» (readonly) и «скрытый» (hidden). За атрибуты файла отвечает параметр Attribs, и если он не указан, то файл не будет иметь вообще никаких атрибутов.

[Files]
Source: "C:\Program.exe"; DestDir: "{app}"; Attribs: system hidden

Замена существующих файлов

1) Установить файл, который, возможно, уже установлен на компьютере пользователя и имеет более позднюю дату создания, т.е. является более новым (в этом случае инсталлятор по умолчанию оставит существующий файл), поможет флаг **ignoreversion**.

[Files]
Source: "C:\Program.exe"; DestDir: "{app}"; Flags: ignoreversion

2) Если файл, который должен быть установлен на компьютер пользователя, имеет более позднюю версию или более раннее время создания (т.е. является более старым), чем уже установленный файл, и пользователь сам должен выбрать, заменять ли существующий файл более старым или оставить уже установленный более новый, используется флаг **promptifolder**. В таких ситуациях инсталлятор по умолчанию оставит уже установленный (более новый) файл.

[Files]
Source: "C:\Program.exe"; DestDir: "{app}"; Flags: promptifolder

3) Если файл, устанавливаемый из секции **[Files]**, уже установлен на компьютере пользователя, и должен быть заменен или не заменен по желанию пользователя, используйте флаг **confirmoverwrite**.

[Files]
Source: "C:\Program.exe"; DestDir: "{app}"; Flags: confirmoverwrite

В этом случае инсталлятор при установке спросит пользователя, заменить ли существующий файл.

4) Если инсталлируемый файл должен заменить уже установленный на компьютере пользователя файл с тем же именем, а если такого файла в системе пользователя нет, то вообще не устанавливаться, можно использовать флаг **onlyifdestfileexists**. Этот флаг полезен в том случае, когда вы создаете инсталлятор для обновления (апдейта) уже установленной у пользователя программы. Но следует помнить, что пользователь может переименовать некоторые файлы программы, и тогда обновление не установится, даже если пользователь установил программу.

[Files]
Source: "C:\Program.exe"; DestDir: "{app}"; Flags: onlyifdestfileexists

Обратная задача: установить файл только в том случае, если он не существует в системе пользователя, решается с помощью флага **onlyifdoesntexist**.

5) Если на компьютере пользователя уже установлен файл с данным именем и атрибутом readlnly («только для чтения»), при установке инсталлятор спросит у пользователя разрешения заменить существующий файл на новый. Если этого происходить не должно, т.е. файл с атрибутом readonly должен

всегда устанавливаться на компьютер пользователя, хочет пользователь того или нет, для файла указывается флаг **overwritereadonly**.

```
[Files]
Source: "C:\Program.exe"; DestDir: "{app}"; Flags: overwritereadonly
```

6) Если уже установленный и устанавливаемый файлы имеют одну версию, по умолчанию инсталлятор не заменит уже установленный файл. Если установленный файл должен быть заменен (что рекомендуется), используется флаг **replacesameversion**.

```
[Files]
Source: "C:\Program.exe"; DestDir: "{app}"; Flags: replacesameversion
```

О других флагах, поддерживаемых секцией **[Files]**, можно прочитать в файле справки (раздел **Секция [Files]**).

Установка архивов

Если по какой-либо причине вы не хотите использовать методы сжатия файлов, которые поддерживает Inno Setup, можно использовать для сжатия любой архиватор. Файлы для примеров в этой статье будут сжаты с помощью архиватора WinRAR.

Если устанавливаемые файлы уже сжаты архиватором, внутреннее сжатие следует отключить: присвоив директиве Compression секции **[Setup]** значение none, сжатие вообще не будет использоваться, а если в секции **[Files]** задать отдельному файлу флаг **nocompression**, то инсталлятор не будет сжимать только заданный файл.

Устанавливаются сжатые файлы так же, как и обычные:

```
[Files]
Source: "C:\Program.rar"; DestDir: "{app}"; Flags: nocompression
```

Те же правила установки применяются и к самораспаковывающимся архивам:

```
[Files]
Source: "C:\Program.sfx.exe"; DestDir: "{app}"; Flags: nocompression
```

Самораспаковывающийся архив после установки можно запустить из секции **[Run]**, а после распаковки (параметры которой можно задать при создании архива) удалить его. Это может выглядеть так:

```
[Files]
Source: "C:\Program.sfx.exe"; DestDir: "{tmp}"; Flags: nocompression
```

Флаг **deleteafterinstall** в данном случае можно не использовать – архив устанавливается в папку Temp, откуда потом удалится автоматически.

Секция [Ini]

Секция **[Ini]** работает с INI-файлами. Сам формат скрипта Inno Setup очень похож на INI-файл. Если INI-файл с заданным именем не существует в указанной папке, инсталлятор создаст его, а если INI-файл уже существует, он может добавить, удалить или изменить записи в нем, а также удалить существующий INI-файл и создать новый с тем же именем, но другим содержанием.

Параметры секции

Filename – имя INI-файла (с расширением .ini).
Section – имя секции INI-файла (в полученном INI-файле будет заключено в квадратные скобки).

Key – ключ INI-файла.
String – значение ключа Key (строка).

Так секция Settings INI-файла IniFile.ini будет содержать ключ Version равный 1.0.1:

[Ini]
Filename: "{win}\IniFile.ini"; Section: "Settings"; Key: "Version"; String: "1.0.1"

Определение местонахождения установленного приложения

Пользователь может установить ваше приложение в своей системе куда угодно – это его право. Допустим, что спустя время вы написали обновление для этого приложения, и оно должно быть установлено в ту же папку, что и программа. Перед вами встанет задача – определить, куда именно было установлено ваше приложение, и решить ее можно как минимум двумя способами: 1) при установке обновления пользователь сам укажет путь к папке, в которую он установил программу, на странице *Выбор папки установки*; 2) путь будет вычислен программно в секции **[Code]**.

Определение пути к определенной папке с помощью записи в реестре мы уже рассмотрели, когда изучали принцип работы директивы AppModifyPath секции **[Setup]**. Теперь определим путь к папке приложения с помощью записи в INI-файле.

Задача: приложение Program состоит из двух .exe-файлов: Matrix.exe и 2X.exe. Обновление должно добавить в папку уже установленной программы файл Plus.exe, заменить файл Matrix.exe на MatrixUp.exe и добавить в файл IniFile.ini, который находится в папке Windows, запись о версии программы.

Решение:

В этом примере местонахождение установленной программы будет определено в секции **[Code]**. При установке программы в папке Windows системы пользователя создастся INI-файл, с помощью которого в дальнейшем определится путь к папке приложения (в этом состоит недостаток метода: придется позаботиться об установке обновления заранее, еще при написании инсталлятора для самой программы).

Так будет выглядеть скрипт для программы:

```
[Setup]
AppName=Program
AppVerName=Program 1.0
DefaultDirName={pf}\Program 1.0
DefaultGroupName=Program 1.0
;Имя полученного инсталятора
OutputBaseFilename=Program 1.9
SourceDir=C:\
AllowNoIcons=yes
;Во избежание каких-либо недоразумений отключим использование настроек предыдущей установки:
;не использовать папку, которую выбрал пользователь в прошлый раз
UsePreviousAppDir=no
;не использовать группу меню Пуск, выбранную пользователем при предыдущей установке
UsePreviousGroup=no
;не использовать тип установки, выбранный пользователем в прошлый раз
UsePreviousSetupType=no
;не использовать задания, которые выбрал пользователь при прошлой установке
UsePreviousTasks=no
AlwaysShowDirOnReadyPage=yes

[Files]
Source: "Matrix.exe"; DestDir: "{app}"
Source: "2X.exe"; DestDir: "{app}"

;создаем в INI-файле IniFile.ini секцию InstallSettings и в ней ключ InstallPath со строковым значением, равным полному пути к файлу инсталятора ({src}\Vibor 1.0.exe).
```

[Ini]
Filename: "{win}\IniFile.ini"; Section: "InstallSettings"; Key: "InstallPath"; String: "{app}"

;INI-файлы не удаляются деинсталлятором при удалении программы, поэтому, если вы все-таки хотите, чтобы они удалились вместе с программой, удаление надо задать в секции **[UninstallDelete]**
[UninstallDelete]
Type: files; Name: "{win}\IniFile.ini"

Так – для обновления:

[Setup]
AppName=Program
AppVerName=Program 1.1
;местонахождение папки приложения определится в секции **[Code]**
DefaultDirName={code:MyConst}
DefaultGroupName=Program 1.1
;Имя полученного инсталятора
OutputBaseFilename=Program 1.1
SourceDir=C:\
AllowNoIcons=yes
;запрещает пользователю выбирать папку для установки обновления – страница *Выбор папки назначения* отображаться не будет
DisableDirPage=yes
;Во избежание каких-либо недоразумений отключим использование настроек предыдущей установки:
;не использовать папку, которую выбрал пользователь в прошлый раз
UsePreviousAppDir=no
;не использовать группу меню Пуск, выбранную пользователем при предыдущей установке
UsePreviousGroup=no
;не использовать тип установки, выбранный пользователем в прошлый раз
UsePreviousSetupType=no
;не использовать задания, которые выбрал пользователь при прошлой установке
UsePreviousTasks=no
AlwaysShowDirOnReadyPage=yes

;удаляем существующий файл Matrix.exe
[InstallDelete]
Type: files; Name: "{app}\Matrix.exe"

;дополняем существующий INI-файл IniFile.ini новой секцией Settings, содержащей ключ Version со значением 1.1
[Ini]
Filename: “{win}\IniFile.ini”; Section: “Settings”; Key: “Version”; String: “1.1”

;устанавливаем новые файлы
[Files]
Source: “MatrixUp.exe”; DestDir: “{app}”
Source: “Plus.exe”; DestDir: “{app}”

;и небольшой код, в котором описывается константа MyConst – с ее помощью будет определен путь к инсталлятору, которым будет пользоваться директива AppModifyPath при изменении программы из Панели управления
[Code]
```
function MyConst(Default: String): String;
 Begin
  Result:=GetIniString(‘InstallSettings’,’InstallPath’,’{app}’,’IniFile.ini’);
 end;
```

Скрипт считывает из секции InstallSettings INI-файла IniFile.ini значение ключа InstallPath и присваивает его значению функции Result.

Секция [Ini] и удаление программ

Из секции **[Ini]** можно удалять только секции или параметры.

1) Для того, чтобы при удалении программы удалялся определенный ключ в заданной секции, используется флаг **uninsdeleteentry**.

[Ini]
Filename: "{app}\IniFile.ini"; Section: "InstallSettings"; Key: "InstallPath"; String: "{app}"; Flags: uninsdeleteentry
Filename: "{app}\IniFile.ini"; Section: "InstallSettings"; Key: "Installer"; String: "{src}"
Filename: "{app}\IniFile.ini"; Section: "InstallSettings"; Key: "InstallVers"; String: "1.0"

В данном примере при удалении программы из секции InstallSettings удалится ключ InstallPath со своим значением.

2) Для того, чтобы при удалении программы удалилась заданная секция со всеми ее ключами и их значениями, используется флаг **uninsdeletesection**.

[Ini]
Filename: "{app}\IniFile.ini"; Section: "InstallSettings"; Key: "InstallPath"; String: "{app}"; Flags: uninsdeletesection
Filename: "{app}\IniFile.ini"; Section: "InstallSettings"; Key: "Installer"; String: "{src}"
Filename: "{app}\IniFile.ini"; Section: "InstallSettings"; Key: "InstallVers"; String: "1.0"

В примере при удалении приложения секция InstallSettings удалится со всеми ключами: InstallPath, Installer и InstallVers и их значениями.

3) Для того, чтобы при удалении программы удалялась только пустая секция, существует флаг **uninsdeletesectionifempty**. Его можно комбинировать с флагом **uninsdeleteentry** – таким образом вы предотвратите нежелательное удаление ключей.

4) При удалении программы INI-файлы с помощью секции **[Ini]** полностью удалить нельзя – для этого существует секция **[UninstallDelete]**:

[UninstallDelete]
Type: files; Name: "{win}\IniFile.ini"

Если не надо менять существующее значение ключа

Допустим, что в указанной папке уже существует INI-файл, содержащий ключ с именем, заданным в секции **[Ini]**. Тогда при установке инсталлятор сравнит два ключа – уже существующий и тот, который он должен установить, и если они разные, то он заменит старое значение ключа новым. В ситуации, когда этого происходить не должно, применяется флаг **createkeyifdoesntexist** – с ним ключ с заданным именем будет создан только если до установки он не существовал.

[Ini]
Filename: "{app}\IniFile.ini"; Section: "InstallSettings"; Key: "InstallPath"; String: "{app}"
Filename: "{app}\IniFile.ini"; Section: "InstallSettings"; Key: "Installer"; String: "{src}"
Filename: "{app}\IniFile.ini"; Section: "InstallSettings"; Key: "InstallVers"; String: "1.0"; Flags: createkeyifdoesntexist

В данном примере в секции InstallSettings INI-файла IniFile.ini ключ InstallVers будет создан только в том случае, если до установки он не существовал. Если же ключ InstallVers уже существовал, инсталлятор оставит его как есть, и не будет менять его значение.

Секции [InstallDelete] и [UninstallDelete]

Эти секции отвечают за удаление файлов и/или папок перед установкой (**[InstallDelete]**) или при удалении (**[UninstallDelete]**) приложения.

Задача: перед установкой обновления для установленной программы удалить в папке приложения (путь к которой укажет пользователь при установке приложения) директорию и все ее содержимое (подпапки и файлы), и создать папку с тем же именем, куда установить все файлы обновления.

Решение:

```
……
;Перед установкой удалить в папке приложения папку Data со всем ее содержимым
[InstallDelete]
Type: filesandordirs; Name: "{app}\Data"
;Создать папку с тем же именем, что и удаленная (хотя создавать ее не обязательно – инсталлятор все равно ее создаст при выполнении секции [Files]
[Dirs]
Name: "{app}\Data"

;Установить в созданную папку файлы
[Files]
Source: "C:\Program1.exe"; DestDir: "{app}\Data"
Source: "C:\Program2.exe"; DestDir: "{app}\Data"
Source: "C:\ReadMe.txt"; DestDir: "{app}\Data"; Flags: isreadme

;При удалении приложения удалить созданную папку
[UninstallDelete]
Type: filesandordirs; Name: "{app}\Data"
……
```

Эту же задачу можно решить с помощью секции **[Code]**:

```
……
[Files]
Source: "C:\Program1.exe"; DestDir: "{app}\Data"; BeforeInstall: MyBeforeInstall
Source: "C:\Program2.exe"; DestDir: "{app}\Data";
Source: "C:\ReadMe.txt"; DestDir: "{app}\Data"; Flags: isreadme

[UninstallDelete]
Type: filesandordirs; Name: "{app}\Data"

[Code]
{Создать процедуру, которую инсталлятор вызовет перед установкой файла Program1.exe}
Procedure MyBeforeInstall;
Begin
{Если в директории программы существует папка Data, она удаляется со всеми вложенными в нее файлами и папками}
 if DirExists(ExpandConstant('{app}\Data')) then
  DelTree(ExpandConstant('{app}\Data'), True, True, True);
end;
……
```

Использование «масок»

Для того, чтобы не перечислять все однотипные файлы, которые следует удалить при установке или удалении программы, можно использовать «маску»:

[InstallDelete]
Type: files; Name: "{app}*.ini"

В этом примере перед установкой программы из ее папки удалятся все файлы с расширением **.ini**. Используйте «маски» для удаления файлов очень осторожно, особенно если ваше приложение устанавливается в папку Windows.

Задание «масок» для инсталлятора то же, что и для Windows: * - префикс (суффикс) любой длинны, ? – префикс (суффикс) из одного символа и т.д.

Если вы не уверены, что удаление файлов с помощью секций **[InstallDelete]** и **[UninstallDelete]** пройдет безболезненно для системы пользователя, лучше их вообще не использовать – пользователь вручную удалит ненужные файлы и папки.

Секции [Run] и [UninstallRun]

В секции **[Run]** перечисляются файлы, которые инсталлятор должен выполнить после удачной установки приложения (перед появлением страницы *Установка завершена*).

В секции **[UninstallRun]** перечисляются файлы, которые деинсталлятор должен выполнить перед удалением программы.

Секция [Run]: запустить программу после удачной установки

А) В любом случае

Два следующих примера приведут к одному и тому же результату: после удачной установки программы запустится файл Program.exe.

[Run]
FileName: "Program.exe"; WorkingDir: "{app}"; Flags: nowait

[Run]
FileName: "{app}\Program.exe"; Flags: nowait

Параметр WorkingDir задает папку, из которой будет запускаться файл. Если этот параметр не указан, следует задать полный путь к файлу в параметре FileName.

Для файлов, которые не являются исполняемыми (.exe или .com) или командными (.bat или .cmd), применяется флаг **shellexec** – в этом случае файл запустится с помощью программы, принятой по умолчанию для его типа (например, файл ReadMe.txt будет отображен на странице Блокнота). Флаг **shellexec** не определяет, выполнилась ли программа или нет, поэтому его следует комбинировать с флагами **nowait** (инсталлятор не будет ждать, пока запущенный файл выполнится, и сразу перейдет к следующему параметру секции **[Run]**) или **waituntilidle** (в этом случае установка будет приостановлена для того, чтобы пользователь смог ввести необходимые данные). Пример:

[Run]
FileName: "{app}\ReadMe.txt"; Flags: shellexec nowait

С помощью параметра StatusMsg можно задать сообщение, которое будет отображаться в окне инсталлятора во время выполнения файла. Если параметр не указан, будет отображаться стандартное сообщение.

[Run]
FileName: "{app}\Program.exe"; StatusMsg: "Выполнение программы Program.exe"; Flags: skipifsilent

Б) По желанию пользователя

Для того, чтобы пользователь мог выбрать, какие файлы он хочет запустить, а какие нет, используется флаг **postinstall**. Можно также указать комментарий к каждому из представленных на выполнение файлов с помощью параметра Description (если он не указан, инсталлятор будет использовать стандартный комментарий).

[Run]
Filename: "{app}\ReadMe.txt"; Description: "Прочитать файл ReadMe"; Flags: postinstall shellexec

Если для файла установлен флаг **unchecked**, флажок для него по умолчанию будет сброшен – по желанию пользователь сможет установить его.

Секция [UninstallRun]: запустить программу перед удалением

Так перед удалением приложения запустится программа Program.exe:

[UninstallRun]
FileName: "{win}\Program.exe"

Если некоторое приложение устанавливалось несколько раз (например, поверх него было установлено обновление), может оказаться, что файлы, предназначенные для запуска перед удалением, будут дублироваться. Чтобы этого не случилось, в секции **[UninstallRun]** задается параметр RunOnceId – в этом случае, встретив несколько одинаковых файлов, которые необходимо запустить перед удалением, деинсталлятор выполнит только последний из них, остальные же проигнорирует. Значение параметра RunOnceId может быть любым – главное, чтобы эти значения совпадали, например, для программы и ее обновления.

[UninstallRun]
FileName: “{win}\Program.exe”; RunOnceId: “DeleteProgram”

Флаги управления окнами

hidewizard – во время выполнения программы окно инсталлятора будет скрыто.

Runhidden – выполняемая программа (или файл) запустится в свернутом окне. Очевидно, не следует использовать этот флаг для программы, при выполнении которой потребуется участие пользователя (например, он должен будет ввести какие-либо данные).

runmaximized – программа (файл) запустится в развернутом окне.

runminimized – программа (файл) запустится в свернутом окне.

Специфичные для секций параметры и флаги

Секция [Run]:
Параметры Description, StatusMsg, флаги **postinstall**, **unchecked**, **skipifnotsilent** (указывает инсталлятору не запускать файл при обычной (не ускоренной) установке), **skipifsilent** (указывает инсталлятору не запускать файл при быстрой и очень быстрой установке) применимы только к секции **[Run]**.
Параметр RunOnceId применим только к секции **[UninstallRun]**.

Не выдавать сообщение об ошибке, если запускаемый файл не существует

Если предназначенный для выполнения файл может не существовать, и вы не хотите, чтобы инсталлятор/деинсталлятор выдавал при этом сообщения об ошибке, используйте для указанных в секциях **[Run]**/ **[UninstallRun]** файлов флаг **skipifdoesntexist**.

Секция [Registry]

Эта секция отвечает за создание, редактирование и удаление записей в реестре, которые сделает инсталлятор/деинсталлятор в процессе установки/удаления программы.

Основные параметры секции

Root – корневой ключ. Может иметь следующие значения:
HKCR (HKEY_CLASSES_ROOT)
HKCU (HKEY_CURRENT_USER)
HKLM (HKEY_LOCAL_MACHINE)
HKU (HKEY_USERS)
HKCC (HKEY_CURRENT_CONFIG)

Subkey – имя раздела (может включать константы).

ValueType – тип значения. Может быть следующим:
none – значение по умолчанию. Инсталлятор создаст пустой ключ без параметра; ValueName и ValueData при этом игнорируются.
string – инсталлятор создаст строковый параметр (REG_SZ).
expandsz – инсталлятор создаст расширенную строку (REG_EXPAND_SZ).
multisz – инсталлятор создаст многостроковый параметр (REG_MULTI_SZ).
dword – инсталлятор создаст целочисленный параметр (REG_DWORD).
binary – инсталлятор создаст двоичный параметр (REG_BINARY).

ValueName – имя создаваемого параметра (может включать константы).

ValueData – значение параметра реестра.
Если параметр ValueType равен string, expandsz или multisz, это будет строка, которая может содержать константы.
Если тип данных dword, это будет десятичное ("123"), шестнадцатеричное ("$7B"), или константа, представляющая собой целое число.
Если тип данных binary - последовательность шестнадцатеричных – десятичных байтов в форме: "00 ff 12 34".
Если тип данных none, параметр игнорируется.

Примеры создания ключей реестра

Так в корневом ключе HKEY_CURRENT_USER, разделе Software создастся пустой ключ Key (без параметров):

[Registry]
Root: HKCU; Subkey: "Software\Key"

Так мы создадим в корневом ключе HKEY_LOCAL_MACHINE, разделе Software, ключ Key, включающий строковый параметр с именем Name и значением Data, и дочерний ключ DochKey для ключа Key с именем Version и значением 1.0:

[Registry]
Root: HKLM; Subkey: "Software\Key"; ValueType: string; ValueName: "Name"; ValueData: "Data"
Root: HKLM; Subkey: "Software\Key\DochKey"; ValueType: string; ValueName: "Version"; ValueData: "1.0"

Дополнить существующее значение параметра реестра

Константа {olddata} позволяет дополнить уже существующее значение параметра реестра, если его ValueType равен string, expandsz, или multisz.

Например, пусть основное приложение создаст параметр Version со значением 1.0 так:

[Registry]
Root: HKCU; Subkey: "Software\Matrix"; ValueType: string; ValueName: "Version"; ValueData: "1.0"; Flags: createvalueifdoesntexist uninsdeletekey

а обновление для этого приложения дополнит его еще одним значением 1.1 так:

[Registry]
Root: HKCU; Subkey: "Software\Matrix"; ValueType: string; ValueName: "Version"; ValueData: "{olddata} 1.1"; Flags: uninsdeletekey

В итоге значение строкового параметра Version будет равно 1.0 1.1.

Если параметр не существовал до запуска инсталятора или его тип не строковый, {olddata} без предупреждения удаляется. Удалится константа и в случае, если создается параметр типа multisz, а существующий параметр не является многостроковым (т.е. является REG_SZ или REG_EXPAND_SZ).

Для вставки символа разрыва строки для параметра, имеющего тип multisz, используется константа {break}.

Замена ключей реестра и их значений

Для начала следует заметить, что если инсталлятор не обнаружит ключа или параметра, который ему следует удалить перед созданием новых ключей или параметров, он не выдаст сообщения об ошибке, а просто создаст новый ключ или параметр по заданным правилам, а удалять ничего не будет.

1) Флаг **deletekey** позволяет перед созданием нового ключа удалить уже существующий ключ со всеми его дочерними ключами, параметрами и значениями. В приведенном ниже примере перед созданием параметра ReCom в ключе Matrix, этот ключ будет полностью удален, потом инсталлятор создаст новый ключ с тем же именем (Matrix) и в нем – параметр ReCom.

[Registry]
Root: HKCU; Subkey: "Software\Matrix"; ValueType: dword; ValueName: "ReCom"; ValueData: "1"; Flags: deletekey uninsdeletekey

2) Флаг **deletevalue** позволяет перед созданием нового параметра в ключе удалить уже существующий параметр того же типа и с тем же именем. В примере перед созданием строкового параметра Version в ключе Matrix инсталлятор удалит уже существующий в реестре строковый параметр Version и его значение, а потом создаст новый параметр Version со значением 1.1.

[Registry]
Root: HKCU; Subkey: "Software\Matrix "; ValueType: string; ValueName: "Version"; ValueData: "1.1"; Flags: deletevalue uninsdeletekey

3) Флаг **preservestringtype** действует следующим образом: если тип существующего параметра реестра не string или expandsz (т.е. не REG_SZ или REG_EXPAND_SZ), то он будет заменен строковым параметром с заданным значением, а если тип существующего параметра string или expandsz, заменится только его значение, а тип останется.
Если приложение создаст в ключе Matrix реестра параметр Version, имеющий тип dword и значение 1:

[Registry]
Root: HKCU; Subkey: “Software\Matrix”; ValueType: dword; ValueName: “Version”; ValueData: “1”; Flags: uninsdeletekey

то обновление для этого приложения изменит у параметра Version тип с dword на string и значение с 1 на 1.0 следующим образом:

[Registry]
Root: HKCU; Subkey: “Software\Matrix”; ValueType: string; ValueName: “Version”; ValueData: “1.0”; Flags: preservestringtype uninsdeletekey

Если же приложение создаст в ключе Matrix реестра параметр Version, имеющий тип expandsz и значение 1.0.0:

[Registry]
Root: HKCU; Subkey: “Software\Matrix”; ValueType: expandsz; ValueName: “Version”; ValueData: “1.0.0”; Flags: uninsdeletekey

то обновление для этого приложения изменит у параметра Version только значение с 1.0.0 на 1.1.1, а тип expandsz параметра не изменит, таким образом:

[Registry]
Root: HKCU; Subkey: “Software\Matrix”; ValueType: string; ValueName: “Version”; ValueData: “1.1.1”; Flags: preservestringtype uninsdeletekey

Реестр и удаление приложения

1) С помощью флага **uninsclearvalue** при удалении приложения параметр в реестре установится в пустую строку (с типом REG_SZ). Флаг не комбинируется с флагом **uninsdeletekey**.
Так ключ Key и его параметр Version типа dword со значением 1 при удалении приложения будет заменен ключом Key с пустым параметром Version типа REG_SZ.

[Registry]
Root: HKCU; Subkey: "Software\Key"; ValueType: dword; ValueName: "Version"; ValueData: "1"; Flags: uninsclearvalue

2) Флаг **uninsdeletekey** при удалении приложения удаляет ключ реестра со всеми его дочерними ключами, параметрами и значениями. Так при удалении приложения удалится ключ Key и его параметр Version со значением 1.1.1:

[Registry]
Root: HKCU; Subkey: "Software\Key"; ValueType: string; ValueName: "Version"; ValueData: "1.1.1"; Flags: uninsdeletekey

3) При использовании флага **uninsdeletekeyifempty** при удалении приложения ключ реестра удалится, только если он пустой. В примере ключ Matrix при удалении приложения удалится, если останется пустым:

[Registry]
Root: HKLM; Subkey: "Software\Matrix"; Flags: uninsdeletekeyifempty

4) Когда установлен флаг **uninsdeletevalue**, при удалении приложения удалится параметр реестра и его значение. В примере при удалении программы из реестра удалится параметр Version со значением 1.0:

[Registry]
Root: HKCU; Subkey: "Software\Matrix"; ValueType: string; ValueName: "Version"; ValueData: "1.0"; Flags: uninsdeletevalue

Флаг **uninsdeletevalue** удобно использовать в комбинации с флагом **uninsdeletekeyifempty** – если все значения в ключе реестра будут удалены, эффект будет тот же, что и при использовании флага **uninsdeletekey**, а если нет, то ключ с не удаленными по какой-либо причине параметрами удален не будет.

Ключ (параметр) существует – заменить на данный, не существует – не создавать

В ситуациях, когда при установке уже существующий параметр в ключе реестра следует заменить новым, а если ключ или параметр не существует – не создавать ни ключа, ни параметра, используется флаг **dontcreatekey**. В примере если в ключе Key существует параметр Version любого типа и с любым значением, он заменится строковым параметром Version со значением 1.0, а если не существует, то инсталлятор не создаст параметра в реестре (остальные параметры и дочерние ключи в ключе Key останутся без изменения). Если при этом в реестре не существует сам ключ Key, инсталлятор его не создаст.

[Registry]
Root: HKCU; Subkey: “Software\Key”; ValueType: string; ValueName: “Version”; ValueData: “1.0”; Flags: dontcreatekey

Создать ключ (параметр) только если он не существует

При использовании флага **createvalueifdoesntexist** инсталлятор создаст в ключе реестра параметр с заданным значением только в том случае, если он не существует. Если инсталлятор при установке обнаружит, что параметр с заданным именем уже существует, изменять его он не будет. В примере если инсталлятор при установке обнаружит, что в ключе Matrix уже существует параметр с именем Version любого типа и с любым значением, он оставит его как есть, а если параметр с именем Version не существует, инсталлятор создаст строковый параметр Version со значением 1.0.0. Если при этом в реестре не существует сам ключ Matrix, инсталлятор создаст сначала его, а потом параметр Version в нем.

[Registry]
Root: HKCU; Subkey: "Software\Matrix"; ValueType: string; ValueName: "Version"; ValueData: "1.0.0"; Flags: createvalueifdoesntexist

Флаг **createvalueifdoesntexist** не действует, если задан флаг **deletevalue**.

Секции [Languages] и [LangOptions]

Для поддержки многоязыковых установок в Inno Setup существует секция **[Languages]**. В ней обязательно указывается идентификатор языка (параметр Name) и путь к файлу, из которого инсталлятору следует считывать стандартные сообщения (параметр MessagesFile).

Секция **[LangOptions]** служит для задания дополнительных языковых опций, таких как стиль и размер шрифта.
LanguageName – имя языка, которое будет отображено в диалоге *Выбор языка*. Во избежание неприятных последствий для имени русского языка лучше использовать строку символов юникод в шестнадцатеричном виде: LanguageName=<0420><0443><0441><0441><043A><0438><0439>.
LanguageID – цифровой идентификатор языка. Он используется для автоопределения большинства языков, используемых по умолчанию. Всегда должен начинаться со знака "$", если идентификатор шестнадцатеричный.
LanguageCodePage – кодовая страница для языка. Если этому параметру присвоить значение 0, язык всегда будет присутствовать в списке, вне зависимости от того, какая кодовая страница используется в системе.
DialogFontName и **DialogFontSize** – стиль и размер шрифта в диалогах.
WelcomeFontName и **WelcomeFontSize** – стиль и размер шрифта для страниц *Приветствие* и *Установка завершена*.
TitleFontName и **TitleFontSize** – стиль и размер шрифта для названия приложения в фоновом окне инсталлятора (если WindowVisible=yes).
CopyrightFontName и **CopyrightFontSize** – стиль и размер шрифта для значения директивы AppCopyright (если WindowVisible=yes).

Если перед параметром секции **[LangOptions]** не указан идентификатор языка, инсталлятор применит их ко всем языкам, заданным в секции **[Languages]** (кроме параметров **LanguageName** и

LanguageID). Для того, чтобы изменения затронули только конкретный язык, перед параметром следует указать префикс (идентификатор языка и точку):

rus.TitleFontSize=29

Как перевести интерфейс инсталлятора на другой язык?

Inno Setup поддерживает несколько языков установки. Файлы этих языков находятся в директории Inno Setup в папке **Languages** и имеют расширение .isl (inno setup language). Если вы хотите, чтобы ваш инсталлятор поддерживал язык, не включенный в доступные языки Inno Setup, скопируйте файл Default.isl из папки программы и переименуйте его. Не следует работать прямо с файлом Default.isl, т.к. при удалении программы Inno Setup он удалится вместе с остальными ее компонентами, и ваш труд будет утерян. Затем откройте этот файл с помощью Блокнота и переведите значения используемых Inno Setup стандартных сообщений на нужный вам язык.

Многоязыковая установка

Задача: инсталлятор должен поддерживать интерфейс установки на русском и немецком языке, отображать информацию перед установкой и после установки (на странице Мастера) и файл лицензии на выбранном пользователем языке.

Решение:

[Setup]

……

;При наличии более, чем одного языка в секции **[Languages]** в начале установки будет отображен диалог *Выбор языка*

ShowLanguageDialog=yes

;Если LanguageDetectionMethod=none, инсталятор будет использовать первый язык, указанный в секции **[Languages]**, как язык по умолчанию

LanguageDetectionMethod=none

……

[Languages]

;Указываем идентификатор языка и пути к файлам, из которых инсталлятору следует считывать стандартные сообщения, информацию до и после установки и лицензионное соглашение

Name: "rus"; MessagesFile: "C:\Russian.isl"; InfoBeforeFile: "C:\IBFRus.txt"; InfoAfterFile: "C:\IAFRus.txt"; LicenseFile: "C:\LicFRus.txt"

Name: "ger"; MessagesFile: "C:\German.isl"; InfoBeforeFile: "C:\IBFGer.txt"; InfoAfterFile: "C:\IAFGer.txt"; LicenseFile: "C:\LicFGer.txt"

[LangOptions]

;Параметры для русского языка

rus.LanguageName=<0420><0443><0441><0441><043A><0438><0439>

rus.LanguageID=$0419

rus.LanguageCodePage=1251

rus.DialogFontName= MS Shell Dlg

rus.DialogFontSize=8

rus.WelcomeFontName=Verdana

rus.WelcomeFontSize=12

rus.TitleFontName=Arial

rus.TitleFontSize=29

rus.CopyrightFontName=Arial

rus.CopyrightFontSize=8

;Параметры для немецкого языка

ger.LanguageName=Deutsch

ger.LanguageID=$0407

ger.LanguageCodePage=1251

ger.DialogFontName= MS Shell Dlg

ger.DialogFontSize=8
ger.WelcomeFontName=Verdana
ger.WelcomeFontSize=12
ger.TitleFontName=Arial
ger.TitleFontSize=29
ger.CopyrightFontName=Arial
ger.CopyrightFontSize=8

Секцию **[LangOptions]** лучше указывать, даже если в ней используются значения по умолчанию.

После того, как пользователь выберет язык установки в диалоге *Выбор языка*, его имя присвоится константе **{language}**.

Секция [Messages]

Секция **[Messages]** отвечает за стандартные сообщения, которые встречаются в ходе установки вашей программы. Полный их список можно найти, открыв с помощью Блокнота любой из файлов языка установки для Inno Setup (они находятся в папке Languages в директории Inno Setup). Каждое сообщение имеет уникальное имя. Если вы хотите изменить стандартное значение сообщения для любого из языков, присвойте сообщению новое значение в секции **[Messages]**.

Знак %n означает перевод каретки. Также в сообщения можно вставлять значения некоторых директив: [name] – значение AppName, [name/ver] – значение AppVerName.

Изменение стандартных сообщений

Задача: изменить сообщения на странице *Приветствие*.

Решение:

Стандартные сообщения страницы приветствия Мастера описываются параметрами WelcomeLabel1 и WelcomeLabel2.

[Setup]
AppName=Proga
AppVerName=Proga 1.0
DefaultDirName={pf}\Proga
DefaultGroupName=Proga
AllowNoIcons=yes
LicenseFile=C:\LicFRus.txt
Compression=lzma

[Messages]
WelcomeLabel1=Запущен мастер установки [name]
WelcomeLabel2=Сейчас на ваш компьютер будет установлена программа [name/ver].%n%nУбедительная просьба не курить и пристегнуть ремни.
……

Если ваш инсталлятор поддерживает многоязыковую установку, и вы хотите, чтобы стандартное сообщение поменялось только для одного из языков, перед именем параметра следует указать идентификатор языка:

……
[Messages]
rus.WelcomeLabel1=Запущен мастер установки [name]
rus.WelcomeLabel2=Сейчас на ваш компьютер будет установлена программа [name/ver].%n%nУбедительная просьба не курить и пристегнуть ремни.
……

Если идентификатор языка перед именем стандартного сообщения не указан, значение поменяется для всех языков установки.

Специальный идентификатор секции [Messages]

Для секции **[Messages]** существует специальный идентификатор BeveledLabel, с помощью которого можно задать текст, который будет отображаться в левом нижнем углу инсталлятора и деинсталлятора.

```
……
[Messages]
BeveledLabel=BeveledLabel=%nУстановка Proga 1.0
……
```

Секция [CustomMessages]

В секции **[CustomMessages]** можно задать значение для констант {cm:…}. Секция **[CustomMessages]** отличается от секции **[Messages]** тем, что работает с любыми заданными вами сообщениями, а не со стандартными, описанными в файле Default.isl и других файлах языков установки.

Задача: изменить стандартное сообщение UninstallProgram.
Решение:
Стандартное сообщение UninstallProgram для русского языка имеет значение «Деинсталляция». Таким образом можно изменить его на «Удалить»:

```
[Setup]
AppName=Proga
AppVerName=Proga 1.0
DefaultDirName={pf}\Proga
DefaultGroupName=Proga
AllowNoIcons=yes
Compression=lzma

[CustomMessages]
UninstallProgram=Удалить %1

[Files]
Source: "C:\MATRIX.EXE"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\ReadMe.txt"; DestDir: "{app}/Data"; Flags: ignoreversion

[Icons]
Name: "{group}\Proga"; Filename: "{app}\MATRIX.EXE"
Name: "{group}\{cm:UninstallProgram,Proga}"; Filename: "{uninstallexe}"
……
```

Полезные задачи

Сделать обновление на установленную программу

Предположим, что нам требуется не установить программу «с нуля», а сделать обновление (апдейт) на уже установленную программу. Для этого используется директива AppId. Если в скрипте она не указана, компилятор присваивает ей значение директивы AppName. В данной ситуации значение директивы AppId в программе и обновлении должно быть одинаковым. Рассмотрим конкретную задачу.

Задача 1: на компьютере пользователя УЖЕ УСТАНОВЛЕНА программа Program, в которую входят компоненты: prog1, prog2, prog3. В обновлении нужно добавить компонент prog4 и файл readme для него.

Решение:

Пусть обновление для программы Program называется ProgramUpd. Пишем простой скрипт для установки нашего апдейта. Не забудьте задать директиве AppId то же значение, что и в скрипте самой программы. Он будет выглядеть примерно так:

[Setup]
AppName=ProgramUpd
AppVerName= ProgramUpd ver.1.0
DefaultDirName={pf}\ ProgramUpd 1.0
DefaultGroupName= ProgramUpd 1.0
OutputBaseFilename= ProgramUpd Setup 1.0
OutputDir=c:\ProbaUpd\
SourceDir=C:\
AllowNoIcons=yes
AlwaysShowDirOnReadyPage=yes
AppPublisher=My Company, Inc.
AppPublisherURL=http://www.mycompany.com/
AppVersion=1.0
CreateUninstallRegKey=yes
DirExistsWarning=auto

[Files]
Source: "ProgramUpd.exe"; DestDir: "{app}"
Source: "ReadmeUpd.txt"; DestDir: "{app}"
[Tasks]
Name: quicklaunchicon; Description: "Поместить ярлык на рабочий стол"; GroupDescription: "Всякие вкусности:"; Flags: unchecked
[Icons]
Name: "{userdesktop}\ProgramUpd"; IconFilename:"{app}\Ikonka.ico"; Filename: "{app}\ProgramUpd.exe"; Tasks: quicklaunchicon
Name: "{group}\ProgramUpd"; IconFilename:"{app}\Ikonka.ico"; Filename: "{app}\ProgramUpd.exe"
Name: "{group}\Удалить ProgramUpd"; FileName: "{app}\unins001"
Name: "{group}\ReadmeUpd"; Filename: "{app}\ReadmeUpd.txt"

В этом случае мы просто добавили новый файлик в папку программы. Теперь усложним задачу.

Задача 2: та же, только файл prog1 программы Program нужно ЗАМЕНИТЬ файлом prog.

Решение:

В этом случае следует добавить секцию [InstallDelete] и в ней указать имя и путь к файлу, который перед установкой обновления следует удалить. Скрипт будет иметь такой вид:

[Setup]
AppName=ProgramUpd
AppVerName= ProgramUpd ver.1.0
DefaultDirName={pf}\ ProgramUpd 1.0
DefaultGroupName= ProgramUpd 1.0
OutputBaseFilename= ProgramUpd Setup 1.0
OutputDir=c:\ProbaUpd\
SourceDir=C:\
AllowNoIcons=yes
AlwaysShowDirOnReadyPage=yes

AppPublisher=My Company, Inc.
AppPublisherURL=http://www.mycompany.com/
AppVersion=1.0
CreateUninstallRegKey=yes
DirExistsWarning=auto

[InstallDelete]
Type: files; Name: “{app}\Program.exe”
[Files]
Source: "ProgramUpd.exe"; DestDir: "{app}"
Source: "ReadmeUpd.txt"; DestDir: "{app}"
[Tasks]
Name: quicklaunchicon; Description: "Поместить ярлык на рабочий стол"; GroupDescription: "Всякие вкусности:"; Flags: unchecked
[Icons]
Name: "{userdesktop}\ProgramUpd"; IconFilename:"{app}\Ikonka.ico"; Filename: "{app}\ProgramUpd.exe"; Tasks: quicklaunchicon
Name: "{group}\ProgramUpd"; IconFilename:"{app}\Ikonka.ico"; Filename: "{app}\ProgramUpd.exe"
Name: "{group}\Удалить ProgramUpd"; FileName: "{app}\unins001"
Name: "{group}\ReadmeUpd"; Filename: "{app}\ReadmeUpd.txt"

Усложним задачу еще больше.

Задача 3: пусть теперь нам надо перед установкой обновления проверить, установлена ли на компьютере пользователя исходная программа, и если она не установлена, то прекратить установку обновления и выдать сообщение: «Перед установкой апдейта ProgramUpd Вам следует установить программу Program!» Если же исходная программа установлена, обновление добавит в ее папку (которую укажет пользователь) один .exe-файл.

Решение:

Имеем две программы: исходную Proga и обновление для нее ProgaUpd. Программа Proga состоит из одного файла MATRIX.EXE. Обновление ProgaUpd, если найдет в реестре правильный ключ, внесет в папку установленной программы еще один файл MATRIXUP.EXE. При удалении программы после установки обновления удалятся оба файла: MATRIX.EXE и MATRIXUP.EXE.

Методов решения этой задачи можно найти множество. Я предлагаю следующий: добавить при установке программы ключ в реестр, а обновление пусть проверяет наличие этого ключа.

Пример скрипта для установки основной программы Proga будет следующий:

[Setup]
AppName=Proga
AppVerName=Proga 1.0
;Как уже говорилось, для обеих программ: основной и обновления, директива AppId должна быть одинаковой
AppId=Proga
DefaultDirName={pf}\Proga
DefaultGroupName=Proga
Compression=lzma

;В данном случае в реестр можно ввести пустой параметр. Также рекомендуется удалять параметр из реестра при удалении программы – за это отвечает флаг **uninsdeletekey**. Флаг **createvalueifdoesntexist** означает, что параметр будет создан только если он не существует.
[Registry]
Root: HKCU; Subkey: "Software\Matrix"; Flags: createvalueifdoesntexist uninsdeletekey

[Files]
Source: "C:\ MATRIX.EXE "; DestDir: "{app}"

Пример скрипта обновления для программы Proga, которое будет называться ProgaUpd:

[Setup]

```
AppName=ProgaUpd
AppVerName=Proga 1.1
;Как уже говорилось, для обеих программ: основной и обновления, директива AppId должна быть одинаковой
AppId=Proga
DefaultDirName={pf}\Proga
DefaultGroupName=Proga
Compression=lzma

[Files]
Source: "C:\ MATRIXUP.EXE "; DestDir: "{app}"

[Code]
 function InitializeSetup(): Boolean;
begin
 if RegKeyExists(HKCU, 'Software\Matrix') then
  begin
   Result := MsgBox('Поиск установленных компонентов:' #13#13 'Компоненты найдены. Продолжить установку?', mbConfirmation, MB_YESNO) = idYes;
    if Result = False then
    MsgBox('Поиск установленных компонентов:' #13#13 'Компоненты не найдены.Установка невозможна.', mbInformation, MB_OK);
  end
 else MsgBox('Поиск установленных компонентов:' #13#13 'Обновление не может быть установлено! Сначала следует установить программу! ', mbConfirmation, MB_OK);
end;
```

Разберем подробнее секцию **[Code]**. Функция **InitializeSetup()**, как известно, при значении False прерывает установку программы. Функцией **RegKeyExists**(HKCU, 'Software\Matrix') определяем существование ключа в реестре. Если ключ существует, то у пользователя будет спрошено, желает ли он продолжить установку. Если же ключ в реестре не найден, инсталлятор выдаст сообщение, что обновление не может быть установлено без самой программы, и установка прервется.

Переменная Result – это результат функции, который будет использоваться далее в скрипте.

Задача 4: обновление устанавливается только поверх заданной версии приложения, в иных случаях выдает ошибку.

Решение:

Имеем две программы: исходную Proga и обновление для нее ProgaUpd. Программа Proga состоит из одного файла MATRIX.EXE. Обновление ProgaUpd, если найдет в реестре правильный ключ, внесет в папку установленной программы еще один файл MATRIXUP.EXE. При удалении программы после установки обновления удалятся оба файла: MATRIX.EXE и MATRIXUP.EXE.

Скрипт для основной программы:

```
[Setup]
AppName=Proga
AppVerName=Proga 1.0
;Для обеих программ: основной и обновления, директива AppId должна быть одинаковой
AppId=Proga
DefaultDirName={pf}\Proga
DefaultGroupName=Proga
Compression=lzma

[Registry]
;В реестр вносится строковый параметр с именем Version и значением 1.0
Root: HKCU; Subkey: "Software\Matrix"; ValueType: string; ValueName: "Version"; ValueData: "1.0"; Flags: createvalueifdoesntexist uninsdeletekey

[Files]
```

```
Source: "C:\MATRIX.EXE"; DestDir: "{app}"; Flags: ignoreversion
```

Скрипт для обновления:

```
[Setup]
AppName=ProgaUpd
AppVerName=Proga 1.1
;Для обеих программ: основной и обновления, директива AppId должна быть одинаковой
AppId=Proga
DefaultDirName={pf}\Proga
DefaultGroupName=Proga
Compression=lzma

[Files]
Source: "C:\MATRIXUP.EXE"; DestDir: "{app}"

[Code]
function InitializeSetup(): Boolean;
var ResultStr: String;
 begin
  ;Считываем значение параметра Version из реестра и присваиваем его переменной ResultStr
  RegQueryStringValue(HKCU, 'Software\Matrix', 'Version', ResultStr);
   ;Если значение параметра Version равно 1.0, обновление устанавливается, если не равно или не существует – установка прекращается
  if ResultStr='1.0' then
    begin
     Result := MsgBox('Поиск установленных компонентов:' #13#13 'Компоненты найдены. Продолжить установку?', mbConfirmation, MB_YESNO) = idYes;
      if Result = False then
        MsgBox('Поиск установленных компонентов:' #13#13 'Установка прервана.', mbInformation, MB_OK);
    end
  else MsgBox('Поиск установленных компонентов:' #13#13 'Программа не может быть установлена!', mbConfirmation, MB_OK);
 end;
```

Проверка версии Windows.

Задача 1: программа Program может быть установлена только на NT-платформу Windows.
Решение:
В данной задаче сложность скрипта зависит только от вашей фантазии.

1) В таком скрипте инсталлятор «молча» установит вашу программу на NT-платформу, и просто не запустится, если система не на NT-платформе. Сухо и строго – не думаю, что пользователю это понравится.

```
[Code]
function InitializeSetup(): Boolean;
begin
Result:=UsingWinNT;
end;
```

2) Такой инсталлятор пользователю понравится больше: при запуске установки не на NT-платформе он выдаст сообщение, что установка невозможна.

```
[Setup]
AppName=Program
AppVerName=Program 1.0
AppPublisher=Zuka-Buka, Inc.
```

```
DefaultDirName={pf}\Program
DefaultGroupName=Program
AllowNoIcons=yes
OutputDir="C:\Install"
Compression=lzma

[Files]
Source: "C:\MATRIX.exe"; DestDir: "{app}"
Source: "C:\ReadMe.txt"; DestDir: "{app}\Data"

[Code]
function InitializeSetup(): Boolean;
 begin
  if UsingWinNT=True then Result:=True
  else
   begin
   MsgBox('Ïðîãðàììà Программа может быть установлена только в Windows NT', mbConfirmation,
mb_OK);
   Result:=False;
   end;
 end;
```

Помните, что чем «дружественнее» будет ваш инсталлятор, тем лучше.

Задача 2: программа может быть установлена только на Windows 2000 версии 5.0.2195.

Функция GetWindowsVersion возвращает номер версии Windows в виде целочисленной переменной (integer). Верхние 8 бит указывают главную версию, следующие 8 бит – подверсию, и последние 16 бит – номер build’а. Например, эта функция возвратит $05000893 для Windows 2000 версии 5.0.2195.

Так функция вернет только главную версию Windows: "GetWindowsVersion shr 24", так – подверсию: "(GetWindowsVersion shr 16) and $FF", так – номер build’а: "GetWindowsVersion and $FFFF".

```
[Setup]
AppName=Program
AppVerName=Program 1.0
AppPublisher=Zuka-Buka, Inc.
DefaultDirName={pf}\Program
DefaultGroupName=Program
AllowNoIcons=yes
OutputDir="C:\Install"
Compression=lzma

[Files]
Source: "C:\MATRIX.exe"; DestDir: "{app}"
Source: "C:\ReadMe.txt"; DestDir: "{app}\Data"

[Code]
function InitializeSetup(): Boolean;
 begin
   if GetWindowsVersion=$05000893 then Result:=True
  else
   begin
     MsgBox(Программа может быть установлена только в Windows 2000 версии 5.0.2195',
mbConfirmation, mb_OK);
     Result:=False;
   end;
 end;
```

Выбор устанавливаемых компонентов

Задача 1: имеется набор компонентов. По желанию пользователь может установить все компоненты или выбрать нужные ему из списка.

Решение:

Данная задача решается при помощи трех секций: **[Types], [Components]** и **[Files]**.

В секции **[Types]** перечисляются все типы установки с их описанием. В готовом инсталляторе будут отображаться только описания, поэтому их лучше делать на русском языке. Для данной задачи можно создать два типа, например **Полная установка** и **Выборочная установка**, причем не забудьте указать компилятору, какая именно из установок является выборочной флагом **iscustom**.

В секции **[Components]** перечисляются компоненты для установки с описанием и указанием типов установки, к которым они относятся.

В секции **[Files]** перечисляются файлы, которые собственно и будут инсталлированы. Обязательно укажите, к какому компоненту относится каждый файл.

Многоточия в начале и конце примера означают, что вверху должна находиться секция **[Setup]** с необходимыми параметрами, а внизу могут идти другие секции (например, **[Tasks]** или **[Icons]**).

Пример скрипта:

```
……
[Types]
Name: "polnaya"; Description: "Полная установка"
Name: "viborochnaya"; Description: "Выборочная установка"; Flags: iscustom
[Components]
Name: "Component1"; Description: "Описание: программа первая"; Types: polnaya viborochnaya
Name: " Component 2"; Description: "Описание: программа вторая"; Types: polnaya viborochnaya
Name: " Component 3"; Description: "Описание: программа третья"; Types: polnaya viborochnaya
[Files]
Source: "Program1.exe"; Components: Component1; DestDir: "{app}"
Source: " Program2.exe"; Components: Component2; DestDir: "{app}"
Source: " Program3.exe"; Components: Component3; DestDir: "{app}"
……
```

Имя папки приложения зависит от выбранного языка установки

Задача: присвоить папке приложения в зависимости от выбранного языка разные имена.

Решение (способ 1):

Имя действующего языка (в формате String) возвращает функция ActiveLanguage. Зададим константу MyConst, результатом которой в зависимости от значения функции ActiveLanguage присвоится значение ‘Papka’ или ‘Folder’.

Пусть если пользователь выберет установку на русском языке, папка приложения будет называться Papka, а если на немецком, то Folder.

```
[Setup]
AppName=Program
AppVerName=Program 1.0
AppPublisher=Zuka-Buka, Inc.
DefaultDirName={pf}\{code:MyConst}
DefaultGroupName=Program 1.0
UsePreviousAppDir=no
UsePreviousGroup=no
ShowLanguageDialog=yes
LanguageDetectionMethod=none
Compression=lzma

[Files]
Source: "C:\MATRIX.exe"; DestDir: "{app}"
Source: "C:\ReadMe.txt"; DestDir: "{app}\Data"
```

```
[Languages]
Name: "rus"; MessagesFile: "C:\Russian.isl"
Name: "ger"; MessagesFile: "C:\German.isl"

[LangOptions]
rus.LanguageName=<0420><0443><0441><0441><043A><0438><0439>
rus.LanguageID=$0419
rus.LanguageCodePage=1251
rus.DialogFontName= MS Shell Dlg
rus.DialogFontSize=8
rus.WelcomeFontName=Verdana
rus.WelcomeFontSize=12
rus.TitleFontName=Arial
rus.TitleFontSize=29
rus.CopyrightFontName=Arial
rus.CopyrightFontSize=8

ger.LanguageName=Deutsch
ger.LanguageID=$0407
ger.LanguageCodePage=1251
ger.DialogFontName= MS Shell Dlg
ger.DialogFontSize=8
ger.WelcomeFontName=Verdana
ger.WelcomeFontSize=12
ger.TitleFontName=Arial
ger.TitleFontSize=29
ger.CopyrightFontName=Arial
ger.CopyrightFontSize=8

[Code]
function MyConst(Default: String): String;
  begin
    if ActiveLanguage='ger' then Result:='Folder' else Result:='Papka';
  end;
```

Решение (способ 2):

Можно обойтись и без программирования, описав в секции **[CustomMessages]** значения для константы {cm:defdir}.

Пусть если пользователь выберет установку на русском языке, папка приложения будет называться Papka, а если на немецком, то Folder.

```
[Setup]
AppName=Program
AppVerName=Program 1.0
AppPublisher=Zuka-Buka, Inc.
DefaultDirName={pf}\{cm:defdir}
DefaultGroupName=Program 1.0
UsePreviousAppDir=no
UsePreviousGroup=no
AllowNoIcons=yes
ShowLanguageDialog=yes
LanguageDetectionMethod=none
Compression=lzma

[CustomMessages]
;Для русского языка константе defdir присвоится значение ‘Papka’, а для немецкого – ‘Folder’.
rus.defdir=Papka
ger.defdir=Folder
```

[Files]
Source: "C:\MATRIX.exe"; DestDir: "{app}"
Source: "C:\ReadMe.txt"; DestDir: "{app}\Data"

[Languages]
Name: "rus"; MessagesFile: "C:\Russian.isl"
Name: "ger"; MessagesFile: "C:\German.isl"

[LangOptions]
rus.LanguageName=<0420><0443><0441><0441><043A><0438><0439>
rus.LanguageID=$0419
rus.LanguageCodePage=1251
rus.DialogFontName= MS Shell Dlg
rus.DialogFontSize=8
rus.WelcomeFontName=Verdana
rus.WelcomeFontSize=12
rus.TitleFontName=Arial
rus.TitleFontSize=29
rus.CopyrightFontName=Arial
rus.CopyrightFontSize=8

ger.LanguageName=Deutsch
ger.LanguageID=$0407
ger.LanguageCodePage=1251
ger.DialogFontName= MS Shell Dlg
ger.DialogFontSize=8
ger.WelcomeFontName=Verdana
ger.WelcomeFontSize=12
ger.TitleFontName=Arial
ger.TitleFontSize=29
ger.CopyrightFontName=Arial
ger.CopyrightFontSize=8

Этот вариант решения был предложен **mr_eoi** на компьютерном форуме Ru-Board.

Не могу найти решение

Задача : **установить/удалить** компоненты программы из Панели управления Windows. Хотелось бы, чтобы при добавлении/удалении инсталлятор выдавал сообщение: «Вставьте диск в CD-Rom».

Решение:

Эта задача решается с помощью директивы **AppModifyPath** секции **[Setup]**.

Пусть теперь пользователь будет иметь возможность удалить ненужные ему компоненты программы и установить нужные. Для этого на странице *Установка и удаление программ* Панели управления достаточно будет нажать кнопку **Изменить**.

При старте установки в окне можно выбрать, что сделать с программой: удалить компоненты, установить компоненты, удалить все.

Задача : если в указанной пользователем директории приложения существует папка Data, ее удалить вместе с ее содержимым, создать папку с тем же именем, установить туда все файлы. Если папка не существует, прервать установку.

Решение:

[InstallDelete]
Type: filesandordirs; Name: "{app}\Data"

```
[Dirs]
Name: "{app}\Data"

[Files]
Source: "C:\MATRIX.EXE"; DestDir: "{app}\Data"; DestName: "Matritsa.exe"; BeforeInstall:
MyBeforeInstall
Source: "C:\ReadMe.txt"; DestDir: "{app}\Data"; Flags: isreadme

[UninstallDelete]
Type: filesandordirs; Name: "{app}\Data"

[Code]
Procedure MyBeforeInstall;
begin
 if DirExists(ExpandConstant('{app}\Data')) then
  DelTree(ExpandConstant('{app}\Data'), True, True, True);
end;
```

Не получается: если папка не существует, инсталлятор ее создает и продолжает установку, а не прерывает ее.

Задача: Работа с диалоговыми окнами, объектами COM… **(**рассмотреть пример из папки Inno Setup (CodeClasses.iss)).

Задание: ассоциировать файлы с устанавливаемой программой (секция **[Tasks]**). Похоже, что за это отвечает не только инсталлятор, а еще и в программе прописывается какой-то код.

Задача: выдать сообщение на странице Блокнота перед окном *Приветствие* Мастера.