КАК НАПИСАТЬ СВОЮ СОБСТВЕННУЮ ПРОГРАММУ ИНСТАЛЯЦИИ?

Этапы инсталляции

Запомните одно важное правило: инсталлировать программу можно с человеческих носителей (винчестеры, компакт-диски, ZIP-диски) и с дискет :) Если вы собираетесь написать инсталляцию с дискет, которая явно не поместиться на одну дискету, то у вас есть шанс хорошо провести время :)

Как вы знаете, Windows сбрасывает ненужную ей в данный момент информацию на диск. Это правильно, но это касается данных. Программы никогда на диск не сбрасываются, поскольку в Windows сегмент кода программы не может быть изменён. Когда Windows нужна память и ей под руку подворачивается ваша программа, она её просто выкидывает — и всё. Когда ваша программа снова становиться нужна, Windows снова загружает её из выполняемого файла.

Эта в высшей степени корректная техника перестаёт работать при инсталляции с дискет. Ваша программа, например, копирует четвёртую дискету и тут выясняется, что у неё (у программы) пропал кусок кода. Какие проблемы? — Windows пытается прочитать файл **a:\setup.exe** и естественно его не находит (на четвёртой-то дискете? откуда?).

Только не паникуйте! Эта проблема давно решена, иначе вы не могли бы установить на свой компьютер ни одной программы! Всё очень просто — программа инсталляции копирует себя и все необходимые файлы во временный каталог на жёсткий диск и перезапускает себя с жёсткого диска. Это и есть первый этап инсталляции. В зарубежных программах он обычно называется "Prepare to install". Ещё раз обратите внимание на то, что совсем не обязательно выполнять этот этап, если вы инсталлируетесь не с дискет, или если ваша инсталляция умещается на одну дискету.

На втором этапе программа инсталляции обычно показывает пользователю несколько страшных предупреждений; что-то типа "если вы не заплатите за эту программу, то сидеть вам в тюрьме три пожизненных срока". Я слышал, что некоторые пользователи со слабым сердецем даже умирали за компьютером от таких угроз :)

Реализация этого этапа до идиотизма тривиальна, поэтому мы и не будем на нём останавливаться подробно.

Следущий этап — третий. Здесь программа установки дотошно выспрашивает у пользователя кучу всяких важных данных: имя пользователя и его огранизацию, тип установки, куда будем ставить, как будет называться группа программ и так далее. На этом этапе нам встретятся некоторые технические трудности, но их несложно обойти.

Четвёртый этап — копирование. Конечно, это не очень сложно, но некоторые проблемы у нас всё-таки возникнут. Во-первых, надо проверить наличие свободного места на целевом диске. Во-вторых, надо удостовериться, что у нас есть доступ к нужному каталогу. В-третьих, надо проверять, нет ли уже такого файла... Вы ещё не передумали писать программу инсталляции?

Следующий, пятый, этап — настройка системного реестра (registry). Достаточно тривиальная процедура, правда, при инсталляции большого продукта, записывать

придётся очень много.

Предпоследний, шестой, этап, заключается в создании группы программ в меню "Пуск". Или, возможно, вы захотите вынести ярлык на рабочий стол.

Наконец, финальная часть включает демонстрацию нескольких файлов (например, readme), затем онлайновую регистрацию (подробно на ней я останавливаться не буду) и последнее сообщение "Инсталляция успешно завершена".

Теперь мы можем перейти к подробному рассмотрению этапов. Сейчас вы узнаете, как это делается :)

Копирование программы во временный каталог

Чтобы не смущать высокое собрание подробным рассмотрением алгоритма, приведу конкретный пример исходника.

```
program Setup;

uses
Windows,
SysUtils;

const
ReRunParameter = '/install_from_temp_directory';

var
TempPath: array [0..MAX_PATH] of Char;
SrcPath: String;

begin
if ParamStr(1) = ReRunParameter then
SrcPath := ParamStr(2)
else
if GetDriveType(PChar(ParamStr(0)[1] + ':\')) = DRIVE_REMOVABLE then
begin
// Если программа была запущена без ключа и с дискеты, то
// копируем её во временный каталог и перезапускам
// Текущее приложение завершаем.
GetTempPath(MAX_PATH, TempPath);
// Добавлям к пути временного каталога символ '\', если его там нет
if (StrLen(TempPath) > 0) and (TempPath[StrLen(TempPath)] <> '\') then
StrCat(TempPath, '\');
// Копируем файл через вызов функции CopyFile из WinAPI
CopyFile(PChar(ParamStr(0)), PChar(String(TempPath) + ExtractFileName(ParamStr(0))),
False);
// Запускаем файл с двумя параметрами
WinExec(PChar(String(TempPath) + ExtractFileName(ParamStr(0)) + ' ' +
ReRunParameter + ' ' + ExtractFilePath(ParamStr(0))), CmdShow);
```

```
Exit;
end
else
SrcPath := ExtractFilePath(ParamStr(0));
// Здесь начинается программа инсталляции
// Переменная SrcPath показывает нам, откуда надо копировать файлы
end.
```

Есть две грабли, на которые можно наступить в приведённом примере. Первые лежат в вызове функции **GetTempPath**. Если у вас нет переменных окружения **TMP** и **TEMP**, то временным каталогом станет *текущий каталог программы*, то есть, фактически, ваша дискета.

Вы можете проверять, не находится ли временный каталог на сменном диске (с помощью вызова **GetDriveType**), и, если находиться, считать временным каталогом **C:\TEMP** (если его нет — создайте самостоятельно).

Вторые грабли заключаются в том, что после завершения инсталляции программу из временного каталога желательно удалить, но сделать этого вы не сможете, поскольку программа в этот момент выполняется. Вспомните, что в Windows 95 и Windows NT выполняющуся программу удалять нельзя.

В общем случае, решения этой проблемы я не знаю. Собственно, поскольку файл останется во временном каталоге, он будет одним из первых кандидатов на удаление (если пользователь хоть когда-нибудь чистит свой временный каталог :) Тем не менее, есть один хитрый способ удаления этого файла, о котором я расскажу ниже, в параграфе о деинсталляции.

Примечание:

> Если для вас важен размер вашей инсталляции, вы можете взять только тот кусочек, который приведён выше, и сделать из него отдельную программу (которая будет очень небольшого объёма). Саму программу инсталляции вы предварительно сжимаете, а перед запуском распаковываете её во временный каталог (а не копируете, как это сделано здесь).
>
> Обратите внимание, что в этом случае программа должна распаковываться в любом случае, а не только если она запущена с дискеты.

Запугивание пользователя законами об авторских правах

Да, есть и такой этап. Если вы пишете программу, которая будет распространяться как **freeware**, вам всё равно придётся вывести небольшое окно и поставить пользователя в известность о том, что вы не отвечаете за все неприятности, которые могут с ним произойти во время использования вашей программы.

Как это делается? Если вы не знаете, как сделать диалоговое окно, то, по моему, вам ещё рано писать инсталляции. Если знаете, то выведите окно и поместите в нём нужный текст.

Примечание:

> Поместить несколько строк текста можно разными способами. Во-первых, вы можете обрабатывать событие **OnPaint** диалоговой формы и рисовать

многострочный текст на ней с помощью функции Windows API, которя называется **DrawText**. Во-вторых, вы можете вывести текст через компонент **TRichEdit**, предварительно установив у него свойство **Border** в **bsNone**, свойство **Color** — в **clBtnFace**,а свойство **Enabled** — в **False**. Запрещённый (disabled) компонент **TRichEdit** выглядит не так, как запрещённый компонент **TMemo**; и отличие заключается в том, что он не меняет цвет текста на серый (что нам, собственно, и нужно).

Зачем делать компонент запрещённым? Дело в том, что в этом случае он выглядит просто как многострочный **TLabel**, например, его нельзя редактировать, из него нельзя выделить текст, ему не передаётся фокус, если вы пробегаете по компонентам с помощью клавиши **TAB** — несомненно, это то, что нам нужно.

Как получить важные системные данные

На четвёртом этапе нам потребуются некоторые системные данные: имя пользователя и организация, путь, куда потребуется инсталлировать программу и некоторые другие. Сейчас мы разберёмся, как и откуда эти данные можно получить.

Имя пользователя и организация

Во время инсталляции, программы иногда запрашивают имя пользователя и его организацию. Возможно, для работы вашей программы эти данные не понадобятся, но если они вам нужны, вы должны их запросить.

Как правило, программа инсталляции берёт эти данные из Windows (поскольку при установке Windows пользователь их уже вводил) и просит всего лишь изменить их, если это необходимо.

Наш вопрос звучит так: где Windows хранит имя пользователя и организацию? Я, правду сказать, не знаю. Но, пробежавшись по реестру, я обнаружил всего лишь два подходящих места, содержащих эту информацию.

HKEY_LOCAL_MACHINE\Software\Microsoft NT\Windows\ CurrentVersion\
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\ CurrentVersion\
HKEY_CURRENT_USER\Software\Microsoft\Windows\ CurrentVersion\
RegisteredOwner = 'Имя'
RegisteredOrganization = 'Организация'

В доступной мне версии Windows 95, эти значения хранятся в ветке **HKEY_LOCAL_MACHINE**, а в Windows NT — **HKEY_CURRENT_USER** (в подветках **Windows** или **Windows NT**).

Поскольку в этом вопросе нет ясности :) я предлагаю проверять обе ветки. Версию операционной системы можно узнать с помощью функции **GetVersionEx.**

Куда копировать программу

Можно сформулировать наш вопрос и по другому: где находиться каталог **Program Files**? Некоторые инсталляции считают, что это **C:\Program Files**. В действительности, конечно, он может находиться на другом диске, поэтому мы попробуем поискать его по другому... в реестре.

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\ CurrentVersion\
ProgramFilesDir = 'D:\Program Files'
```

Можно воспользоваться функцией **SHGetSpecialFolderLocation** (это даже более корректно с точки зрения Microsoft). Пример использования этой функции вы обнаружите несколькими файлами позже.

Для изменения каталога вы можете вызывать функции **SelectDirectory** или **SHBrowseForFolder**. Можно также создать собственное окно диалога "Выбор каталога" с помощью компонента **DirectoryListBox**. Подробнее о выборе каталога мы поговорим позднее, когда будем рассматривать тонкости процесса инсталляции.

Сколько осталось свободного места на диске

Программа инсталляции перед копированием файлов обязана проверить, сколько на целевом диске осталось свободного дискового пространства. Это делается при помощью функции **GetDiskFreeSpace** (из модуля **Windows**) или функции **DiskFree** (из модуля **SysUtils**). Вторая функция — это надстройка Delphi над Win API (в смысле, она вызывает **GetDiskFreeSpace**), но у неё значительно меньше параметров.

Группы программ

Обычно программа инсталляции создаёт для новой программы новую группу. Как правило, когда вы вводите название группы, рядом присутствует список, в котром перечислены все существующие группы. Получить такой список можно двумя способами. Один из них — работа с DDE-сервером, который называется **Program Manager**. Этот способ мы подробно рассмотрим чуть позже. Второй способ не очень сложен и основан на том факте, что всё меню "Программы" находиться в одном из каталогов вашего диска. Все подменю являются на самом деле подкаталогами, а пукнты — обычными ссылками (файлами с расширением **.lnk**).

Путь к папке, содержащей меню "Программы", вы можете найти в реестре:

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\
CurrentVersion\Explorer\Shell Folders\
Programs = 'D:\WINNT\Profiles\mark\Главное меню\Программы'
```

Не очень сложно прочитать содержимое этого каталога с помощью функций **FindFirst/FindNext.** Ниже мы и об этом поговорим подробнее, поскольку чтение содержимое каталогов потребуется нам при написании универсальной процедуры копирования файлов.

Копирование

Нет другого такого процесса в программе инсталляции, который бы выглядел снаружи столь простым и был бы в реализации столь сложным, как *копирование*. Сложная фраза. В переводе на русский язык она означает, что сделать хорошую операцию копирования файлов нелегко :)

Почему?

Потому что операция копирования файлов должна быть идеальной.

Начнём с самого просто случая — копирования одного файла. Для того, чтобы скопировать один файл, вы можете вызвать функцию Windows, которая называется **CopyFile**:

---

function CopyFile(lpExistingFileName, lpNewFileName: PChar; bFailIfExists: BOOL): BOOL; **stdcall**;

Как видим, функции необходимо передать имена двух файлов: исходного и целевого. Третий параметр отвечает за то, как функция будет поступать, если целевой файл уже существует. Значение **True** говорит о том, что функция не будет копировать файл, значение **False** — о том, что целевой файл будет перезаписан.

Функция возвращает **True**, если операция копирования была успешно выполнена. Значение **False** подсказывает нам, что необходимо вызвать функцию **GetLastError** для того, чтобы узнать код произошедшей ошибки.

Эта функция работает очень надёжно, поскольку она является частью операционной системы и практически наверняка именно она тестировалась огромное количетво раз. Тем не менее, первое, на чём мы остановимся — что в этой функции нас не устраивает?

```
if CopyFile(PChar(SourcePath), PChar(TargetPath), False) then
// Выполнилась успешно.
else
// Ошибка. Код ошибки можно получить, вызвав GetLastError.
```

У этой функции один недостаток, но он способен перекрыть все её достоинства. *Мы не имеем доступа к процессу копирования*. Это означает, что мы не можем показывать индикатор процесса копирования и не можем прервать функцию **CopyFile**, если пользователь нажал кнопку "Отмена" или клавишу Escape.

Насколько это важно? На этот вопрос вы вольны отвечать самостоятельно. Если вам потребуется копировать большое количество маленьких файлов, то для вас этот недостаток неважен. Если вы собираетесь копировать файлы размером в несколько мегабайт, то у пользователя вашей программы могут возникнуть определённые проблемы.

В Windows NT 4.0 появилась новая функция — **CopyFileEx**, которая позволяет снять все появившиеся проблемы, но добавляет ещё одну — она не работает в Windows 95.

:)

В Delphi для копирования файлов, мы можем воспользоваться объектами класса **TFileStream**. Например, так:

---

procedure CopyFile(Source, Target: **String**);

```
var
SourceStream: TFileStream;
TargetStream: TFileStream;
begin
SourceStream := TFileStream(Source, fmOpenRead);
try
TargetStream := TFileStream(Target, fmCreate);
try
TargetStream.CopyFrom(SourceStream, 0);
FileSetDate(TargetStream.Handle, FileGetDate(SourceStream.Handle));
finally
TargetStream.Free;
end;
finally
SouceStream.Free;
end;
FileSetAttr(Target, FileGetAttr(Source));
end;
```

Обработку ошибок, как говорится в таких случаях, я оставляю читателю в качестве упражнения... :)

Немного поговорим о приведённой процедуре. Саму операцию копирования выполняет метод **CopyFrom**. Если второй параметр этого метода равен нулю, то копируется сразу весь файл, в ином случае копируется указанное количество байтов. Мы можем копировать файл блоками, со всеми вытекающими отсюда преимуществами:

```
. . .
const
BlockSize = 65536;
. . .
var
ElapsedSize: Integer;
CopySize: Integer;
. . .
ElapsedSize := SourceStream.Size - SourceStream.Position;
while ElapsedSize > 0 do
begin
if ElapsedSize < BlockSize then
CopySize := ElapsedSize
else
CopySize := BlockSize;
TargetSource.CopyFrom(SourceSize, CopySize);
ElapsedSize := SourceStream.Size - SourceStream.Position;
// Здесь какие-то действия, например, модификация индикатора процесса
end;
. . .
```

Замечательные функции **FileGetDate**, **FileSetDate**, **FileGetAttr**, **FileSetAttr** выполняют очень важную работу, про которую не надо забывать: копируют дату создания файла и его атрибуты. Сейчас мы не будем углубляться в особенности файловой системы NTFS, в

которой есть дата последней модификации файла и расширенные атрибуты, поскольку приведённого кода нам вполне для наших целей хватит.

Примечание:

> Корректное копирование предполагает, что у целевого файла обязательно устанавливается флаг Archive. Я не буду углуляться в то, зачем это делается, тем более, что в нашем случае этого делать не обязательно.
>
> Тем не менее, если вы хотите реализовать полноценную операцию копирования, не забудьте про эту маленькую особенность.

В следующей статье мы поговорим о сжатии...

Копирование (продолжение)

Существует такая штука, как сжатие данных. Хорошо, если вы инсталлируете свою программу с компакт-диска. Как правило, в такой ситуации вы не знаете, что ещё записать на эту бездонную болванку и несколько лишних мегабайт вам в данной ситуации совсем не помешают :)

Совершенно по другому обстоят дела с дискетами. На дискеты надо помещать сжатую информацию — в этом случае вся ваша программа возможно (только возможно) влезет всего лишь на тридцать четыре дискеты. Вот как делать сжатие — это вопрос.

Я расскажу вам о нескольких методах сжатия данных, которыми вы можете воспользоваться.

Метод номер раз — это сжатие файлов стандартными утилитами, разработанными фирмой Microsoft. Когда-то это была программа **compress.exe** (9554), сейчас — **cabarc.exe** (35703).

```
hInFile := LZOpenFile(PChar(SourcePath), ofInReOpenBuff, OF_READ);
hOutFile := LZOpenFile(PChar(TargetPath), ofOutReOpenBuff, OF_CREATE or OF_WRITE);
iLZError := LZCopy(hInFile, hOutFile);
if iLZError > 0 then
// Операция выполнилась успешно, скопировано iLZError байт
else
// Ошибка номер iLZError
LZClose(hOutFile);
LZClose(hInFile);
```

Метод номер два — сжатие файлов с помощью библиотеки ZLib, которая поставляется вместе с Delphi (она находится в каталоге **\Info\Extras\ZLib** оригинального диска с Delphi 3.0). Она предоставляет вам два класса, которые являются наследниками **TStream**. Вы можете воспользоваться кодом функции копирования при помощи **TFileStream** из предыдущей статьи для того, чтобы реализовать как сжатие, так и распаковку произвольного потока (в том числе и файла).

Ещё один метод — использование динамической библиотеки **unrar.dll** (27336), разработанной Евгением Рошалом. Существуют и другие библиотеки (даже компоненты),

вы можете свободно найти их, если будете достаточно долго шляться по Интернету.

Примечание:

Мне кажется, что менее всего размер вашей инсталляции увеличится, если вы используете методы, предлагаемые Microsoft; ровно потому, что они встроены в Windows и вам не надо записывать их на дискету.

Копирование нескольких файлов представляется достаточно простым, раз уж мы научились копировать один файл. Наиболее просто это делается, если ваши файлы поставляются в одном архиве (**.CAB** или **.RAR**). Сложным может показаться копирование файлов по маске (***.***) и копирование вложенных подкаталогов. Ниже приводится исходный текст процедуры, которая составляет список файлов в каталоге и всех вложенных подкаталогах.

```
procedure ReadTree(Path: String; Strings: TStrings);
procedure ReadFolder(Path: String; Strings: TStrings);
var
SearchRec: TSearchRec;
FindResult: Integer;
begin
FindResult := FindFirst(Path + '*.*', faAnyFile, SearchRec);
while FindResult = 0 do
begin
// Если найден подкаталог, рекурсивно читаем его содержимое
// Не забываем игнорировать подкаталоги '.' и '..'
with SearchRec do
if (Name <> '.') and (Name <> '..') then
begin
Strings.Add(Path + Name);
if (Attr and faDirectory <> 0) then
ReadFolder(Path + Name + '\', Strings);
end;
FindResult := FindNext(SearchRec);
end;
FindClose(SearchRec);
end;
begin
// Эта процедура заносит в Strings список файлов во всех вложенных папках
// каталога Path и сами эти папки
Strings.Clear;
if (Length(Path) > 0) and (Path[Length(Path)] <> '\') then
Path := Path + '\';
ReadFolder(Path, Strings);
end;
```

Отдельно стоит поговорить о тех файлах, которые могут использоваться сразу несколькими программами. Для этих файлов существует даже специальное название — *разделяемые* (поскольку несколько программ *делят* их между собой). Обычно они записываются в системный каталог Windows (для Windows 95 это как правило **\WINDOWS\SYSTEM**, для Windows NT — **\WINNT\SYSTEM32**). Если системный каталог доступен только для чтения, то эти файлы необходимо записывать в каталог

Windows (**\WINDOWS** и **\WINNT** соответственно), который всегда доступен для записи.

```
function GetSysDir: String;
var
szPath: array [0..MAX_PATH - 1] of Char;
I: Integer;
Stream: TStream;
begin
// Получаем системный каталог
GetSystemDirectory(szPath, MAX_PATH);
Result := StrPas(szPath);
// Добавляем обратный слеш в конец пути, если его там нет
if (Length(Result) > 0) and (Result[Length(Result)] <> '\') then
Result := Result + '\';
// Подбираем имя файла вида XXXXXXXX.TMP, где XXXXXXXX ---
// шестнадцатиричное число, который не существует в системном каталоге
I := 0;
while FileExists(Result + IntToHex(I, 8) + '.TMP') do
Inc(I);
try
// Создаём файл и удаляем его. Если всё нормально, то каталог доступен
// для записи.
Stream := TFileStream(Result + IntToHex(I, 8) + '.TMP', fmCreate);
Stream.Free;
DeleteFile(Result + IntToHex(I, 8) + '.TMP');
except
// Если создать файл не удалось, в качестве системного каталога будем
// использовать каталог Windows.
GetWindowsDirectory(szPath, MAX_PATH);
Result := StrPas(szPath);
if (Length(Result) > 0) and (Result[Length(Result)] <> '\') then
Result := Result + '\';
end;
end;
```

Если разделяемый файл уже существует в целевом каталоге, то необходимо сравнить версии, языки и др. характеристики двух файлов и на основании этого сравнения решать — копировать файл или не надо.

При копировании разделяемых файлов требуется уведомить Windows о том, что одним разделением стало больше. Это делается через реестр. При замещении файлов, которые в момент инсталляции используются Windows требуется определённая техника, поскольку перезаписать занятый файл нельзя. Эти вопросы в ближайшее время будут освещены в следующей статье. Поистине, копирование файлов — тема неисчерпаемая :)

Напоследок, исследуем вопрос о том, куда копировать файлы? В соответствии с рекомендациями Microsoft, каталог вашей программы должен иметь форму **<Program Files>\<Название вашей фирмы>\<Название продукта>**. Вы можете также включить в название каталога информацию о версии продукта, например **C:\Program Files\Borland\Delphi 3**, хотя возможен и вариант **C:\Program Files\Borland\Delphi\3** (реально фирма Borland использует первый вариант). Бывают и исключения, в частности

FAR Евгения Рошала ставится в **C:\Program Files\FAR**.

Разделяемые файлы следует копировать в системный каталог Windows, а если он защищён от записи — в каталог Windows.

А сейчас мы переходим к реестру.

Системный реестр

Системый реестр — это древовидная структура данных, в который вы можете хранить настройки своей программы. Помимо вас, в реестр складывают данные другие программы, в том числе и операционная система.

Реестр является не только важным этапом во время инсталляции, но и прекрасным источником самой различной информации, поэтому я остановлюсь на нём подробнее.

Итак, реестр состоит из нескольких крупных деревьев, каждое из которых имеет уникальное название. Нас будут интересовать только два из них: **HKEY_CURRENT_USER** и **HKEY_LOCAL_MACHINE**.

Дерево **HKEY_LOCAL_MACHINE** (часто сокращаемое до **HKLM**) может содержать настройки программы для текущей машины безотносительно к тому, кто за ней работает. В Windows NT пользователи, не обладающие правами администратора, не могут записывать данные в это дерево. Из этого следует один простой вывод — если программа имеет общепользовательские настройки, то устанавливать её на компьютер (и изменять эти настройки впоследствии) может только администратор.

HKEY_CURRENT_USER (**HKCU**) содержит настройки программы для текущего пользователя. Это дерево для каждого пользователя компьютера своё. Оно может храниться не только на локальной машине, но и на сервере, что позволяет пользователю на каждом компьютере сети иметь одни и те же настройки (обои, клавиши переключения раскладки и др.).

Для того, чтобы просматривать и редактировать реестр, вы можете запустить редактор реестра (эта программа, как и следовало ожидать, называется **regedit.exe** :) Если вы часто работаете с реестром, её лучше всего вынести в меню "Программы".

Деревья реестра состоят из *разделов* (keys, в редакторе реестра они выглядят, как папки). У каждого раздела могут пыть различные параметры (values). По крайней мере один параметр есть у каждого раздела — в редакторе реестра он называется "(По умолчанию)" (в английской версии, естественно, "(Default)").

Как работать с реестром, вы можете прочитать в документации (ключевое слово **TRegistry**). Я же всего лишь отмечу, что выбрать дерево вы можете, используя свойство **RootKey**; для чтения/записи данных, вам потребуется *открыть раздел* (метод **OpenKey**); чтобы данные сохранились, не забудьте *раздел закрыть* (метод **CloseKey**). Название параметра по умолчанию — пустая строка.

В Windows регламентированы правила записи параметров программы в реестр. Вы, конечно, можете помещать данные куда угодно, однако, знать эти правила вам не

помешает.

В соответствии с правилами, параметры программы должны находиться в \Software\<Название фирмы>\<Название программы>\<Версия> Реальное же расположение поддерева вашей программы зависит только от вас. Если вам не требуется навороченной схемы работы с версиями, вы можете отказаться от <Версии>. Если у вас нет собственной фирмы :), можно отказаться и от <Названия фирмы> (но документация жутко не рекомендует этого делать). Например, программы RAR и FAR Евгения Рошала прописывают свои параметры в ветках **HKCU\Software\WinRAR** и **HKCU\Software\FAR**.

В реестре хранится куча полезной информации. Ниже я немного расскажу вам об этом, но прежде хочу заметить, что большая часть данных Windows находится в ветках **HKLM\Software\Microsoft\Windows\CurrentVersion** и **HKCU\Software\Microsoft\Windows\CurrentVersion**, поэтому в дальнейшем, в целях сокращения, я вместо **Software\Microsoft\Windows\CurrentVersion** буду просто ставить три точки.

Для того, чтобы поместить свою программу на рабочий стол, или в меню **Автозагрузка**, или сохранить результаты работы в папке **Мои документы**, требуется знать, где расположены соответствующие каталоги. Всем этим хозяйством заведует **Проводник**, поэтому нужную информацию можно найти в нём.

Нам потребуется раздел **HKCU\...\Explorer\Shell Folders** (обратите внимание, что **Shell Folders** пишется через пробел). Этот раздел содержит такие параметры (и не только их):

Desktop	Рабочий стол (папка).
Favorites	Избранное (папка).
Personal	Мои документы (папка).
Programs	Меню **Программы** (папка).
Start Menu	Меню кнопки **Пуск** (папка).
Startup	Меню **Автозагрузка** (папка).
Templates	Шаблоны документов (папка).

Обычно, в эти каталоги помещаются не сами файлы, а ссылки на них (о ссылках мы поговорим чуть позже).

В NT существует ещё несколько папок, которые связаны с "меню для всех". Раздел, где их можно найти, практически такой же: **HKLM\...\Explorer\Shell Folders** (заметьте, что теперь он принадлежит ветке **HKLM**, а не **HKCU**).

Common Desktop	Рабочий стол (папка).
Common Programs	Меню **Программы** (папка).
Common Start Menu	Меню кнопки **Пуск** (папка).
Common Startup	Меню **Автозагрузка** (папка).

Обратите внимание на наличие пробела после слова **Common**.

Если вы хотите, чтобы ваша программа деинсталляции появилась в стандартном списке деинсталляторов (Панель управления/Установка и удаление программ/Закладка "Установка/Удаление"), вы можете прописать её в реестре. Создайте в ветке **HKLM\...\Uninstall** раздел с произвольным именем (чаще всего используют название программы) и добавьте к нему два строковых параметра: **DisplayName** (то, что будет показано в списке готовых к деинсталляции программ) и **UninstallString** (командная строка запуска деинсталлятора — можно использовать параметры).

В списке, на рисунке справа, вы видите, о каких названиях идёт речь.

Если вы инсталлируете на компьютер разделяемые файлы (которыми могут пользоваться два или более приложений), пропишите их в **HKLM\...\SharedDLLs**. Названием параметра служит имя разделяемого файла, а значением (целого типа) — количество ссылок на него. При инсталляции файла вы проверяете, существует ли он в реестре, и если да, просто увеличиваете количество ссылок на единицу. Если разделяемого файла в реестре нет, создайте его, и установите количество ссылок равным единице. При деинсталляции файла уменьшите количество ссылок на 1, и если оно стало равным нулю — удаляйте его.

Как видим, если файл уже есть на компьютере, копировать его не обязательно. Иногда случается, что у вас более новый файл, чем тот, который уже установлен, и в этом случае его *нужно* скопировать.

Напоследок рассмотрим ещё несколько интересных разделов реестра. В ветках **HKCU\...\Run**, **HKLM\...\Run**, **HKLM\...\RunOnce** и **HKLM\...\RunServiceOnce** можно прописывать программы, которые вы хотите запускать при включении компьютера или входе в систему. Названием параметра может служить произвольная строка, а значением (строковым) будет путь к программе, которую вы хотите запустить.

Раздел **Run** просто запускает указанную программу всякий раз, когда пользователь входит в систему (действует аналогично папке **Автозагрузка**). Раздел **RunOnce** работает так же, как и **Run**, однако после выполнения программы, её параметр из **RunOnce** удаляется — это приводит к тому, что программа запускается только один раз. Наконец, **RunServiceOnce** действует точно также, как и **RunOnce**, но выполняется при включении компьютера, то есть ещё до того, как пользователь вошёл в систему.

Излишне, наверное, говорить, что **HKCU\...\Run** выполняется только для одного конкретного пользователя, а **HKLM\...\Run** — для любого пользователя компьютера.

С помощью реестра вы можете сделать так, чтобы **Проводник** ассоциировал иконку с вашим типом файлов, и запускал вашу программу при двойном щелчке на файле.

Для этого нам придётся исследовать ещё одну ветку реестра — **HKEY_CLASSES_ROOT** (**HKCR**).

Примечание:

> Эта ветка появилась в реестре самой первой (если углябляться в историю) и служила для хранения данных о COM-объектах (о тех же OLE-серверах, например), доступных системе. Это было ещё в Windows 3.x, то есть очень давно :)

Сейчас в реестре появилось ещё несколько веток, и поэтому **HKEY_CLASSES_ROOT** потеряла своё привилегированное положение. В связи с тем, что она обслуживает не только типы файлов, но и COM, ситуация с реестром немного запутанная. Мы разберём возможности этой ветки на примере программы WinRAR Евгения Рошала.

[HKEY_CLASSES_ROOT\.rar] @="WinRAR" "Content Type"="application/x-compressed" [HKEY_CLASSES_ROOT\.rar\ShellNew] "Data"=hex:52,61,72,21,1a,07,00,cf, 90,73,00,00,0d,00,00,00,00,00,00,00 [HKEY_CLASSES_ROOT\WinRAR] @="WinRAR archive" [HKEY_CLASSES_ROOT\WinRAR\shell] [HKEY_CLASSES_ROOT\WinRAR\shell\open] [HKEY_CLASSES_ROOT\WinRAR\shell\open\command] @="C:\\Program Files\\WinRAR\\WinRAR.exe \"%1\"" [HKEY_CLASSES_ROOT\WinRAR\DefaultIcon] @="C:\\Program Files\\WinRAR\\WinRAR.exe,0"

Для начала необходимо создать раздел **HKCR\.ext** (здесь ext — расширение файлов, с которым работает ваша программа). Параметром по умолчанию этого раздела должна быть некая уникальная строка (в нашем примере это **WinRAR**). Вы можете использовать имя программы, имя программы плюс тип, имя программы и версию, имя своей любимой бабушки, и вообще, любое имя. Далее этот идентификатор я буду называть **ExtentionID**.

Дополнительный возможный (необязательный) параметр — это **Content type**.

Необязательный подраздел **HKEY_CLASSES_ROOT\.ext\ShellNew** используется для того, чтобы пользователь мог создавать в **Проводнике** файлы вашего типа (правая кнопка мыши/Создать/...).

Следующий раздел — **HKCR\ExtentionID**. Значением по умолчанию этого раздела будет название файлов данного типа, например, **WinRAR archive** или **Документ Microsoft Word**. Это название используется **Проводником** (в том числе, и в подменю создания файлов). В дальнейшем эту строку я буду называть **DocumentName**.

Добавьте раздел **HKCR\ExtentionID\shell\open\command** для того, чтобы пользователь мог открывать файлы вашего типа двойным щелчком. Поскольку такой файл зачастую содержит несколько иконок, в случае необходимости указывайте порядковый номер иконки (нумеруются, начиная с нуля).

И, немного о том, как создаются новые файлы. Выше я написал, что для создания нового файла используется раздел **HKEY_CLASSES_ROOT\.ext\ShellNew**. В этом разделе может находится один из трёх параметров: **FileName**, **Command** и **Data**. В **FileName** записывается имя файла, который должен находится в каталоге с шаблонами документов (там лежат пустые файлы зарегестрированных типов) — он будет скопирован в указанный каталог с новым именем **DocumentName.ext**.

Примечание:

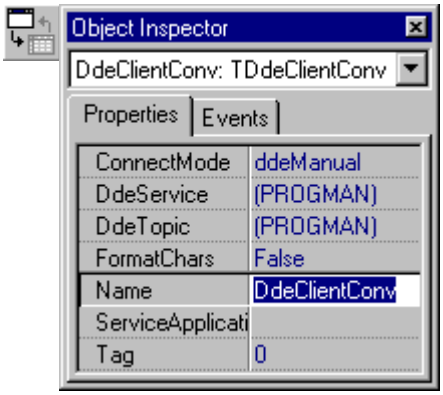
> Каталог с шаблонами документов (см. таблицу выше), можно найти в разделах **Shell Folders**/**User Shell Folders**, параметр **Templates**.

Command задаёт имя программы, которая должна вызываться для создания файла. Этой программе может быть передан параметр — имя нового файла (**DocumentName.ext**).

Если пустой файл указанного типа очень прост, и состоит из нескольких байт, мы можем записать эти байты в реестр, в параметр **Data**. Проводник создаст новый файл

DocumentName.ext, и скопирует эти данные в него.

Создание группы программ



Менеджер программ (Program Manager) знаком многим из тех, кто работал в Windows ещё в незапамятные времена. Сейчас, когда в Windows появился **Проводник**, менеджер программ ушёл на второй план и исользуется сейчас, в основном, с одной-единственной целью - создавать группы программ во время инсталляции. :)

Менеджер программ может быть сервером DDE-соединения. Если вы ничего не знаете о DDE - не беспокойтесь: всё, что нам потребуется, вы прочитаете в этой статье ниже.

Для работы с DDE-серверами мы будем использовать компонент **DdeClientConv**, который находится на закладке **System** панели компонентов.

Что такое DDE? DDE (Dynamic Data Excahnge) - это устаревшая технология, которая использовалась давным давно для передачи данных между программами (сейчас для этих целей используется OLE). Передача данных в DDE производится через *соединения*. *Программа-клиент* устанавливает соединение с *программой-сервером*, передаёт/принимает данные и разрывает соединение.

Применительно к нашей задаче, менеджер программ будет выступать в роли DDE-сервера, а программа инсталляции - в роли DDE-клиента. DDE предоставляет двухуровневую схему связи с сервером: для её установления клиенту необходимо указать *имя сервера* и *имя сервиса* (topic), который клиент хочет у сервера запросить. Для больших и толстых серверов, имя сервиса является насущной необходимость, для маленьких и простых - непонятным параметром, необходимость которого очень трудно объяснить.

Менеджер программ - простая и маленькая программа, поэтому имя сервиса у неё совпадает с именем сервера. Поместим на форму компонент **DdeClientConv** и настроим его в инспекторе объектов.

Свойства **DDEService** и **DDETopic** должны быть установлены в **PROGMAN**. Свойство **ConnectionMode** переключите в **ddeManual** -устанавливать и разрывать соединение мы будем вручную. Почему? Одним из правил, которых я придерживаюсь, является правило *не кушать ресурсы без необходимости*. Я создаю объекты в программах сразу перед тем, как они мне понадобятся; выделяю память непосредственно перед операторами, её использующими; и открываю файл, только тогда, когда уже готов его читать. Зачастую. :) Именно поэтому открывать соединение с менеджером программ я предлагаю только тогда, когда вся остальная работа уже выполнена.

Хорошо. Будем считать, это было лирическое отступление и вернёмся к нашим баранам. Может оказаться, что у нас под рукой нет подходящей формы для того, чтобы поместить на неё **DdeClientConv**. В этом случае можно создавать компонент **DdeClientConv** вручную: **var**
DdeClientConv: TDdeClientConv;

```
begin
DdeClientConv := TDdeClientConv.Create(nil);
with DdeClientConv do
begin
ConnectMode := ddeManual;
DDEService := 'PROGMAN';
DDETopic := 'PROGMAN';
. . .
Free;
end;
end;
```

Перед тем, как что-то сделать с сервером, вы должны вызвать метод **OpenLink** (установить соединение), а после работы - **CloseLink** (соответственно, разорвать).

Помимо приёма и передачи данных, DDE-сервер может *выполнять команыды клиента*. Всё очень просто: мы передаём серверу строку, говорим, что это команда - и он её исполняет.

Методы **ExecuteMacro** или **ExecuteMacroLines** компонента **DdeClientConv** передают серверу команду (и команды, соответственно) для выполнения. Команды менеджера программ заключаются в квадратные скобки, и выглядят приблизительно так: [CreateGroup("SomeTools",0)] [ShowGroup("Стандартные")] [AddItem("c:\windows\notepad.exe")]

Эти команды мы подробнее рассмотрим ниже, а перед этим остановимся ещё на одном важном моменте работы с менеджером программ - получении информации о текущем состоянии дел. Что нас может интересовать? Какие группы уже созданы, какие элементы находятся в этих группах...

Обратите внимание, что этой информацией мы можем и пренебречь - для того, чтобы создать группу, нам вовсе не требуется знать, какие группы уже созданы.

Пример показывает, как можно прочитать список групп из менеджера программ:

```
var
GroupList: TStringList;
begin
GroupList := TStringList.Create;
. . .
GroupList.Text := String(DdeClientConv.RequestData('Groups'));
. . .
GroupList.Free;
end;
```

После выполнения этой строчки, в **GroupList** появится список доступных групп.

Примечание:

> Если вы работаете под NT, возможно, вас волнует вопрос, как прочитать список групп из общедоступной части меню **Программы**. Для этого необходимо воспользоваться чем-то, что не является менеджером программ. Как это сделать, я расскажу чуть ниже.

Прочитав список групп, мы можем прочитать содержимое любой группы из этого списка

```
GroupList: TStringList;
ItemList: TStringList;
begin
GroupList := TStringList.Create;
ItemList := TStringList.Create;
. . .
GroupList.Text := String(DdeClientConv.RequestData('Groups'));
ItemList.Text := String(DdeClientConv.RequestData(PChar(GroupList[0])));
. . .
ItemList.Free;
GroupList.Free;
end;
```

Менеджер программ возвращает в первой строке название группы (в кавычках), имя файла группы (с расширением **.GRP**) и количество элементов в группе. Всё это разделяется запятыми. Далее идёт информация об элементах: командная строка (в кавычках), каталог по умолчанию, иконка, позиция в группе (по горизонтали и вертикали), номер иконки, клавиши быстрого доступа (в целочисленном виде) и флаг "свернуть при запуске".

Вот содержимое группы **Стандартные**: "Стандартные",C:\WINDOWS\ГЛАВНО~1\ ПРОГРА~1\СТАНДА~1,7,1
"Таблица символов","C:\WINDOWS\CHARMAP.EXE",,
C:\WINDOWS\CHARMAP.EXE,416,32,0,0,0
"Текстовый редактор WordPad","C:\PROGRA~1\ACCESS~1\WORDPAD.EXE",,
C:\PROGRA~1\ACCESS~1\WORDPAD.EXE,352,32,0,0,0
"Просмотр рисунков","C:\WINDOWS\WANGIMG.EXE",,
C:\WINDOWS\WANGIMG.EXE,288,32,0,0,0
"Калькулятор","C:\WINDOWS\CALC.EXE",, C:\WINDOWS\CALC.EXE,224,32,0,0,0
"Графический редактор Paint","C:\PROGRA~1\ ACCESS~1\MSPAINT.EXE",,
C:\PROGRA~1\ACCESS~1\MSPAINT.EXE,160,32,0,0,0
"Интерактивная регистрация","C:\WINDOWS\ SYSTEM\REGWIZ.EXE /i
software\microsoft\windows\currentversion",,
C:\WINDOWS\SYSTEM\REGWIZ.EXE,96,32,0,0,0
"Блокнот","C:\WINDOWS\NOTEPAD.EXE",, C:\WINDOWS\NOTEPAD.EXE,32,32,0,0,0

По большому счёту, нам эта информация не потребуется. Но, может быть, вы захотите заменить один из элементов группы во время инсталляции - в этом случае, конечно, надо знать, что изменять.

Далее я опишу способ создать группу и элемент в группе.

```
var
Commands: TStringList;
begin
. . .
Commands := TStringList.Create;
Commands.Add('[CreateGroup("Нестандартные",0)]');
Commands.Add('[ShowGroup("Нестандартные",1,0)]');
Commands.Add('[AddItem("c:\windows\notepad.exe","Блокнот")]');
DdeClientConv.ExecuteMacroLines(Commands, False);
Commands.Free;
. . .
end;
```

В приведённом примере мы создаём группу **Нестандартные** (в пику группе

Стандартные). Далее, мы показываем её (без этого добавление элемента не будет работать - ох, как я с этим намучился в своё время!). Наконец, добавляем Блокнот в эту группу, присваивая ему имя "Блокнот" (оригинально, не правда ли?). Важным здесь является то, что, во-первых, команды менеджеру программ заключаются в квадратные скобки, и, во-вторых, параметры команды, содержащие пробелы и прочие нестандартные символы, заключаются в кавычки. Ну, и, если вы вызываете несколько команд сразу, пользуйтесь методом **ExecuteMacroLines** вместо нескольких вызовов **ExecuteMacro**.

Полный список команд плюс описание вы найдёте в файле **win32.hlp**. Здесь я приведу только их список, чтобы вы знали, что искать: CreateGroup, ShowGroup, DeleteGroup, Reload, AddItem, ReplaceItem, DeleteItem и ExitProgman.

Я написал утилиту, которая позволит вам поиграться с менеджером программ. Для того, чтобы создать, показать или удалить группу программ, введите название группы в поле ввода **Группа** (или выберите её в списке) и нажмите соответствующую кнопку.

Та же самая техника работает и для элементов группы - если вы хотите добавить новый элемент, введите название программы и имя файла в соответствующих полях ввода перед тем, как нажать кнопку **Создать.**

Может быть это не самая сложная программа из тех, которые я написал, но она (а) доступна в исходниках и (б) позволяет "почувствовать", как работает менеджер программ.

Теперь перейдём ко второму способу работать с группами программ - через интерфейсы, предоставляемые **Проводником.**

Создание группы программ (продолжение)

Теперь мы подробнее остановимся на создании группы программ средствами **Проводника**. Дело в том, что меню, которое вы видите, нажав на кнопку **Пуск**, хранится на диске в виде обычных каталогов и файлов.

Для того, чтобы создать группу, мы должны всего лишь создать каталог; для того, чтобы создать элемент — скопировать файл в каталог или создать *ярлык* на существующую программу. Что такое ярлык? Ярлык — это файл с расширением **.LNK**, в котором хранится информация о каком-то другом файле. Для операционной системы ярлык олицетворяет собой файл, на который он ссылается.

Для чего применяются ярлыки? Для того, чтобы вам не нужно было хранить на своём диске несколько одинаковых файлов. Например, программа WinZip во время инсталляции помещает себя на рабочий стол, в меню **Пуск** и в меню **Программы**. Что, вы думаете, файл **winzip.exe** хранится на диске в трёх экземплярах? Отнюдь.

Учтите, что ярлык занимает на диске где-то 250–400 байт, поэтому для файлов меньшего размера ярлыки создавать бессмысленно.

С этого момента я объявляю ярлыки новой парадигмой создания групп! :) Звучит громко... Нам осталось только выяснить, как это делается. :) Итак, группы менеджера программ доступны нам через кнопку **Пуск** подменю **Программы**. В действительности, все они являются подкаталогами, одного из каталогов Windows (это может быть, например, **C:\Windows\Главное меню\Программы**. Путь к этому каталогу **Проводник** хранит в

реестре.

```
procedure ReadGroups(Strings: TStrings);
var
ARegistry: TRegistry;
Programs: String;
SearchRec: TSearchRec;
FindResult: Integer;
begin
Strings.Clear;
// Находим каталог
ARegistry := TRegistry.Create;
with ARegistry do
begin
RootKey := HKEY_CURRENT_USER;
if OpenKey('\Software\Microsoft\Windows\CurrentVersion\Explorer\ Shell Folders', False) then
begin
Programs := ReadString('Programs');
CloseKey;
end
else
Programs := '';
Free;
end;
if (Length(Programs) > 0) and (Programs[Length(Programs)] <> '\') then
Programs := Programs + '\';
// Читаем содержимое каталога
FindResult := FindFirst(Programs + '*.*', faDirectory, SearchRec);
while FindResult = 0 do
begin
with SearchRec do
if (Name <> '.') and (Name <> '..') and (Attr and faDirectory <> 0) then
Strings.Add(Name);
FindResult := FindNext(SearchRec);
end;
FindClose(SearchRec);
end;
```

Точно таким же макаром можно прочитать содержимое любой группы. Достаточно очевидно делаются такие вещи, как создать группу, удалить группу, переименовать (!) группу и так далее. Более того, теперь вы можете создавать группы в меню **Пуск** и на рабочем столе (в статье о системном реестре рассказано как найти путь к этим каталогам). Мы так же можем воспользоваться функцией **SHGetSpecialFolderLocation** для того, чтобы выяснить, где находится тот или иной каталог **Проводника**.

```
function GetSpecialFolderLocation(nFolder: Integer): String;
var
ppidl: PItemIDList;
Malloc: IMalloc;
```

```
szPath: array[0..MAX_PATH - 1] of Char;
begin
SHGetSpecialFolderLocation(Handle, nFolder, ppidl);
SHGetMalloc(Malloc);
SHGetPathFromIDList(ppidl, szPath);
Malloc.Free(ppidl);
Malloc := nil;
Result := String(szPath);
end;
```

См. описание функции **SHGetSpecialFolderLocation** для того, чтобы узнать, какие константы можно использовать в качестве **nFolder**.

Что нам осталось узнать? Как получить информацию о ярлыке и как создать ярлык. Обе эти операции удобно делать с помощью *интерфейса* **IShellLink**, предоставляемого нам **Проводником**. Об интерфейсах (и связанных с ними COM-объектах) говорить можно долго, однако, для наших целей это не нужно. Нам достаточно знать, что интерфейсы очень напоминают обычне классы, за несколькими исключениями. Вот, например:

```
const
MAX_DESCRIPTION = 100;
var
Bitmap: TBitmap;
ShellLink: IShellLink;
Description: array[0..MAX_DESCRIPTION - 1] of Char;
begin
// Создаём обычный объект
Bitmap := TBitamp.Create;
// Создаём COM-объект
CoCreateInstance(CLSID_ShellLink, nil, CLSCTX_INPROC_SERVER, IID_IShellLinkA,
ShellLink);
. . .
// Используем обычный объект
Bitmap.Width := 100;
// Используем COM-объект
ShellLink.GetDescription(Description, MAX_DESCRIPTION);
. . .
// Удаляем обычный объект
Bitmap.Free;
// Удаляем COM-объект
ShellLink := nil;
end;
```

Итак, для того, чтобы создать COM-объект в Дельфи, вы должны вызвать функцию **CoCreateInstance**, передав её в качестве параметров несколько странного вида констант. COM-объекты автоматически удаляются по завершении процедуры, в которой они были созданы. Тем не менее, я предпочитаю явное указание того факта, что COM-объект должен быть уничтожен:

```
ShellLink := nil;
```

Теперь приведу пример, показывающий, как можно создать ярлык на рабочем столе.

```
var
Desktop: String;
ShellLink: IShellLink;
hRes: HRESULT;
PersistFile: IPersistFile;
begin
// Находим каталог
Desktop := GetSpecialFolderLocation(CSIDL_DESKTOPDIRECTORY);
if (Length(Desktop) > 0) and (Desktop[Length(Desktop)] <> '\') then
Desktop := Desktop + '\';
CoInitialize(nil);
hRes := CoCreateInstance(CLSID_ShellLink, nil, CLSCTX_INPROC_SERVER,
IID_IShellLinkA, ShellLink);
if SUCCEEDED(hRes) then
begin
ShellLink.SetPath(PChar(Application.ExeName));
ShellLink.SetDescription('Вот моя программа!');
hRes := ShellLink.QueryInterface(IID_IPersistFile, PersistFile);
if SUCCEEDED(hRes) then
begin
PersistFile.Save(PWideChar(WideString(Desktop + 'Моя программа.lnk')), True);
PersistFile := nil;
end;
ShellLink := nil;
end;
CoUninitialize;
end;
```

Подробнее узнать о том, что можно делать с помощью интерфейса **IShellLink** вы можете в **win32.hlp** (ключевое слово — **IShellLink** :) Вызовы функций **CoInitialize** и **CoUninitialize** обеспечивают работу COM, всегда вызывайте первую из них перед работой, а вторую — после работы.

Для работы этих примеров вам потребуется включить (uses) в свой модуль файлы **Windows**, **ShlObj**, **ActiveX**, **ShellApi**. По недосмотру, или по какой-то другой причине, Инпрайз не внесла константу **IID_IPersistFile** ни в один из этих файлов. Эту константу можно найти в модуле **Ole2**, однако этот модуль оставлен в 3-ей и 4-ой версиях Дельфи только в целях совместимости со 2-ой версией, где COM-интерфейсы были реализованы по другому. Я сам уже запутался, честно говоря, объясняя, что здеь к чему :) Если в двух словах, скопируйте эти строчки из модуля **Ole2.pas** в свою программу и расслабтесь:

```
const
IID_IPersistFile: TGUID = (
D1:$0000010B;D2:$0000;D3:$0000;D4:
($C0,$00,$00,$00,$00,$00,$00,$46));
```

Напоследок остановимся на том, какой же метод создания групп лучше? Давайте

рассмотрим достоинства и недостатки этих методов.

Менеджер программ достался нам в наследство от 16-битных версий Windows. Он капризен и работает неторопливо. Помимо всего прочего, с помощью него нельзя помещать ярлыки на рабочий стол и в меню **Пуск.**

С другой стороны, интерфейсы проводника более современны, дают больший контроль за происходящим, работают быстрее... И помимо прочего, заставляют программиста осваивать новую технологии (а если программист ленив, ему такие стимулы необходимы, как воздух :)

И тем не менее — для создания групп в меню **Программы** я рекомендую вам пользоваться менеджером программ.

Почему?

Потому что менеджер программ, помимо того, что создаёт все необходимые каталоги и ярлыки, отражает изменения в файлах с расширением **.GRP**. Может оказаться, что некоторые программы (старые) работают с файлами групп, и, следовательно, некорректно будут работать с вашими ярлыками.

Конечно, ситуация, о которой я вам рассказываю, гипотетическая. Я ещё не разу не слышал о каких-бы то ни было действительных нареканиях для способа с **IShellLink**. В любом случае, решайте сами.

Примечание

> Всё, о чём я говорил выше, относится только к группам, ярлыки на рабочем столе или в меню **Пуск** в любом случае придётся создавать с помощью **IShellLink.**

Деинсталляция

Итак, поговорим немного о деинсталляции.

В идеале, деинсталляция должна привести компьютер к виду, в котором он был до инсталляции. На практике, это возможно не для всех приложений, особенно, если они разделяют ресурсы с другими программами.

Программа инсталляции должна вести *журнал инсталляции,* в который должны быть занесены все действия, производимые этой программой: создание разделов в реестре, секций в **.INI**-файлах, копирование, переименование, регистрация ActiveX-компонентов и многое другое.

Программа деинсталляции может, основываясь на этом журнале, произвести деинсталляцию продукта.

Но это всё вещи очевидные, как и многое другое в деинсталляции. Мы можем для каждого действия программы инсталляции указать действие, которое должно происходить при деинсталляции.

Создать каталог **Directory**.	Удалить каталог **Directory**.

Копировать файл **Sourcr** в **Target.**	Удалить файл **Target.**
Копировать разделяемый файл **Sourcr** в **Target**. Увеличить счётчик инсталляций на 1, если он уже существует или присвоить ему 1 в противном случае.	Уменьшить счётчик инсталляций на 1. Если он равен 0, то удалить файл **Target**
Создать раздел **Key** в реестре.	Удалить раздел **Key** в реестре.
Создать параметр **Value** в реестре.	Удалить параметр **Value** в реестре.
Изменить значение параметра **Value** с **Old** на **New.**	Записать в **Value** значение **Old.**
Создать новый INI-файл.	Удалить INI-файл.
Создать секцию в INI-файле.	Удалить секцию в INI-файле.
Записать параметр в секцию INI-файла. Если параметр уже существует, сохранить его содержимое.	Если в журнале сохранено предыдущее содержимое параметра — записываем его. В противном случае удаляем параметр из секции.

В этой таблице записаны некоторые часто встречающиеся операции. Возможно, для ваших целей вам потребуется кое-что ещё.

Обратите внимание, что при деинсталляции журнал должен обрабатываться в обратном порядке. Например, если при инсталляции сначала был создан каталог, а потом в него скопированы несколько файлов, то при деинсталляции сначала удаляются эти файлы, а затем каталог.

В статье, посвящённой системному реестру, рассказано, как поместить свою программу в список программ деинсталляции (Панель управления/Установка и удаление программ).

Что ещё? Самое главное — вы намучаетесь с удалением самой программы деинсталляции с жёткого диска. Дело в том, что программу невозможно удалить до тех пор, пока она запущена — Windows закрывает к ней доступ. Что делать? Если мы посмотрим, как с этой ситуацией справляются распространённые инсталляторы (например, InstallShield и Wise), то увидим, что они оставляют программу деинсталляции на диске. Она становится разделяемым ресурсом, частью операционной системы (помещается в каталог Windows). Например, у меня на диске находятся C:\WINDOWS\UNINST16.EXE, C:\WINDOWS\UNINST.EXE (InstallShield) и C:\WINDOWS\UNWISE.EXE (Wise). Это достаточно корректное решение, поскольку этими инсталляторами пользуются многие программы. Мы можем сделать то же самое, изменив имя программы деинсталляции (uninst и unwise уже заняты :)

Мы также можем скопировать программу деинсталляции во временный каталог и запустить её оттуда. Она, конечно, не будет уничтожена, однако при следующей чистке временного каталога, пользователь её удалит. Какие проблемы могут возникнуть на этом пути? Обратите внимание, что программу деинсталляции мы должны будем скопировать во временный каталог *только во время инсталляции* — в ином случае пользователь может удалить её значительно раньше, чем она ему потребуется для деинсталляции (а откуда он знает, что это за файл?). Значит, вариант может быть таким: скопировать программу, запусить её из нового каталога, а текущую копию завершить. Здесь как раз и находится проблема: как только наша программа будет завершена, фокус будет передан назад в окно **Установка и удаление программ**, а мы как раз начнём запрашивать пользователя, действительно ли он согласен удалить продукт с машины...

Вы можете написать простую программу для того, чтобы убедиться, что выглядит это некрасиво. У этой проблемы может быть несколько решений, например, первая копия может удалить продукт с машины, а затем скопировать себя во временный каталог и перезапуститься уже без главного окна просто для того, чтобы удалить один файл и каталоги, в котором он находится. Или мы можем поместить деинсталлятор только в нашу группу программ, следовательно, пользователь сможет запустить её только оттуда, и у него не возникнет проблем с окном **Установка и удаление программ**.

Наконец, мы можем удалить-таки наш единственный EXE-файл автоматически. Для того, чтобы понять, как это делается, достаточно вспомнить о разделе **RunOnce** системного реестра.

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\ CurrentVersion\RunOnce] SomeName="C:\WINDOWS\COMMAND.COM /C DEL C:\TEMP\DEINST.EXE"

При следующей перезагрузке компьютера будет вызван командый процессор, который и удалит ненужный EXE-файл. Затем параметр **SomeName** будет автоматически удалён из раздела **RunOnce**.

Командным процессором может быть не только **command.com**, но и **cmd.exe** (в NT), **4dos.com**, **4nt.exe** и другие программы. Переменная окружения **COMSPEC** содержит полный путь к текущему командному процессору

---

function GetCommand: **String**; **var** szCommandPath: **array**[0..MAX_PATH - 1] **of** Char; **begin** GetEnvironmentVariable('COMSPEC', szCommandPath, MAX_PATH); Result := **String**(szCommandPath); **end**;