«Инсталлятор приложения»

**Цель работы:** Для созданной ранее утилиты создать инсталлируемый вариант для ОС Windows и ОС Linux. Преобразовать утилиту в службу для Windows и в демон для Linux.

* Используемые системы

Linux

|  |
| --- |
| System**: Linux Mint 18** Sarah x86\_64  Kernel: 4.4.0-34-generic DE: X-Cinnamon Session: cinnamon  ----------------------------------  Processor: Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz  Memory (Gb): 3.7  Video: 00:02.0 VGA compatible controller: Intel Corporation 3rd Gen Core processor Graphics Controller (rev 09) Subsystem: Acer Incorporated [ALI] 3rd Gen Core processor Graphics Controller Kernel driver in use: i915 -- 01:00.0 VGA compatible controller: NVIDIA Corporation GF108M [GeForce GT 630M] (rev a1) Subsystem: Acer Incorporated [ALI] GF108M [GeForce GT 630M] Kernel driver in use: nvidia  ---------------------------------- |

Windows

|  |
| --- |
| C:\Users\Saboteur>systeminfo  Имя узла: SABOTEUR-PC  Название ОС: **Microsoft Windows 7** Максимальная  Версия ОС: 6.1.7601 Service Pack 1 сборка 7601  Изготовитель ОС: Microsoft Corporation  Параметры ОС: Изолированная рабочая станция  Сборка ОС: Multiprocessor Free  Зарегистрированный владелец: Saboteur  Зарегистрированная организация:  Код продукта: 00426-OEM-8992662-00006  Дата установки: 20.09.2015, 2:18:48  Время загрузки системы: 19.12.2015, 11:38:47  Изготовитель системы: Acer  Модель системы: Aspire V3-571G  Тип системы: x64-based PC  Процессор(ы): Число процессоров - 1.  [01]: Intel64 Family 6 Model 58 Stepping 9 Gen  uineIntel ~2501 МГц  Версия BIOS: Acer V1.13, 09.10.2012  Папка Windows: C:\Windows  Системная папка: C:\Windows\system32  Устройство загрузки: \Device\HarddiskVolume1  Язык системы: ru;Русский  Язык ввода: ru;Русский  Часовой пояс: (UTC+04:00) Волгоград, Москва, Санкт-Петербург  Полный объем физической памяти: 3 934 МБ  Доступная физическая память: 1 027 МБ  Виртуальная память: Макс. размер: 7 867 МБ  Виртуальная память: Доступна: 4 672 МБ  Виртуальная память: Используется: 3 195 МБ  Расположение файла подкачки: C:\pagefile.sys |

**Программа работы**

* Создать инсталлируемую версию утилиты для ОС Windows
* Создать инсталлируемую версию утилиты для ОС Linux
* Создать версию утилиты в виде службы Windows
* Создать версию утилиты в виде демона Linux
* Создание инсталлируемую версию утилиты для ОС Windows

Для создания инсталируемых программ используется расширение InstallShield.[1]

Переходим во вкладку создания новых проектов и выбираем в пункте Other Project Type пункт Enable InstallShield Limited Edition.

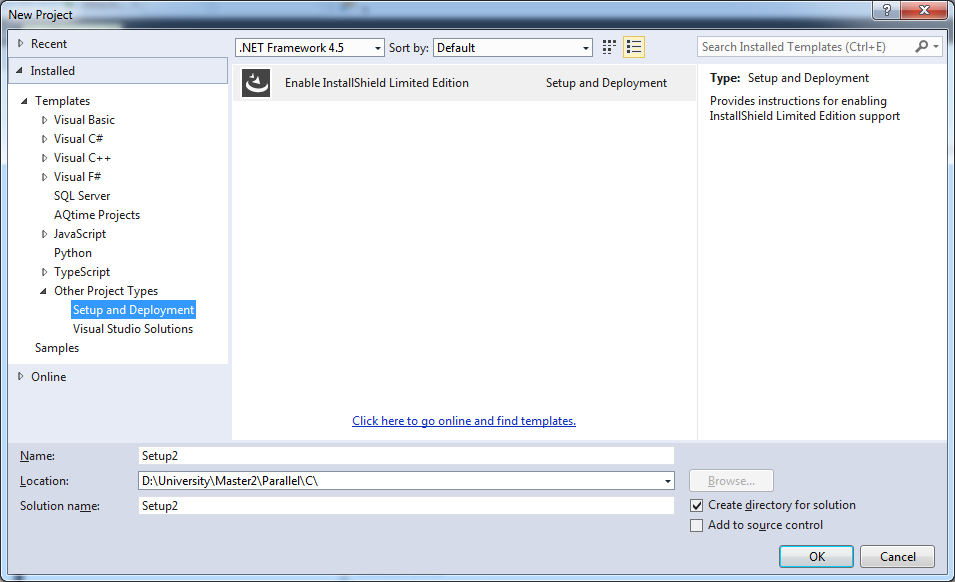


Рис.1. Выбор расширения InstallShield

Процесс скачки и установки легок и интуитивно понятен.

После создания нового проекта появляется окно настройки проекта (рис.3)

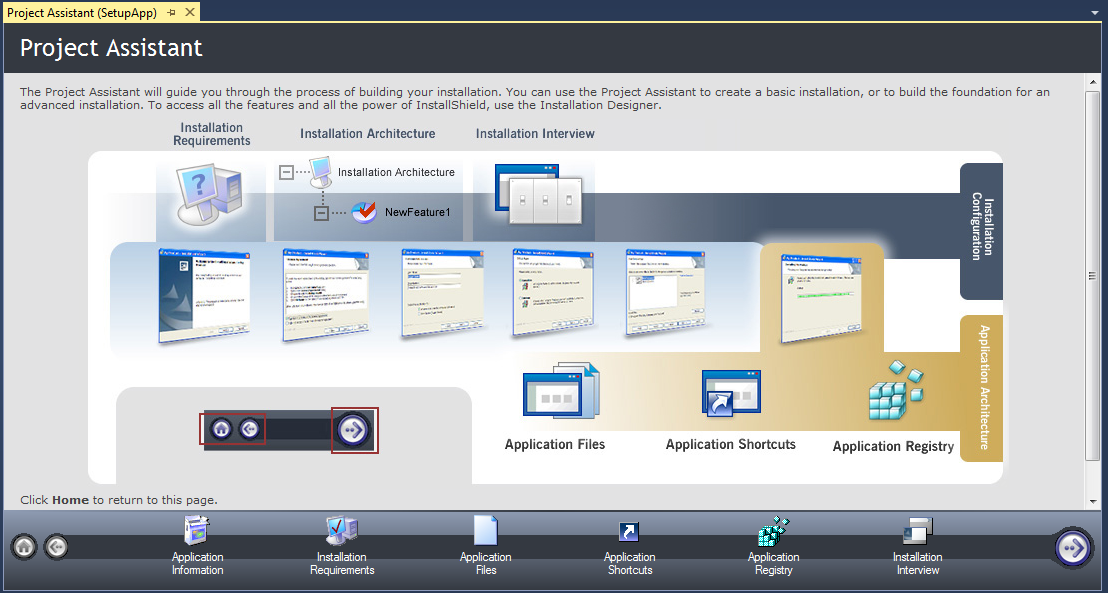


Рис.3. Окно настройки проекта

Настраиваем окно информации о приложении (рис.3)

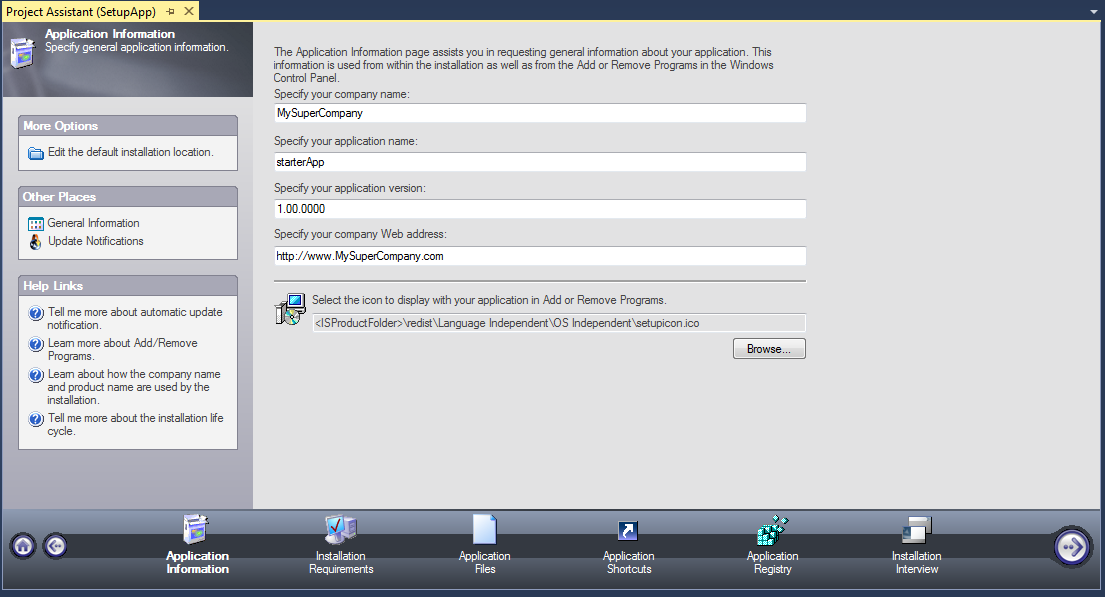


Рис. 4. Окно информации о приложении

Настраиваем окно требований к установке (рис. 5)

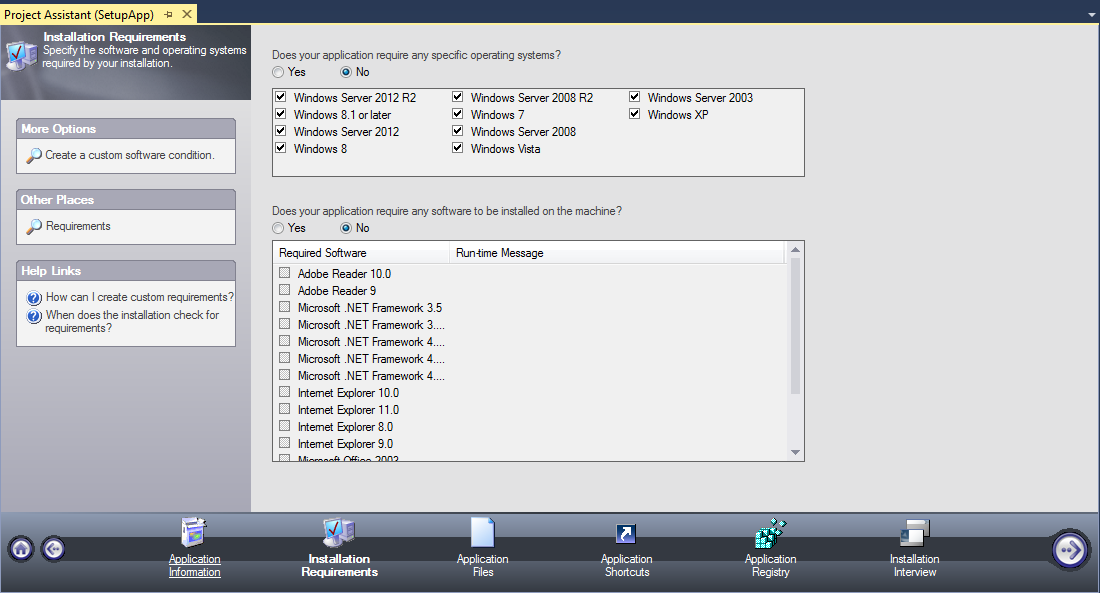


Рис. 5. Окно требований к установке

Во вкладке «Application Files» выбираем все файлы из папки Release целевого приложения (рис.6)

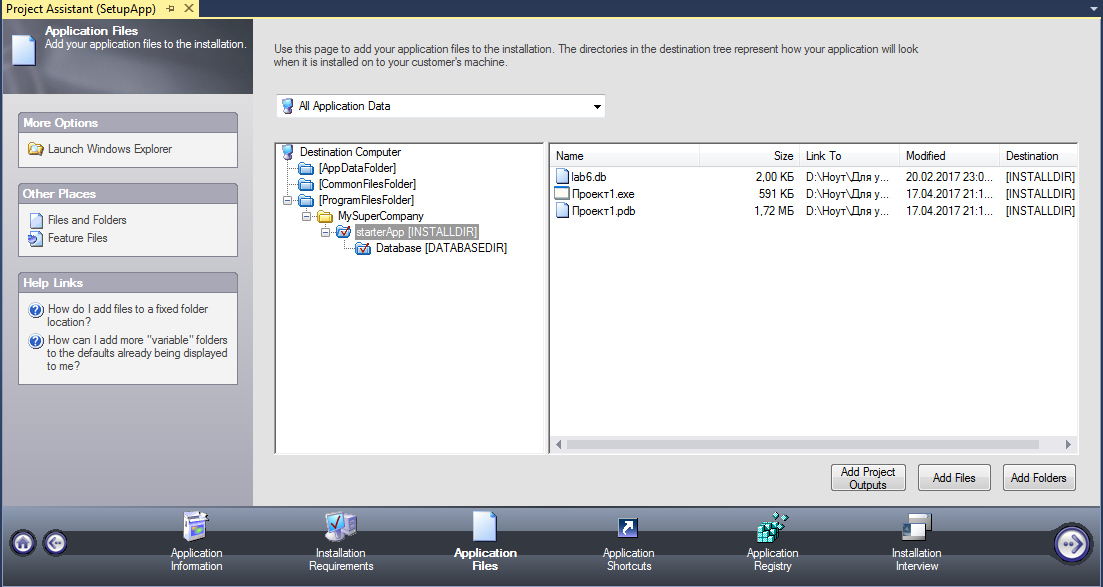


Рис. 6. Окно «Файлы приложения»

Заполняем окно «Ярлыки приложения» (рис.7)

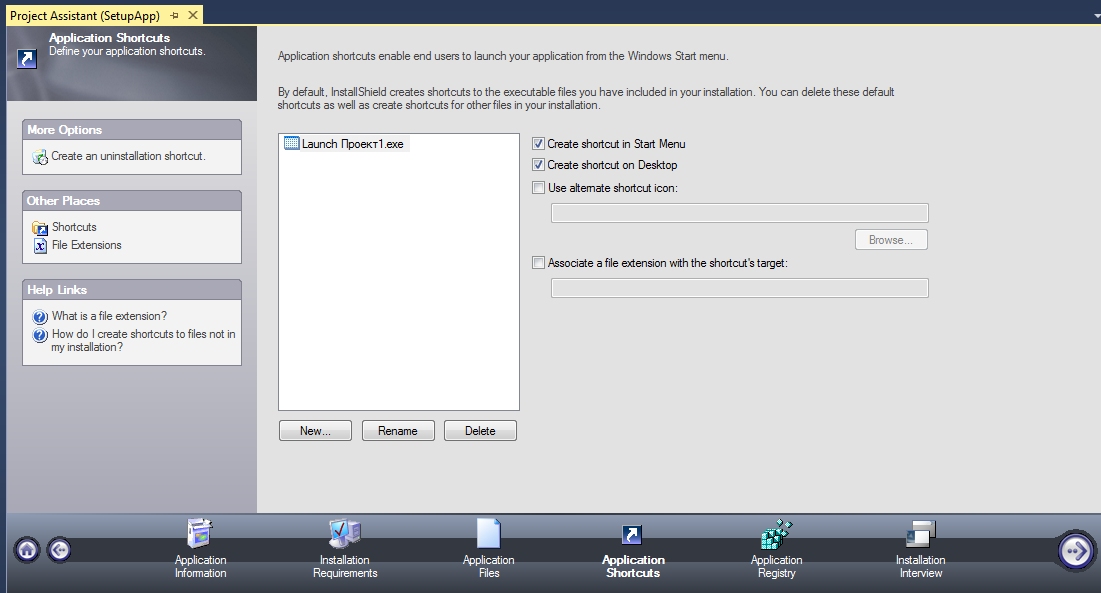


Рис.7. Окно «Ярлыки приложения»

Настраиваем окно «Диалоговое окно установщика»

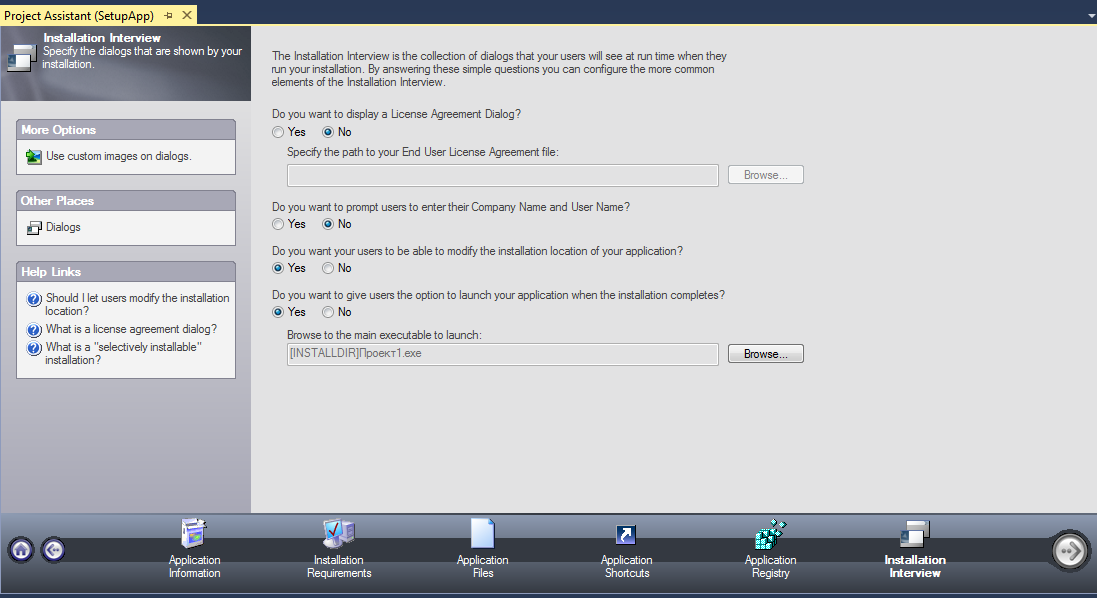


Рис.8. Окно «Диалоговое окно установщика»

После выставления всех настроек можно приступать к сборке инсталлятора.

В результате получаем инсталлирующий файл в папке с проектом.

Попробуем установить свою программу.

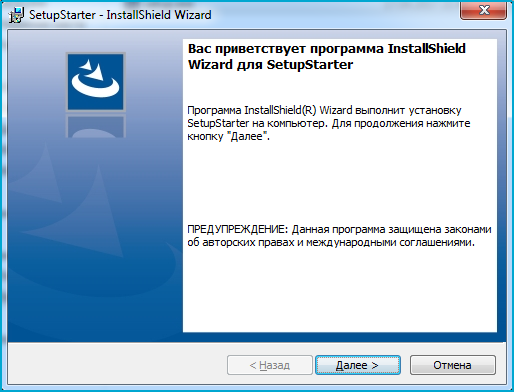


Рис.9. Окно приветствия инсталлятора

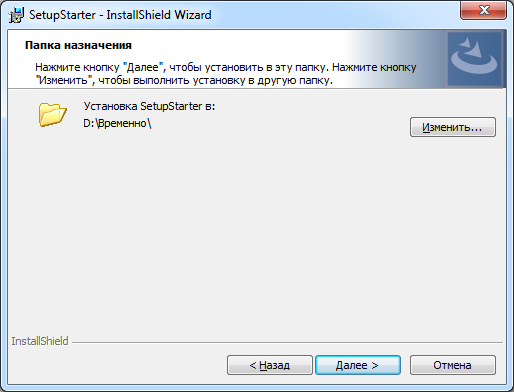


Рис.10. Окно выбора пути установки.

В результате установки, в выбранной папке появились файлы программы, а на рабочем столе иконка приложения.

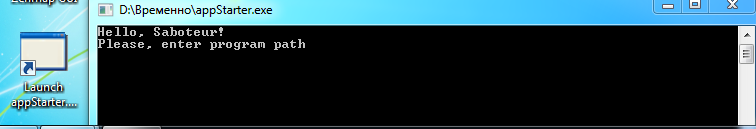


Рис.11. Иконка приложения на рабочем столе

**Вывод**:

Создавать инсталлируемую версию приложения является легким процессом и не занимает много времени. Все действия по созданию инсталлятора автоматизированы, а настройки задаются при помощи графического интерфейса.

* Создание версию утилиты в виде службы Windows.

Слу́жбы ОС Windows (англ. Windows Service, службы) — приложения, автоматически (если настроено) запускаемые системой при запуске Windows и выполняющиеся вне зависимости от статуса пользователя. Имеет общие черты с концепцией демонов в Unix.

Для создания службы из командной строки можно использовать программу SC (Sc.exe). SC представляет из себя утилиту командной строки, которая реализует вызовы ко всем функциям интерфейса прикладного программирования (API) управления службами Windows. С ее помощью можно производить любые действия со службами —  просматривать состояние, управлять (запускать, останавливать и т.п.), изменять параметры, а также создавать новые службы.[2]

При создании службы с помощью SC нет необходимости вручную создавать записи в реестре и затем перезагружать компьютер, чтобы обеспечить обновление базы данных диспетчером служб. Также SC позволяет указать имя удаленного компьютера, что дает возможность управлять службами как на локальном, так и на удаленном компьютере.

Для создания нового сервиса запускаем команду Sc create. Она создает запись службы в реестре и в базе данных диспетчера служб. Sc create имеет следующий синтаксис:

sc create [ServiceName] [binPath= ] <параметр1= > <параметр2= >

ServiceName — указывает имя, которое будет присвоено разделу службы в реестре.   
binPath — указывает путь к исполняемому файлу службы.

Создадим свою службу:

|  |
| --- |
| C:\Windows\system32>sc create MyStarterApp binPath= D:\University\Release\appSta  rter.exe DisplayName= "My starterApp" type= own start= auto  [SC] CreateService: успех |

Проверим результат в списке служб

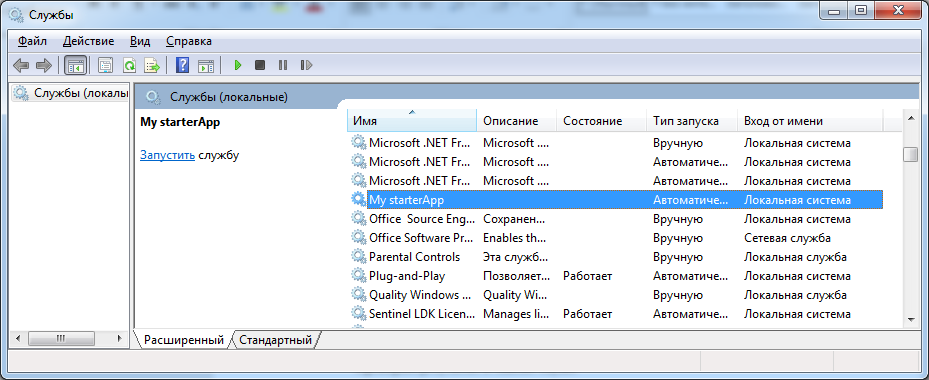


Рис.12. Запуск приложения в виде службы.

Изменять параметры уже созданной службы можно командой Sc config. Например, изменим отображаемое имя службы:

|  |
| --- |
| C:\Windows\system32>sc config MyStarterApp DisplayName= "MySypperStarterApp"  [SC] ChangeServiceConfig: успех |

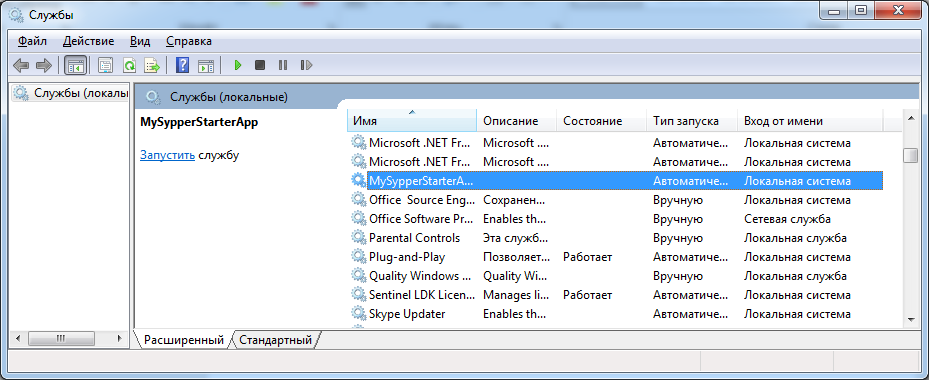


Рис.13. Отображение службы с новым именем

Удаление службы:

|  |
| --- |
| C:\Windows\system32>sc delete MyStarterApp  [SC] DeleteService: успех |

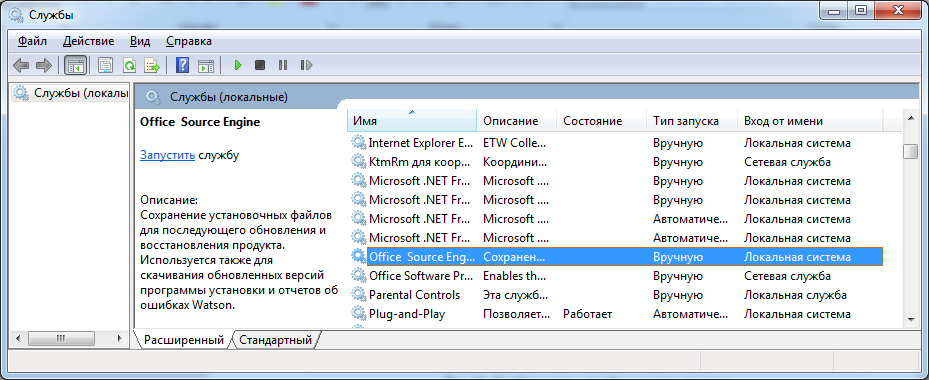


Рис.14. Отображение служб после удаления службы

* Вывод:

Создание службы ОС Windows достаточно простой процесс. Управление созданной службой также не составляет никакого труда. Создать службу указанным способом можно в любой конфигурации Windows, т.к. утилита sc является базовой утилитой.

Создать инсталлируемую версию утилиты для ОС Linux

В качестве инсталлируемой утилиты будем создавать бинарный deb пакет.

Для начала необходимо создать папку, которая в дальнейшем будет преобразована в пакет.[3]

mkdir starterApp

Содержимое папки должно иметь определенное строение. Для начала выведем строение конечного результата:

|  |
| --- |
| user3@user-Aspire-V3-571G ~/Univer/debPackeg $ tree statLib  statLib  ├── DEBIAN  │   ├── control  │   └── copyright  └── tmp  └── appStarter  ├── appStatLib  └── lab6.db |

Далее рассмотрим содержимое созданной директории

Директория DEBIAN содержит фалы настроек, а все файлы вне этой директории представляют собой вид создаваемой программы относительно корневой папки. Т.е. в данном случае предполагается что в директории /tmp/appStarter будет создано два файла: файл программы и файл базы данных. Два этих файла и есть файлы программы.

Рассмотрим подробнее директорию DEBIAN.

В ней могут содержаться различные конфигурационные файлы:

* control: Основная информация

##### changelog: история изменений

##### rules: правила компиляции

##### conffiles: список файлов конфигурации

##### dirs: список папок для создания

##### menu: создание пунктов меню

##### md5sums: контрольные суммы файлов

##### watch: мониторинг сайта, откуда была скачана прога

##### /inid.d: init-скрипт

##### Главным же файлом является файл control. Он может быть и единственным.

##### Рассмотрим его содержимое, которое является минимальным необходимым набором:

|  |
| --- |
| Package: starterApp  Version: 1.0-1  Section: misc  Architecture: all  Depends: bash, sed (>= 3.02-8)  Maintainer: o\_O Tync <o-o-tync.habrahabr.ru>  Description: Super Shell Script  A super example script.  .  It does nothing :) |

##### Package – имя пакета.

##### Version – версия программы.

##### Section - определяет задачу, для которой приложение обычно используется (группа приложений).

Architecture - архитектура процессора, для которой предназначен пакет.

Depends - список пакетов через запятую, которые требуются для установки этого пакета.

Maintainer - имя и почта мэйнтейнера пакета: человека, который «дебианизировал» приложение.

Description - Описание пакета.

Сборка пакета:

|  |
| --- |
| user3@user-Aspire-V3-571G ~/Univer/debPackeg $ fakeroot dpkg-deb --build statLibdpkg-deb: сборка пакета «starterapp» в файл «statLib.deb».  user3@user-Aspire-V3-571G ~/Univer/debPackeg $ mv statLib.deb statLib\_1.0-1\_all.deb |

Установка, проверка и удаление пакета.

|  |
| --- |
| user3@user-Aspire-V3-571G ~/Univer/debPackeg $ **sudo dpkg -i statLib\_1.0-1\_all.deb**  Выбор ранее не выбранного пакета starterapp.  (Чтение базы данных … на данный момент установлено 239048 файлов и каталогов.)  Подготовка к распаковке statLib\_1.0-1\_all.deb …  Распаковывается starterapp (1.0-1) …  Настраивается пакет starterapp (1.0-1) …  user3@user-Aspire-V3-571G ~/Univer/debPackeg $ **ls /tmp/appStarter/**  appStatLib lab6.db  user3@user-Aspire-V3-571G ~/Univer/debPackeg $ **sudo dpkg -r starterapp** (Чтение базы данных … на данный момент установлено 239050 файлов и каталогов.)  Удаляется starterapp (1.0-1) …  user3@user-Aspire-V3-571G ~/Univer/debPackeg $ l**s /tmp/appStarter/**  ls: cannot access '/tmp/appStarter/': No such file or directory  user3@user-Aspire-V3-571G ~/Univer/debPackeg $ |

**Вывод:**

Создание установочного пакета в Linux несколько сложнее чем создание инсталлятора в Windows. Однако установочный пакет имеет больше параметров и может быть легко настроен для создания более сложных сценариев установки. Например есть возможность создания инициализационных скриптов.

Создать версию утилиты в виде демона Linux

Демон — компьютерная программа в системах класса UNIX, запускаемая самой системой и работающая в фоновом режиме без прямого взаимодействия с пользователем.

Для того чтобы программы стала демоном, необходимо внести некоторые изменения в ее исходный код. Изменения вносятся в main функцию программы.

Суть изменений в том, чтобы создать дочерний процесс в котором будет выполняться программа демон, а процесс родитель закрыть. Это нужно для того, чтобы отделить демон от управляющего терминала и перевести его в фоновый режим.

|  |
| --- |
| int main(int argc,char \*\*argv)  {  writeLog("Init process was Start");    pid\_t ppid, sid;    ppid = fork(); //создаем дочерний процесс  if(ppid < 0)  {  exit(1); //ошибка если не удалось создать процесс  }  else if(ppid != 0)  {  exit(0); // завершаем родительский процесс  }  //обновляем процесс потомок  umask(0);//даем права на работу с фс  sid = setsid();//генерируем уникальный индекс процесса  if(sid<0)  {  exit(1);  }  if((chdir("/"))<0) //выходим в корень фс  {  exit(1);  }    close(STDIN\_FILENO);//закрываем доступ к стандартным потокам ввода-вывода  close(STDOUT\_FILENO); //для того чтобы не нагружать процесс лишними действиями  close(STDERR\_FILENO);  RouterSnifDeamon(); //вызываем функцию, инициирующую работу основной программы  return 0;  } |

Переход в корень диска, необходим для того, чтобы впоследствии не было проблем связанных с размонтированием дисков. Если текущая папка демона будет находиться на диске, который необходимо будет отмонтировать, то система не даст этого, до тех пор, пока демон не будет остановлен.

Далее необходимо добавить программу в автозагрузку. Для этого необходимо создать свой скрипт автозагрузки.

Создаём пустой файл.

Первой строкой пишем:

#!/bin/sh

Данная строка указывает, какую командную оболочку необходимо использовать. Дальше свои команды.

|  |
| --- |
| #!/bin/sh  /home/user/NetBeansProjects/RouterSnif/dist/Debug/GNU-Linux/routersnif |

Сохраним его под оригинальным названием (чтоб не совпадал с уже существующими) в каталоге /usr/sbin/.

Чтобы скрипт запускался при загрузке, необходимо прописать его в файле /etc/rc.local до строчки exit 0. Если у вас не существует данного файла, создайте его и вставьте в него следующее содержимое:

|  |
| --- |
| #!/bin/sh -e  #Здесь вставляем строку с указанием вашего скрипта.  /usr/sbin/rSnif  exit 0 |

Перезагружаем компьютер, чтобы проверить автозапуск утилиты.

|  |
| --- |
| user3@user-Aspire-V3-571G ~ $ ps -A -o "ruser pid ppid start command" | head -n 1&& ps -A -o "ruser pid ppid start command" | grep -v grep |grep -E "init|routercsniff"  RUSER PID PPID STARTED COMMAND  root 1 0 00:43:47 /sbin/init splash  root 1237 1 00:44:23 /home/user3/NetBeansProjects/RouterCSniff/dist/Debug/GNU-Linux/routercsniff |

Из вывода видно, что процесс routercsniff запущен init процессом с ppid 1. Также время запуска утилиты указывает на то, что она была запущена при старте системы.

Вывод

В данной работе получены знания по созданию файла инсталляции для любой программы в ОС Windows. Создан свой инсталлирующий файл.

Получены знания по созданию службы Windows на основе собственной утилиты. Проведены опыты по управлению этой службой.

В ОС Linux создан установочный deb пакет. Изучена структура установочной директории.

Создан свой демон, который способен запускаться при старте операционной системы.

Список используемых источников:

1. Создание инсталлятора в Visual Studio. [URL:http://vscode.ru/articles/sozdanie-installyatora-v-visual-studio.html](url:http://vscode.ru/articles/sozdanie-installyatora-v-visual-studio.html)
2. Создание службы в Windows. URL:<http://windowsnotes.ru/cmd/sozdanie-sluzhby-v-windows/>
3. Как собрать бинарный deb пакет: подробное HowTo. <https://habrahabr.ru/post/78094/>

# Создание своего скрипта автозапуска при загрузке компьютера URL: http://help.ubuntu.ru/wiki/%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5\_%D1%81%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%B0\_%D0%B0%D0%B2%D1%82%D0%BE%D0%B7%D0%B0%D0%BF%D1%83%D1%81%D0%BA%D0%B0