Руководство разработчика по пакетированию приложений

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Номер по каталогу: 820–5498–12
Апрель 2009 г.

090518@22510

Содержание

Введение

В руководстве *Руководство разработчика по пакетированию приложений* содержатся поэтапные указания и соответствующие дополнительные сведения по разработке, сборке и проверке пакетов. В состав настоящего руководство входят также дополнительные приемы, которые могут оказаться полезными во время создания пакетов.

Примечание – Рассматриваемый выпуск Solaris™ поддерживает системы, в которых используются процессорные архитектуры SPARC® и x86 следующих семейств: UltraSPARC®, SPARC64, AMD64, Pentium и Xeon EM64T. Поддерживаемые системы перечислены в *списке совместимого оборудования для ОС Solaris*, ссылка на который приведена на странице `http://www.sun.com/bigadmin/hcl`. В настоящем документе учитываются различия в реализации между платформами различных типов.

Термины, относящиеся к платформе x86, имеют в данном документе следующие значения:

- Термин "x86" относится к расширенному семейству 64-разрядных и 32-разрядных продуктов, совместимых с x86.
- Термин "x64" указывает на то, что информация относится к 64-разрядным системам AMD64 или EM64T.
- Термин "32-разрядный x86" указывает на то, что информация относится к 32-разрядным системам на базе x86.

Поддерживаемые системы перечислены в *списке совместимого оборудования для ОС Solaris*.

Целевая аудитория

Настоящее руководство предназначено для разработчиков приложений, в обязанности которых входит разработка и сборка пакетов.

Хотя большая часть руководства рассчитана на начинающих разработчиков пакетов, в нем содержится и информация, полезная для более опытных разработчиков пакетов.

Структура руководства

Разделы настоящего руководства перечислены в следующей таблице.

Название главы	Описание главы
Глава 1, «Разработка пакета»	Описываются компоненты пакета и критерии разработки пакета. Также рассматриваются связанные с этим процессом команды, файлы и сценарии.
Глава 2, «Сборка пакета»	Описывается процесс и обязательные задачи при сборке пакета. Также приводятся поэтапные указания для каждой задачи.
Глава 3, «Расширение функциональности пакета (задачи)»	Приводятся поэтапные указания по добавлению необязательных функций в пакет.
Глава 4, «Проверка и запись пакета»	Описывается процесс проверки целостности пакета и переноса пакета на распространяемый носитель.
Глава 5, «Практические примеры создания пакета»	Приводятся практические примеры создания пакетов.
Глава 6, «Дополнительные методы создания пакетов»	Описываются более сложные методы создания пакетов.
Глоссарий	Определяются термины, используемые в данном руководстве.

Дополнительная литература

В приведенной ниже документации, которую можно приобрести в розничной продаже, предоставлены дополнительные сведения по созданию пакетов System V.

- *Двоичный интерфейс приложений System V*
- *Двоичный интерфейс приложений System V - дополнение для процессора SPARC*
- *Двоичный интерфейс приложений System V - дополнение для процессора Intel386*

Документация, поддержка и обучение

На веб-сайте Sun можно найти информацию по следующим дополнительным ресурсам:

- Документация (`http://www.sun.com/documentation/`)
- Техническая поддержка (`http://www.sun.com/support/`)
- Обучение (`http://www.sun.com/training/`)

Компания Sun охотно ознакомится с Вашими комментариями

Компания Sun заинтересована в совершенствовании документации, поэтому просим направлять замечания и пожелания. Комментарии можно оставить по вдресу `http://docs.sun.com`, перейдя по ссылке "Feedback".

Типографские условные обозначения

В следующей таблице приведены типографские условные обозначения, используемые в настоящем руководстве.

ТАБЛИЦА P–1 Типографские условные обозначения

Шрифт	Описание	Пример
`AaBbCc123`	Имена команд, файлов и каталогов, а также данные, выводимые на экран компьютера.	Отредактируйте файл `.login`. Для вывода списка всех файлов используйте команду `ls -a`. `machine_name%, вам письмо.`
`AaBbCc123`	Текст, вводимый пользователем (в отличие от выводимых системой данных).	`machine_name%` **`su`** `Пароль:`
aabbcc123	Местозаполнитель: заменяется фактическим именем или значением.	Команда для удаления файла: `rm` *filename*.
AaBbCc123	Названия руководств, новые термины и термины, на которые следует обратить особое внимание.	См. Главу 6 в документе *Руководство пользователя*. *Кэш* – это копия, сохраненная локально. *Не сохраняйте* файл. **Примечание.** Некоторые выделенные элементы в интерактивном режиме выглядят полужирными.

Запросы интерпретатора в примерах команд

В следующей таблице представлен запрос системы UNIX® по умолчанию, а также запросы суперпользователя для программ csh, sh и ksh.

ТАБЛИЦА P–2 Запросы интерпретатора

Интерпретатор команд	Запрос
csh	`machine_name%`
csh для суперпользователя	`machine_name#`
sh и ksh	`$`
sh и ksh для суперпользователя	`#`

◆ ◆ ◆ ГЛАВА 1

1 Разработка пакета

Перед сборкой пакета необходимо знать, какие файлы следует создать и какие команды выполнить. Необходимо принять во внимание требования вашего приложения к программному обеспечению компьютера, а также потребности вашего клиента. Вашими клиентами являются администраторы, которые будут устанавливать пакет. В данной главе речь идет о файлах, командах и критериях, которые нужно знать и принимать во внимание до начала сборки пакета.

Ниже приведен перечень вопросов, рассмотренных в данной главе.

Где найти задачи по пакетированию

Ниже приведены карты задач с поэтапными указаниями для сборки и проверки пакетов.

Что такое пакеты?

Прикладное программное обеспечение поставляется в виде блоков, называемых *пакетами*. Пакет - это набор файлов и каталогов, требуемых для работы программного продукта. Пакет обычно разрабатывается и собирается разработчиком приложения после завершения разработки кода приложения. Программный продукт необходимо объединить в один или несколько пакетов для того, чтобы его легче было переносить на распространяемый носитель. После этого программный продукт может быть тиражирован и установлен администраторами.

Пакет - это набор файлов и каталогов в определенном формате. Этот формат соответствует двоичному интерфейсу приложений (ABI), являющемуся частью определения интерфейса System V.

Компоненты пакета

Существуют две категории компонентов пакета:

- *объекты пакета* - файлы устанавливаемого приложения;
- *контрольные файлы* - контролируют, как, куда и при каких условиях устанавливается пакет.

Контрольные файлы также разделены на две категории: *информационные файлы* и *сценарии установки*. Некоторые контрольные файлы являются обязательными. Некоторые контрольные файлы являются необязательными.

Перед созданием пакета приложений необходимо создать обязательные и необязательные компоненты, которые будут составлять пакет. После этого можно выполнить сборку пакета с помощью команды `pkgmk`.

Для сборки пакета необходимо иметь следующее.

- Объекты пакета (файлы и каталоги прикладного ПО)
- Два обязательных информационных файла (`pkginfo` и `prototype`)
- Необязательные информационные файлы
- Необязательные сценарии установки

На рисунке ниже описано содержимое пакета.

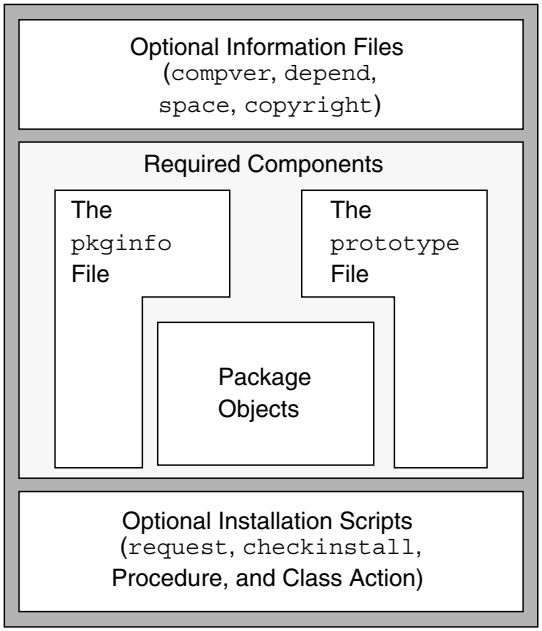


РИСУНОК 1–1 Содержимое пакета

Обязательные компоненты пакета

Перед сборкой пакета необходимо создать следующие компоненты.

- Объекты пакета

  Эти компоненты образуют приложение. Они могут состоять из следующих элементов.

  - Файлы (исполняемые файлы или файлы данных)
  - Каталоги
  - Именованные каналы
  - Ссылки
  - Устройства

- Файл `pkginfo`

  Файл `pkginfo` содержит обязательную информацию о пакете, определяющую значения параметров. В число значений параметров входят краткое название пакета, полное название пакета и архитектура пакета. Для получения дополнительной информации см. раздел «Создание файла `pkginfo`» на стр. 28 и справочную страницу `pkginfo(4)`.

Примечание – Есть две справочные страницы с названием `pkginfo(1)` На первой справочной странице описана команда раздела 1, которая отображает информацию об установленных пакетах. На второй справочной странице описан файл раздела 4, содержащий характеристики пакета. При работе со справочными страницами обязательно указывайте соответствующий раздел справочной страницы. Пример: `man -s 4 pkginfo`.

- Файл `prototype`

  Файл `prototype` содержит обязательную информацию о пакете. В нем перечислены все компоненты пакета. Для каждого объекта пакета, информационного файла и сценария установки имеется одна запись. Эта запись состоит из нескольких полей с данными, которые описывают каждый компонент, включая его местоположение, атрибуты и тип файла. Для получения дополнительной информации см. раздел «Создание файла `prototype`» на стр. 34 и справочную страницу `prototype(4)`.

Необязательные компоненты пакета

Информационные файлы пакета

В пакет можно включить четыре необязательных информационных файла:

- Файл `compver`

  Определяет предыдущие версии пакета, которые совместимы с данной версией.

- Файл `depend`

  Указывает другие пакеты, у которых есть особая связь с вашим пакетом.

- Файл `space`

  Определяет потребность в дополнительном дисковом пространстве на целевом компьютере помимо того, которое требуется для объектов, указанных в файле `prototype`. Например, дополнительное пространство может понадобиться для файлов, которые динамически создаются во время установки пакета.

- Файл `copyright`

  Определяет текст сообщения об авторских правах, которое показывается во время установки пакета.

Каждый информационный файл пакета должен иметь запись в файле `prototype`. Для получения дополнительной информации о создании этих файлов см. раздел «Создание информационных файлов» на стр. 57.

Сценарии установки пакета

Сценарии установки не являются обязательными. Тем не менее, можно снабдить пакет сценариями, которые предлагают действия, выполняемые пользователями во время установки пакета. Сценарий установки имеет следующие характеристики.

- Сценарий состоит из команд интерпретатора sh.
- Права для файла сценария должны быть установлены на 0644.
- Наличие в сценарии идентификатора интерпретатора команд (`#! /bin/sh`) не обязательно.

Существуют четыре типа сценариев.

- Сценарий `request`

  Сценарий `request` запрашивает ввод от администратора, устанавливающего пакет.
- Сценарий `checkinstall`

  Сценарий `checkinstall` производит специальную верификацию файловой системы.

  Примечание – Сценарий `checkinstall` доступен только в выпуске Solaris™ 2.5 и совместимых выпусках.

- Процедурные сценарии

  Процедурные сценарии определяют действия, которые выполняются в определенный момент установки и удаления пакета. Можно создать четыре процедурных сценария со следующими заранее установленными именами: `preinstall`, `postinstall`, `preremove` и `postremove`.
- Сценарии действий над классами

  Сценарии действий над классами определяют набор действий, которые будут выполняться над группой объектов.

Для получения дополнительной информации о сценариях установки см. раздел «Создание сценариев установки» на стр. 63.

Что следует принять во внимание перед сборкой пакета

Перед сборкой пакета необходимо решить, будет ли ваш продукт состоять из одного или нескольких пакетов. Обратите внимание, что установка нескольких маленьких пакетов занимает больше времени, чем установка одного большого пакета. Несмотря на то, что создание одного пакета более рационально, это не всегда возможно. Если принято решение собрать несколько пакетов, следует определить, каким образом сегментировать код приложения. В данном разделе представлен перечень критериев, которые необходимо использовать при планировании сборки пакета.

Многие критерии по созданию пакета предполагают компромисс между ними. Удовлетворить все требования в равной степени часто очень сложно. Критерии представлены по степени их важности. Однако эта последовательность служит лишь примерным руководством, и ее можно изменить в зависимости от обстоятельств. Хотя каждый из этих критериев является важным, вопрос о том, как оптимизировать эти потребности для получения хорошего набора пакетов, разработчику предстоит решать самому.

Дополнительные идеи по разработке пакетов содержатся в Глава 6, «Дополнительные методы создания пакетов».

Пакеты должны иметь возможность удаленной установки

Все пакеты должны *допускать удаленную установку*. Возможность удаленной установки означает, что администратор при установке пакета может устанавливать его на клиентской системе не обязательно в корневую (`/`) файловую систему, где выполняется команда `pkgadd`.

Оптимизация для клиент-серверных структур

Предусмотрите различные типы составов и настроек системного ПО (например, автономная система и сервер) при планировании пакетов. В хорошо спланированном пакете происходит разделение файлов для оптимизации установки на любом типе структуры. Например, содержимое корневой файловой системы (`/`) и файловой системы `/usr` должно быть сегментировано для облегчения поддержки серверных настроек.

Разделение на пакеты в соответствии с функциональностью

Пакеты должны быть самодостаточными с четко определенным набором функциональности. Например, в пакет, содержащий UFS, должны входить все служебные программы UFS, и в нем не должно быть никаких двоичных файлов, не относящихся к UFS.

Пакеты должны быть упорядочены в функциональные блоки с точки зрения потребителя.

Разделение на пакеты в соответствии с лицензионным режимом

Помещайте код, требующий лицензионных выплат по договорным соглашениям, в отдельный пакет или группу пакетов. Не распределяйте этот код по большему числу пакетов, чем необходимо.

Разделение на пакеты в соответствии с зависимостями от системы

Сохраняйте зависимые от системы двоичные файлы в отдельных пакетах. Например, код ядра должен находиться в отдельном пакете, а каждая реализуемая архитектура состоять из отдельного экземпляра пакета. Это правило применимо также к двоичным файлам для различных архитектур. Например, двоичные файлы для системы SPARC будут находиться в одном пакете, а файлы для системы x86 - в другом.

Предотвращение перекрытия пакетов

При создании пакетов старайтесь по возможности избегать дублирования файлов. Ненужное дублирование приводит к сложностям в обслуживании и проблемам с версиями файла. Если продукт состоит из нескольких пакетов, постоянно сравнивайте содержимое этих пакетов для выявления дублированных файлов.

Разделение на пакеты в соответствии с локализацией

Элементы локализации должны находиться в отдельном пакете. При идеальном разделении на пакеты все локализации продукта должны поставляться по схеме: одна национальная настройка - один пакет. К сожалению, в некоторых случаях организационные критерии конфликтуют с функциональными критериями и критериями разделения на продукты.

Международные настройки по умолчанию также могут поставляться в отдельном пакете. Такая структура делает возможной изоляцию файлов, необходимые для изменений в локализации, и стандартизацию формата поставки пакетов локализации.

Пакеты системы IPS

В этом документе приведено описание пакетов SVR4. Если предполагается использование в ОС OpenSolaris, можно воспользоваться пакетами системы IPS (Image Packaging System). В ОС OpenSolaris реализована поддержка пакетов SVR4 и IPS. Система IPS взаимодействует с сетевыми хранилищами и использует файловую систему ZFS. В ОС OpenSolaris доступна возможность публикации существующих пакетов SVR4 в хранилище IPS с помощью команды `pkgsend(1)`.

В следующей таблице представлено сравнение команд для систем пакетирования SVR4 и IPS. Для получения более подробной информации о IPS см. документ Getting Started With the Image Packaging System (`http://dlc.sun.com/osol/docs/content/IPS/ggcph.html`) (Начало работы с системой IPS).

ТАБЛИЦА 1–1 Задачи пакетирования: IPS и SVR4

Задача	Команда IPS	Команда SVR4
Установка нового пакета	`pkg install`	`pkgadd -a`
Просмотр информации о состоянии пакета	`pkg list`	`pkginfo`
Проверка правильности установки пакета	`pkg verify`	`pkgchk -v`
Просмотр информации о пакете	`pkg info`	`pkginfo -l`
Просмотр списка содержимого пакета	`pkg contents`	`pkgchk -l`
Удаление пакета	`pkg uninstall`	`pkgrm`

Команды, файлы и сценарии, используемые при создании пакета

В данном разделе описаны команды, файлы и сценарии, которые можно использовать при разработке пакетов. Все они описаны в справочных страницах и детально рассмотрены в этом руководстве в соответствии с конкретной задачей, которую они выполняют.

В приведенной ниже таблице представлены команды, которые помогут собрать пакет, проверить его, установить его и получить сведения о нем.

ТАБЛИЦА 1–2 Команды для пакетов

Задача	Команда/ справочная страница	Описание	Дополнительная информация
Создание пакетов	pkgproto(1)	Создает файл prototype для ввода в команду pkgmk	«Пример. Создание файла prototypec помощью команды pkgproto» на стр. 40
pkgmk(1)	Создает устанавливаемый пакет	«Сборка пакета» на стр. 48	
Установка, удаление и перенос пакетов	pkgadd(1M)	Устанавливает пакет ПО в систему	«Установка пакетов ПО» на стр. 94
pkgask(1M)	Сохраняет ответы на сценарий запроса request	«Правила разработки сценариев request » на стр. 68	
pkgtrans(1)	Копирует пакеты на распространяемый носитель	«Запись пакета на распространяемый носитель» на стр. 104	
pkgrm(1M)	Удаляет пакет из системы	«Удаление пакета» на стр. 104	
Получение информации о пакетах	pkgchk(1M)	Проверяет целостность пакета ПО	«Проверка целостности пакета» на стр. 96
pkginfo(1)	Отображает информацию о пакете ПО	«Команда pkginfo» на стр. 100	
pkgparam(1)	Отображает значения параметров пакета	«Команда pkgparam» на стр. 98	
Изменение установленных пакетов	installf(1M)	Внедряет новый объект пакета в уже установленный пакет	«Правила разработки процедурных сценариев» на стр. 74 и Глава 5, «Практические примеры создания пакета»
removef(1M)	Удаляет объект пакета из уже установленного пакета	«Правила разработки процедурных сценариев» на стр. 74	

В таблице ниже представлены информационные файлы, помогающие собрать пакет.

ТАБЛИЦА 1–3 Информационные файлы пакета

Файл	Описание	Дополнительная информация
admin(4)	Файл со значениями по умолчанию для установки пакета	«Файл административных значений по умолчанию» на стр. 138
compver(4)	Файл совместимости пакета	«Определение зависимостей пакета» на стр. 57
copyright(4)	Информация об авторских правах пакета	«Создание сообщения об авторских правах» на стр. 60
depend(4)	Файл зависимостей пакета	«Определение зависимостей пакета» на стр. 57
pkginfo(4)	Файл характеристик пакета	«Создание файла pkginfo» на стр. 28
pkgmap(4)	Файл описания содержимого пакета	«Файл pkgmap» на стр. 49
prototype(4)	Информационный файл пакета	«Создание файла prototype» на стр. 34
space(4)	Файл с информацией о требуемом месте на диске для пакета	«Резервирование дополнительного места на диске на целевой системе» на стр. 61

В приведенной ниже таблице представлены необязательные сценарии установки, которые влияют на процесс установки пакета.

ТАБЛИЦА 1–4 Сценарии установки пакета

Сценарий	Описание	Дополнительная информация
request	Запрашивает информацию у установщика	«Создание сценария request » на стр. 68
checkinstall	Собирает данные о файловой системе	«Сбор данных о файловой системе с помощь сценария checkinstall » на стр. 70
preinstall	Выполняет требования клиентской установки перед установкой класса	«Создание процедурных сценариев » на стр. 73
postinstall	Выполняет требования клиентской установки после того, как все тома установлены	«Создание процедурных сценариев » на стр. 73
preremove	Выполняет требования клиентской установки перед удалением класса	«Создание процедурных сценариев » на стр. 73

ТАБЛИЦА 1–4 Сценарии установки пакета *(Продолжение)*

Сценарий	Описание	Дополнительная информация
`postremove`	Выполняет требования клиентской установки после того, как все классы были удалены	«Создание процедурных сценариев » на стр. 73
Действие над классом	Выполняет ряд действий над определенной группой объектов	«Создание сценариев действий над классами» на стр. 75

◆ ◆ ◆ ГЛАВА 2

Сборка пакета

В данной главе рассказывается о процессе и задачах, связанных со сборкой пакета. Некоторые из этих задач являются обязательными. Некоторые из этих задач являются необязательными. Обязательные задачи детально рассмотрены в данной главе. Для получения сведений о необязательных задачах, которые позволяют придать пакету дополнительную функциональность, см. Глава 3, «Расширение функциональности пакета (задачи)» и Глава 6, «Дополнительные методы создания пакетов».

Ниже следует перечень вопросов, рассматриваемых в данной главе.

Процесс сборки пакета (Карта задач)

В Таблица 2–1 описана процедура, которой необходимо придерживаться при сборке пакетов, особенно при отсутствии опыта их сборки. Хотя порядок выполнения первых четырех задач не является обязательным, он обеспечивает наибольшую простоту выполняемых действий. Накопив опыт в области создания пакетов, вы сможете переставлять эти задачи в любой наиболее удобной для вас последовательности.

Став опытным разработчиком пакетов, вы сможете автоматизировать процесс их сборки с помощью команды `make` и файлов Makefile. Для получения дополнительной информации см. справочную страницу о команде `make(1S)`.

ТАБЛИЦА 2–1 Процесс сборки пакета (Карта задач)

Задача	Описание	Инструкции
1. Создать файл `pkginfo`	Создать файл `pkginfo` для описания характеристик пакета.	«Как создать файл `pkginfo`» на стр. 32
2. Упорядочить содержимое пакета	Упорядочить компоненты пакета в иерархическую структуру каталогов.	«Упорядочение содержимого пакета» на стр. 32
3. (не обязательно) Создать информационные файлы	Определить зависимости пакета, включить сообщение об авторских правах и зарезервировать дополнительное место на целевой системе.	Глава 3, «Расширение функциональности пакета (задачи)»
4. (не обязательно) Создать сценарии установки	Настроить процессы установки и удаления пакета.	Глава 3, «Расширение функциональности пакета (задачи)»
5. Создать файл `prototype`	Описать объект пакета в файле `prototype`.	«Создание файла `prototype`» на стр. 34
6.Собрать пакет	Собрать пакет с помощью команды `pkgmk`.	«Сборка пакета» на стр. 48
7. Проверить и записать пакет	Проверить целостность пакета перед копированием его на распространяемый носитель.	Глава 4, «Проверка и запись пакета»

Переменные среды пакета

Переменные среды можно использовать в обязательных информационных файлах, `pkginfo` и `prototype`. Кроме того, можно использовать параметр для команды `pkgmk`, используемой для сборки пакета. Поскольку в данной главе обсуждаются эти файлы и команды, здесь представлена дополнительная информации по переменным, зависящая от контекста. Однако перед началом сборки пакета следует ознакомиться с различными типами переменных и их возможным влиянием на успешное создание пакета.

Существуют два типа переменных:

- Переменные, используемые при сборке пакета (переменные сборки)

  Переменные сборки начинаются со строчной буквы, и их значение определяется в *период сборки*, когда производится сборка пакета с помощью команды `pkgmk`.

- Переменные, используемые при установке пакета (переменные установки)

  Переменные установки начинаются с заглавной буквы, и их значения определяются в *период установки*, когда пакет устанавливается с помощью команды `pkgadd`.

Общие правила использования переменных среды

В файле `pkginfo` определение переменной имеет форму *PARAM=value* (Параметр=значение), и первая буква в *PARAM* является заглавной. Значения этих

переменных определяются только во время установки. Если значение одной из этих переменных не может быть определено, выполнение команды `pkgadd` прерывается, и выдается сообщение об ошибке.

В файле `prototype` определение переменной может принять форму `!`*PARAM=value* или `$`*variable*. *PARAM* и *variable* могут начинаться как с заглавной, так и со строчной буквы. Вычисляются значения только для переменных, значения которых известны в период сборки. Если *PARAM* или *variable* является переменной сборки или установки, значение которой неизвестно во время сборки пакета, работа команды `pkgmk` прерывается, и появляется сообщение об ошибке.

Параметр *PARAM=value* можно включить и в качестве параметра команды `pkgmk`. Этот параметр работает так же, как в файле `prototype`, за исключением того, что область его действия является глобальной для всего пакета. Определение `!`*PARAM=value* в файле `prototype` является локальным для этого файла и той части пакета, который оно определяет.

Если *PARAM*является переменной установки, а *variable* - переменной установки или сборки с известным значением, то команда `pkgmk` вставляет определение в файл `pkginfo`, и это определение будет доступно в период установки. Однако команда `pkgmk` не определяет значения *PARAM*, содержащиеся в именах путей, указанных в файле `prototype`.

Краткий обзор переменных среды пакета

В таблице ниже представлен краткий обзор форматов, местоположения и области применения переменных.

ТАБЛИЦА 2–2 Краткий обзор переменных среды пакета

Место определения переменной	Формат определения переменной	Тип определяемой переменной	Когда определяется значение переменной	Где определяется значение переменной	Элементы, которые переменная может заменить
Файл `pkginfo`	*PARAM=value*	Переменная сборки	Игнорируется во время сборки	нет	Нет
Установка	Во время установки	В файле `pkgmap`	*owner* (владелец), *group* (группа), *path* (путь) или цель ссылки		

ТАБЛИЦА 2–2 Краткий обзор переменных среды пакета *(Продолжение)*

Место определения переменной	Формат определения переменной	Тип определяемой переменной	Когда определяется значение переменной	Где определяется значение переменной	Элементы, которые переменная может заменить
Файл `prototype`	`!`*PARAM=value*	Переменная сборки	Во время сборки	В файле `prototype` и любых включенных файлах	*mode* (режим), *owner* (владелец), *group* (группа) или *path* (путь)
Установка	Во время сборки	В файле `prototype` и любых включенных файлах	Только в командах `!search` и `!command`		
Командная строка `pkgmk`	*PARAM=value*	Переменная сборки	Во время сборки	В файле `prototype`	*mode* (режим), *owner* (владелец), *group* (группа) или *path* (путь)
Установка	Во время сборки	В файле `prototype`	Только команда `!search`		
Во время установки	В файле `pkgmap`	*owner* (владелец), *group* (группа), *path* (путь) или link target (целевая ссылка)			

Создание файла `pkginfo`

Файл `pkginfo` представляет собой файл ASCII, в котором описаны характеристики пакета и даны сведения, которые помогают контролировать процесс установки.

Каждая запись в файле `pkginfo` представляет из себя строку, которая устанавливает значения параметра в формате *PARAM=value*. *PARAM* может быть любым стандартным параметром, описанным на справочной странице `pkginfo(4)` Не существует обязательного порядка, в котором следует указывать параметры.

Примечание – Каждое *значение value* может быть заключено в одинарные или двойные кавычки, например: '*value*' или “*value*”). Если *значение value* содержит символы, считающиеся специальными в интерпретаторе команд, необходимо использовать кавычки. В практических примерах, приведенных в данной книге, не используются кавычки. Пример использования двойных кавычек приведен на справочной странице `pkginfo(4)`.

Можно также создавать свои собственные параметры пакета, присвоив им значение в файле pkginfo. Ваши собственные параметры должны начинаться с заглавной буквы. Дальнейшие буквы могут быть как заглавными, так и строчными. Заглавная буква означает, что параметр (переменная) будет определен во время установки, а не во время сборки. Для получения дополнительной информации о разнице между переменными установки и переменными сборки см. раздел «Переменные среды пакета» на стр. 26.

Примечание – Конечные пробелы после любого значения параметра игнорируются.

Необходимо определить следующие пять параметров в файле pkginfo: PKG, NAME, ARCH, VERSION и CATEGORY. Параметры PATH, PKGINST и INSTDATE вставляются автоматически программным обеспечением во время сборки пакета. Не следует изменять эти восемь параметров. Для информации об остальных параметрах см. справочную страницу pkginfo(4).

Определение экземпляра пакета

Один и тот же пакет может иметь различные версии, быть совместимым с различными архитектурами или и то и другое одновременно. Каждая разновидность пакета называется *экземпляр пакета*. Экземпляр пакета определяется по сочетанию определений параметров PKG, ARCH и VERSION в файле pkginfo.

Команда pkgadd назначает *идентификатор пакета* каждому экземпляру пакета в процессе установки. Идентификатор пакета - это аббревиатура пакета с цифровым суффиксом, например SUNWadm.2. Данный идентификатор служит отличительным признаком экземпляра пакета, по которому его можно отличить от любых других пакетов, включая другие экземпляры этого же пакета.

Определение аббревиатуры пакета (PKG)

Аббревиатура пакета - это краткое название пакета, которое определяется параметром PKG в файле pkginfo. Аббревиатура пакета должна иметь следующие характеристики:

- Аббревиатура должна состоять из алфавитно-цифровых символов. Первый символ не может быть числом.
- Аббревиатура не должна быть длиннее 32 символов.
- Аббревиатура не должна совпадать с одной из зарезервированных аббревиатур: install, new или all.

Примечание – Первые четыре символа должны быть уникальными для компании. Например у всех пакетов, созданных компанией Sun Microsystems™, первые четыре символа аббревиатуры – “SUNW”.

В качестве примера сокращенной записи пакета в файле pkginfo можно привести PKG=SUNWcadap.

Указание архитектуры пакета (ARCH)

Параметр ARCH в файле pkginfo определяет, какие архитектуры связаны с данным пакетом. Имя архитектуры не должно превышать 16 алфавитно-цифровых символов. Если пакет связан с несколькими архитектурами, то все архитектуры необходимо перечислить в списке через запятую.

Пример спецификации архитектуры пакета в файле pkginfo :

```
ARCH=sparc
```

Указание архитектуры системы команд пакета (SUNW_ISA)

Параметр SUNW_ISA в файле pkginfo указывает, какая архитектура системы команд связана с пакетом Sun Microsystems. Может принимать следующие значения:

- sparcv9 для пакета, содержащего 64-разрядные объекты
- sparcv для пакета, содержащего 32-разрядные объекты

Например, значение SUNW_ISA в файле pkginfo для пакета, содержащего 64-разрядные объекты, будет:

```
SUNW_ISA=sparcv9
```

Если параметр SUNW_ISA не установлен, то по умолчанию архитектура системы команд пакета устанавливается на значение параметра ARCH.

Указание версии пакета (VERSION)

Параметр VERSION в файле pkginfo указывает версию пакета. Название версии не должно превышать 256 символов ASCII и не может начинаться с левой скобки.

Пример спецификации версии в файле pkginfo:

```
VERSION=release 1.0
```

Определение имени пакета (NAME)

Имя пакета - это полное название пакета, определяемое параметром NAME в файле pkginfo.

Поскольку системные администраторы часто используют имена пакетов для указания, следует ли устанавливать тот или иной пакет, написание четких и выразительных полных имен пакетов очень важно. Имена пакетов должны соответствовать следующим ниже критериям.

- Укажите, для чего нужен этот пакет (например, для обеспечения определенных команд или функциональности или укажите, что пакет необходим для конкретного оборудования).
- Укажите, для чего используется этот пакет (например, для разработки драйверов устройств).
- Включите описание аббревиатуры пакета с помощью ключевых слов, кратко расшифровывающих аббревиатуру. Например, полное имя пакета для аббревиатуры SUNWbnuu – "Basic Networking UUCP Utilities, (Usr)" .
- Укажите раздел диска, на который будет устанавливаться пакет.
- Используйте термины в соответствии с их значением, принятым в отрасли.
- Используйте возможность написания 256–символьного имени.

Пример имени пакета, определенного в файле pkginfo:

```
NAME=Chip designers need CAD application software to design
abc chips.  Runs only on xyz hardware and is installed in the
usr partition.
```

Определение категории пакета (CATEGORY)

Параметр CATEGORY в файле pkginfo указывает, к какой категории принадлежит пакет. Пакет должен принадлежать как минимум к одной из следующих категорий: system (система) или application (приложение). Имена категорий обозначаются алфавитно-цифровыми символами. Имена категорий не зависят от регистра и не должны превышать 16 символов в длину.

Если пакет принадлежит к нескольким категориям, укажите эти категории в списке через запятую.

Пример спецификации CATEGORY в файле pkginfo :

```
CATEGORY=system
```

▼ Как создать файл pkginfo

1 **С помощью своего любимого текстового редактора создайте файл с именем** `pkginfo`**.**

Этот файл может быть создан в любом месте системы.

2 **Откройте файл для правки и определите пять обязательных параметров.**

Этими пятью параметрами являются: `PKG`, `NAME`, `ARCH`, `VERSION` и `CATEGORY`. Для получения дополнительной информации об этих параметрах ознакомьтесь с разделом «Создание файла `pkginfo`» на стр. 28.

3 **Добавьте к файлу любые необязательные параметры.**

Создайте свои собственные параметры или прочтите справочную страницу `pkginfo(4)` для получения сведений о стандартных параметрах.

4 **Сохраните изменения и выйдите из редактора.**

Пример 2–1 Создание файла `pkginfo`

В данном примере показано содержимое допустимого файла `pkginfo` с пятью обязательными параметрами, а также с параметром `BASEDIR`. Параметр `BASEDIR` более подробно обсуждается в разделе «Поле *path*» на стр. 36.

```
PKG=SUNWcadap
NAME=Chip designers need CAD application software to design abc chips.
Runs only on xyz hardware and is installed in the usr partition.
ARCH=sparc
VERSION=release 1.0
CATEGORY=system
BASEDIR=/opt
```

См. также См. раздел «Как упорядочить содержимое пакета» на стр. 33.

Упорядочение содержимого пакета

Упорядочите объекты пакета в иерархическую структуру каталога, которая будет повторять структуру объектов пакета в целевой системе после установки. Если вы выполните эту процедуру до создания файла `prototype`, вы сэкономите время и усилия при создании этого файла.

▼ Как упорядочить содержимое пакета

1 **Определите, сколько пакетов вам нужно создать и какие объекты будут находиться в каждом пакете.**

Для получения справки о выполнении этого действия см. раздел «Что следует принять во внимание перед сборкой пакета» на стр. 17.

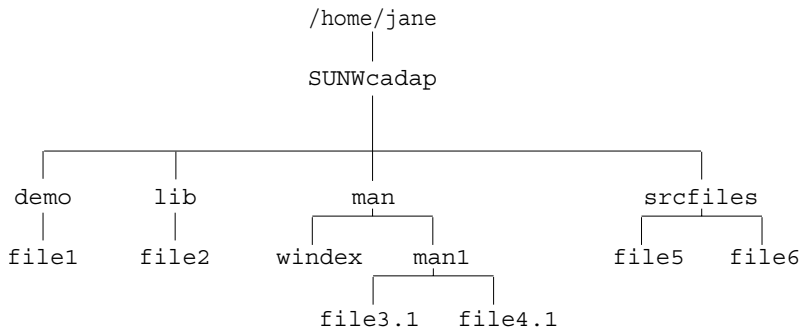
2 **Создайте каталог для каждого пакета, который необходимо собрать.**

Вы можете создать этот каталог в любом месте своей системы и назвать его по своему усмотрению. В примерах к этой главе предполагается, что каталог пакета имеет то же имя, что и аббревиатура пакета.

```
$ cd /home/jane
$ mkdir SUNWcadap
```

3 **Упорядочите объекты пакета в каждом пакете в структуру каталогов под соответствующим каталогом пакета. Структура каталога должна повторять структуру объектов пакета в целевой системе после установки.**

Например, для пакета приложений САПР `SUNWcadap` требуется следующая структура каталога.



4 **Определитесь, где будет храниться информационные файлы. Если целесообразно, создайте каталог для хранения этих файлов в одном месте.**

В данном примере предполагается, что файл `pkginfo` из раздела «Как создать файл `pkginfo`» на стр. 32 был создан в домашнем каталоге Jane.

```
$ cd /home/jane
$ mkdir InfoFiles
$ mv pkginfo InfoFiles
```

См. также См. раздел «Создание файла `prototype` с помощью команды `pkgproto`» на стр. 46.

Создание файла prototype

Файл prototype - это файл в кодировке ASCII, используемый для указания информации об объектах пакета. Каждая запись в файле prototype описывает объект, например файл данных, каталог, исходный файл или исполняемый объект. Записи в файле prototype состоят из нескольких информационных полей, разделенных пробелами. Обратите внимание на то, что поля *должны* следовать в определенном порядке. Строки с комментариями начинаются со знака диеза (#) и игнорируются.

Создать файлprototype можно с помощью текстового редактора или команды pkgproto. При создании этого файла в первый раз, вероятно, проще это сделать с помощью команды pkgproto, потому что она создает файл на основе ранее созданной иерархии каталогов. Если вы не упорядочили файлы в соответствии с описанием, приведенном в разделе «Упорядочение содержимого пакета» на стр. 32, перед вами стоит нелегкая задача создания файла prototype с нуля в своем любимом текстовом редакторе.

Формат файла prototype

Ниже приведен пример формата каждой строки в файле prototype:

partftypeclasspathmajorminormodeownergroup

part	Необязательное числовое поле, позволяющее сгруппировать объекты пакета в части. Значение по умолчанию - 1.
ftype	Поле, состоящее из одного символа, указывающего на тип объекта. См. раздел «Поле ftype» на стр. 35.
class	Класс установки, к которому принадлежит объект. См. раздел «Поле *class*» на стр. 35.
path	Абсолютное или относительное имя пути, указывающее, в каком месте целевой системы будет находиться объект пакета. См. раздел «Поле *path*» на стр. 36.
major	Старший номер устройства для специальных блочных или символьных устройств.
minor	Младший номер устройства для специальных блочных или символьных устройств.
mode	Восьмеричный режим объекта (например, 0644). См. раздел «Поле *mode*» на стр. 39.
owner	Владелец объекта (например, bin или root). См. раздел «Поле *owner*» на стр. 39.

group	Группа, к которой принадлежит объект (например, bin или sys). См. раздел «Поле *group*» на стр. 40.

Обычно определяются только поля *ftype*, *class*, *path*, *mode*, *owner*,и *group*. Эти поля описаны в следующих разделах. Для получения дополнительной информации по этим полям см. справочную страницу prototype(4).

Поле ftype

Поле *ftype* или тип файла является односимвольным полем, обозначающим тип файла объекта пакета. Допустимые типы файлов представлены в следующей ниже таблице.

ТАБЛИЦА 2–3 Допустимые типы файлов в файле prototype

Допустимое значение типа файла	Описание типа файла
f	Стандартный исполняемый файл или файл данных
e	Файл, который изменяется во время установки или удаления (может быть общим для нескольких пакетов)
v	Непостоянный файл (содержимое которого будет меняться, например, файл журнала)
d	Каталог
x	Монопольный каталог, доступный только этому пакету (может содержать незарегистрированные журналы или информацию базы данных)
l	Ссылочный файл
p	Именованный канал
c	Специальное символьное устройство
b	Специальное блочное устройство
i	Информационный файл или сценарий установки
s	Символическая ссылка

Поле *class*

Поле *class* именует класс, к которому принадлежит объект. Использование классов является необязательной функцией для разработки пакета. Более подробно эта функция рассматривается в разделе «Создание сценариев действий над классами» на стр. 75.

Если классы не используются, объект принадлежит к классу none. При выполнении команды pkgmk для сборки пакета команда вставляет параметр CLASSES=none в файл pkginfo. У файлов типа i поле *class* должно быть пустым.

Поле *path*

Поле *path* используется для определения местонахождения объекта пакета в целевой системе. Указать местоположение файла можно с помощью абсолютного (например, `/usr/bin/mail`) или относительного имени пути (например, `bin/mail`). Использование абсолютного имени пути означает, что местоположение объекта в целевой системе определяется пакетом и не может быть изменено. Объекты пакета с относительными именами пути являются *перемещаемыми*.

Перемещаемому объекту не требуется абсолютного пути в целевой системе. Вместо этого его местоположение определяется в ходе процесса установки.

Перемещаемыми могут быть отдельные объекты или все объекты пакета. Перед написанием сценария установки или созданием файла `prototype` необходимо решить, будут ли объекты пакета иметь фиксированное местоположение (например, сценарии запуска в каталоге `/etc`) или перемещаемое .

Существуют два типа перемещаемых объектов: *коллективно перемещаемые* и *индивидуально перемещаемые*.

Коллективно перемещаемые объекты

Коллективно перемещаемые объекты расположены в месте, связанном с общей установочной базой, называемой *базовым каталогом* . Базовый каталог определяется в файле `pkginfo` с помощью параметра `BASEDIR`. Например, перемещаемый объект `tests/generic` в файле `prototype` требует, чтобы в файле `pkginfo` по умолчанию был установлен параметр`BASEDIR`. Пример:

```
BASEDIR=/opt
```

В данном примере после установки объекта он будет расположен в каталоге `/opt/tests/generic`.

Примечание – Единственным каталогом, куда можно устанавливать программное обеспечение, не являющееся базовым ПО Solaris, это каталог `/opt`.

По возможности старайтесь использовать коллективно перемещаемые объекты. В общем случае основная часть пакета может быть перемещаемой, а несколько файлов (например, файлы в каталогах `/etc` или `/var`) могут быть указаны как абсолютные. Однако, если пакет содержит множество различных перемещений, рекомендуется разделить пакет на несколько пакетов с явно указанными значениями `BASEDIR` в соответствующих файлах `pkginfo`.

Индивидуально перемещаемые объекты

Индивидуально перемещаемые объекты, в отличие от коллективно перемещаемых, не ограничены размещением в одном и том же каталоге. Для определения индивидуально перемещаемого объекта необходимо указать установочную переменную в поле *path* файла `prototype`. После определения установочной переменной нужно создать сценарий `request`, запрашивающий установщик о расположении перемещаемого базового каталога, или сценарий `checkinstall` для получения имени пути из данных файловой системы. Для получения дополнительной информации по сценарию `request` см. раздел «Создание сценария `request` » на стр. 68, а информация по сценарию `checkinstall` находится в разделе «Сбор данных о файловой системе» на стр. 72.

Внимание – Управлять индивидуально перемещаемыми объектами сложно. Использование индивидуально перемещаемых объектов может привести к большому разнесению компонентов пакета, которые будет сложно изолировать при установке нескольких версий или архитектур пакета. По возможности старайтесь использовать коллективно перемещаемые объекты.

Параметрические имена путей

Параметрическое имя пути - это имя пути, содержащее спецификацию переменной. Например, `/opt/$PKGINST/`*filename* является параметрическим именем пути, поскольку включает спецификацию переменной `$PKGINST`. Значение спецификации переменной по умолчанию *должно* быть определено в файле `pkginfo`. Позднее это значение может быть изменено сценариями `request` или `checkinstall`.

Спецификация переменной должна стоять в начале или в конце имени пути или должна быть заключена между символами наклонной черты (/). Допускаются следующие формы параметрических имен пути.

```
$PARAM/tests
tests/$PARAM/generic
/tests/$PARAM
```

После того как спецификация переменной определена, путь может быть вычислен как абсолютный или перемещаемый. В примере ниже файл `prototype` содержит следующую запись:

```
f none $DIRLOC/tests/generic
```

Файл `pkginfo` содержит следующую запись:

```
DIRLOC=/myopt
```

Имя пути $DIRLOC/tests/generic приобретает вид абсолютного имени пути /myopt/tests/generic независимо от того, был ли установлен параметр BASEDIR в файле pkginfo.

В данном примере файл prototype идентичен одному из предыдущих примеров, а в файле pkginfo содержатся следующие записи:

```
DIRLOC=firstcut
BASEDIR=/opt
```

Имя пути $DIRLOC/tests/generic будет выражено в виде перемещаемого имени пути /opt/firstcut/tests/generic.

Для получения дополнительной информации о параметрических именах путей см. раздел «Использование параметрических базовых каталогов» на стр. 140.

Краткие замечания о расположении каталогов источника и приемника объекта

Поле *path* в файле prototype определяет расположение объекта в целевой системе. Если структура каталога не повторяет структуру, которую необходимо получить в целевой системе, то следует указать существующее местоположение объектов пакета в файле prototype. Для получения дополнительной информации о структурировании объектов в пакете см. раздел «Упорядочение содержимого пакета» на стр. 32.

Если структура области разработки отличается от желаемой структуры пакета, то можно использовать формат *path1=path2* в поле *path*. В этом формате *path1* означает расположение объекта в целевой системе, а *path2* - расположение объектов в системе разработчика.

Этот же формат имени пути (*path1=path2*) можно использовать следующим образом: определить *path1* в качестве перемещаемого имени объекта , *a path2* - в качестве полного имени пути к этому объекту в системе разработчика.

Примечание – *path1* не должен содержать неопределенных переменных сборки, но может содержать неопределенные переменные установки. *path2* не может содержать никаких неопределенных переменных, однако можно использовать переменные обоих типов. Для получения дополнительной информации о разнице между переменными установки, и переменными, создаваемыми в процессе сборки пакета, см. раздел «Переменные среды пакета» на стр. 26.

Для ссылок должен использоваться формат *path1= path2*, поскольку они создаются с помощью команды pkgadd. В общем случае путь *path2* в ссылках никогда не должен быть абсолютным. Данный путь должен указываться относительно части пути *path1*, обозначающей каталог.

Вместо формата *path1=path2* можно использовать команду `!search`. Для получения дополнительной информации см. раздел «Указание пути поиска для команды pkgmk.» на стр. 45.

Поле *mode*

Поле *mode* может содержать восьмеричное число, знак вопроса (`?`) или спецификацию переменной. Восьмеричное число определяет режим объекта после его установки на целевую систему. Знак вопроса `?` означает, что при установке объекта режим останется неизменным. Подразумевается, что объект с тем же именем уже существует в целевой системе.

Спецификация переменной в форме $*mode* означает, что данное поле будет установлено во время сборки пакета. Первая буква переменной должна быть строчной. Обратите внимание, что эта переменная должна быть определена во время сборки пакета либо в файле `prototype`, либо в качестве параметра команды `pkgmk`. Для получения дополнительной информации о разнице между переменными установки и переменными сборки пакета см. раздел «Переменные среды пакета» на стр. 26.

У файлов типа `i` (информационные файлы), `l` (жесткая ссылка) и `s` (символическая ссылка) это поле должно быть пустым.

Поле *owner*

Поле *owner* может содержать имя пользователя, знак вопроса (`?`) или спецификацию переменной. Имя пользователя не должно быть длиннее 14 символов и должно соответствовать имени, уже существующему в целевой системе (например, `bin` или `root`). Знак вопроса `?` означает, что при установке объекта владелец (owner) останется неизменным. Подразумевается, что объект с тем же именем уже существует в целевой системе.

Спецификация переменной может быть указана в форме $ *Owner* или $*owner*, т.е. первая буква может быть как заглавной, так и строчной. Если переменная начинается со строчной буквы, то ее необходимо определить в период сборки пакета в файле `prototype` или в качестве параметра команды `pkgmk`. Если переменная начинается с заглавной буквы, то спецификация переменной будет внесена в файл `pkginfo` в качестве значения по умолчанию и может быть переопределена в ходе установки с помощью сценария `request`. Для получения дополнительной информации о разнице между переменными, создаваемыми в процессе установки, и переменными, создаваемыми в процессе сборки пакета, см. раздел «Переменные среды пакета» на стр. 26.

У файлов типа `i` (информационный файл) и `lb` (жесткая ссылка) это поле должно быть пустым.

Поле *group*

Поле *group* может содержать имя группы, знак вопроса (?) или спецификацию переменной. Имя группы не должно быть длиннее 14 символов и должно соответствовать имени, уже существующему в целевой системе (например, bin или sys). Знак вопроса ? означает, что при установке объекта группа (group) останется неизменной. Подразумевается, что объект с тем же именем уже существует в целевой системе.

Спецификация переменной может быть указана в форме $ *Group* или $*group*, т.е. первая буква переменной может быть как заглавной, так и строчной. Если переменная начинается со строчной буквы, то ее необходимо определить в период сборки пакета в файле prototype или в качестве параметра команды pkgmk. Если переменная начинается с заглавной буквы, то спецификация переменной будет внесена в файл pkginfo в качестве значения по умолчанию и может быть переопределена в ходе установки с помощью сценария request. Для получения дополнительной информации о разнице между переменными установки и переменными сборки пакета см. раздел «Переменные среды пакета» на стр. 26.

У файлов типа i (информационный файл) и l (жесткая ссылка) это поле должно быть пустым.

Создание файла prototype с нуля

При необходимости создания файла prototype с нуля можно использовать свой любимый текстовый редактор и добавлять по одной записи для каждого объекта пакета. Для получения дополнительной информации по формату этого файла см. раздел «Формат файла prototype» на стр. 34 и справочную страницу prototype(4). После определения каждого объекта пакета можно включить в файл некоторые функции, описанные в разделе «Добавление функций в файл prototype» на стр. 43.

Пример. Создание файла prototype с помощью команды pkgproto

С помощью команды pkgproto можно создать базовый файл prototype при условии, что структура каталогов пакета была упорядочена в соответствии с процедурой, описанной в разделе «Упорядочение содержимого пакета» на стр. 32. Например, с помощью используемой в примерах структуре каталогов и файла pkginfo, описание которого дано в предыдущих разделах, команды для создания файла prototype будут выглядеть следующим образом:

```
$ cd /home/jane
$ pkgproto ./SUNWcadap > InfoFiles/prototype
```

В результате получится следующий файл prototype:

```
d none SUNWcadap 0755 jane staff
d none SUNWcadap/demo 0755 jane staff
f none SUNWcadap/demo/file1 0555 jane staff
d none SUNWcadap/srcfiles 0755 jane staff
f none SUNWcadap/srcfiles/file5 0555 jane staff
f none SUNWcadap/srcfiles/file6 0555 jane staff
d none SUNWcadap/lib 0755 jane staff
f none SUNWcadap/lib/file2 0644 jane staff
d none SUNWcadap/man 0755 jane staff
f none SUNWcadap/man/windex 0644 jane staff
d none SUNWcadap/man/man1 0755 jane staff
f none SUNWcadap/man/man1/file4.1 0444 jane staff
f none SUNWcadap/man/man1/file3.1 0444 jane staff
```

Примечание – Действительный владелец и группа, в которую входит сборщик пакета, записываются командой pkgproto. Полезным приемом считается использование команд chown -R и chgrp -R, указывающих владельца и группу, *до* выполнения команды pkgproto.

В приведенном выше примере создание файла prototype не закончено. В следующем разделе приводится информация по завершению создания этого файла.

Настройка файла prototype, созданного с помощью команды pkgproto

Несмотря на то, что команда pkgproto очень полезна при создании первичного файла prototype, она не создает записей для всех объектов пакета, которые должны быть определены. Эта команда не создает полных записей. Команда pkgproto не делает ничего из перечисленного ниже.

- Не создает полных записей для объектов типа v (непостоянные файлы), e (редактируемые файлы), x (монопольные каталоги) и i (информационные файлы или сценарии установки)
- Не поддерживает множественные классы с единым вызовом

Содание записей для объектов типа v, e, x и i

Как минимум необходимо изменить файл prototype и добавить в него объекты с типом файла i. Если информационные файлы и сценарии установки были сохранены на первом уровне каталога пакета (например, /home/jane/SUNWcadap/pkginfo), тогда запись в файле prototype будет выглядеть следующим образом:

```
i pkginfo
```

Если же они не были сохранены на первом уровне каталога пакета, то потребуется указать из исходное местоположение. Пример:

```
i pkginfo=/home/jane/InfoFiles/pkginfo
```

В качестве альтернативы можно использовать команду `!search` для указания местоположения, которое команда `pkgmk` будет искать при сборке пакета. Для получения дополнительной информации см. раздел «Указание пути поиска для команды pkgmk.» на стр. 45.

Для добавления записей для объектов типа `v`, `e` и `x` используйте формат, описанный в разделе «Формат файла prototype» на стр. 34, или воспользуйтесь справочной страницей `prototype(4)`.

Примечание – Не забывайте всегда назначать класс для файлов типа `e` (редактируемые) и создавать для него связанный сценарий действия над классом. В противном случае файлы будут удалены в ходе удаления пакета, даже если указанное имя пути совместно используется другими пакетами.

Использование нескольких классов

При использовании команды `pkgproto` для создания базового файла `prototype` можно назначить все объекты пакета классу `none` или одному определенному классу. Как показано в разделе «Пример. Создание файла prototypeс помощью команды pkgproto» на стр. 40, базовая команда `pkgproto` назначает все объекты классу `none`. Чтобы назначить все объекты какому-либо определенному классу, можно использовать параметр `-c`. Пример:

```
$ pkgproto -c classname /home/jane/SUNWcadap > /home/jane/InfoFiles/prototype
```

При использовании нескольких классов придется изменить файл `prototype` вручную и отредактировать поле *class* для каждого объекта. При использовании классов необходимо также определить параметр `CLASSES` в файле `pkginfo`, а также написать сценарии действий над классами. Использование классов является необязательным и подробно обсуждается в разделе «Создание сценариев действий над классами» на стр. 75.

Пример. Настройка файла `prototype`, созданного с помощью команды `pkgproto`

При наличии файла `prototype`, созданного ранее при помощи команды `pkgproto`, как описано в разделе «Пример. Создание файла prototypeс помощью команды pkgproto» на стр. 40, в нем требуется произвести некоторые изменения.

- Необходима запись для файла pkginfo.
- Поле *path* необходимо перевести в формат *path1=path2*, поскольку источник пакета находится в каталоге /home/jane . Так как исходный пакет представляет собой иерархический каталог, а команда !search не может осуществлять поиск рекурсивно, проще использовать формат *path1=path2*.
- Поля *owner* и *group* должны содержать имена существующих в целевой системе пользователей и групп. Следовательно, при указании владельца jane возникнет ошибка, так как данный владелец не является частью операционной системы SunOS™.

  Измененный файл prototype будет выглядеть следующим образом:

```
i pkginfo=/home/jane/InfoFiles/pkginfo
d none SUNWcadap=/home/jane/SUNWcadap 0755 root sys
d none SUNWcadap/demo=/home/jane/SUNWcadap/demo 0755 root bin
f none SUNWcadap/demo/file1=/home/jane/SUNWcadap/demo/file1 0555 root bin
d none SUNWcadap/srcfiles=/home/jane/SUNWcadap/srcfiles 0755 root bin
f none SUNWcadap/srcfiles/file5=/home/jane/SUNWcadap/srcfiles/file5 0555 root bin
f none SUNWcadap/srcfiles/file6=/home/jane/SUNWcadap/srcfiles/file6 0555 root bin
d none SUNWcadap/lib=/home/jane/SUNWcadap/lib 0755 root bin
f none SUNWcadap/lib/file2=/home/jane/SUNWcadap/lib/file2 0644 root bin
d none SUNWcadap/man=/home/jane/SUNWcadap/man 0755 bin bin
f none SUNWcadap/man/windex=/home/jane/SUNWcadap/man/windex 0644 root other
d none SUNWcadap/man/man1=/home/jane/SUNWcadap/man/man1 0755 bin bin
f none SUNWcadap/man/man1/file4.1=/home/jane/SUNWcadap/man/man1/file4.1 0444 bin bin
f none SUNWcadap/man/man1/file3.1=/home/jane/SUNWcadap/man/man1/file3.1 0444 bin bin
```

Добавление функций в файл prototype

Помимо определения каждого объекта пакета в файле prototype, можно расширить его функциональность следующим образом.

- Определить дополнительные объекты, которые будут создаваться во время установки пакета.
- Создать ссылки во время установки.
- Распределить пакеты по нескольким томам.
- Организовать вложенные файлы prototype.
- Установить значения по умолчанию для полей *mode*, *owner* и *group*.
- Указать путь поиска для командыpkgmk.
- Установить переменные среды.

В представленных ниже разделах содержится информация о том, как произвести указанные изменения.

Определение дополнительных объектов, которые будут создаваться во время установки

С помощью файла prototype можно определять объекты, не поставляемые на распространяемом носителе. Если во время установки объекты требуемого типа не существуют, команда pkgadd создает эти объекты.

Для того чтобы указать, что объект будет создаваться на целевой системе, добавьте для него запись в файле prototype и укажите соответствующий тип файла.

Например, если на целевой системе необходимо создать каталог, отсутствующий на распространяемом носителе, следует добавить следующую запись в файл prototype:

```
d none /directory 0644 root other
```

Для создания пустого файла на целевой системе запись для такого файла в файле prototype может принять следующий вид:

```
f none filename=/dev/null 0644 bin bin
```

Единственными объектами, которые *должны* поставляться на распространяемом носителе, являются обычные файлы и сценарии редактирования (файлы типов e, v, f), а также каталоги, в которых они содержатся. Любые дополнительные объекты создаются без ссылки на поставляемые объекты, каталоги, именованные каналы, устройства, жесткие и символические ссылки.

Создание ссылок во время установки

Для создания ссылок во время установки пакета в записи файла prototype для ссылочного объекта укажите следующее:

- Его тип в виде l (ссылка) или s (символическая ссылка).
- Имя пути ссылочного объекта в формате *path1=path2*, где *path1* - файл назначения, а *path2* - исходный файл. В общем случае путь *path2* в ссылках никогда не должен быть абсолютным. Данный путь должен указываться относительно части пути *path1*, обозначающей каталог. Например, запись в файле prototype, определяющая символическую ссылку, может выглядеть следующим образом:

  ```
  s none etc/mount=../usr/etc/mount
  ```

Создание относительных ссылок будет зависеть от способа установки пакета, то есть будет ли он установлен как абсолютный или перемещаемый пакет.

Распределение пакетов по нескольким томам

При сборке пакета с помощью команды pkgmk эта команда производит расчеты и действия, необходимые для упорядочения многотомного пакета. Многотомный пакет называется *сегментированным пакетом*.

Существует и другой способ указать, в какой части будет располагаться объект. Для этого следует воспользоваться необязательном полем *part* в файле `prototype`. Номер, указанный в этом поле, отменяет команду `pkgmk` и помещает компонент в часть, указанную в этом поле. Обратите внимание на то, что существует взаимно-однозначное соответствие между частями и томами для сменных носителей, отформатированных как файловые системы. Если тома были назначены разработчиком, при недостаточном количестве места на любом из томов команда `pkgmk` выдает ошибку.

Вложение файлов `prototype`

Можно создать несколько файлов `prototype`, а затем включить их при помощи команды `!include` в один файл `prototype`. Вложение файлов может потребоваться для облегчения поддержки.

В следующем примере присутствуют три файла `prototype`. Редактируется главный файл (`prototype`). Два других файла (`proto2` и `proto3`) включаются в этот файл.

```
!include /source-dir/proto2
!include /source-dir/proto3
```

Установка значений по умолчанию для полей *mode*, *owner* и *group*.

Чтобы установить значения по умолчанию для полей *mode*, *owner* и *group* отдельных объектов пакета, необходимо вставить команду `!default` в файл `prototype`. Пример:

```
!default 0644 root other
```

Примечание – Действие команды `!default` начинается с места ее включения в файл и продолжается до его конца. Действие команды не распространяется на вложенные файлы.

Следует иметь в виду, что для каталогов (тип файла `d`) и редактируемых файлов (тип файла `e`), имеющихся в целевой системе (таких как `/usr` или `/etc/vfstab`), необходимо установить в значениях полей *mode*, *owner* и *group* в файле `prototype` вопросительный знак (`?`). Таким образом, не будут нарушены существующие параметры, измененные администратором сайта.

Указание пути поиска для команды `pkgmk`.

Если исходное расположение объектов пакета отличается от из расположения в целевой системе, и использование формата *path1=path2*, описанного в разделе «Краткие замечания о расположении каталогов источника и приемника объекта» на стр. 38, нежелательно, то можно использовать команду `!search` в файле `prototype`.

Например, если в домашнем каталоге был создан каталог pkgfiles, содержащий все информационные файлы и сценарии установки, то можно указать, что необходим поиск этого каталога при сборке пакета с помощью команды pkgmk.

Команда в файле prototype будет выглядеть следующим образом:

```
!search /home-dir/pkgfiles
```

Примечание – Запрос на поиск не распространяется на вложенные файлы. Кроме того, поиск ограничен определенными перечисленными каталогами. Рекурсивный поиск также не производится.

Установка переменных среды

К файлу prototype можно добавлять команды и в следующей форме: !*PARAM=value*. Команды, представленные в такой форме, определяют переменные в существующей среде. При наличии нескольких файлов prototype действие этой команды распространяется на тот файл prototype , в котором она была определена.

Переменная *PARAM* может начинаться с заглавной или со строчной буквы. Если значение переменной *PARAM* неизвестно во время сборки пакета, то выполнение команды pkgmk прерывается, и выдается сообщение об ошибке. Для получения дополнительной информации о разнице между переменными установки и переменными сборки пакета см. раздел «Переменные среды пакета» на стр. 26.

▼ Создание файла prototype с помощью команды pkgproto

Примечание – Рекомендуется создавать информационные файлы и сценарии установки до создания файла prototype. Однако этот порядок не является обязательным. Файл prototype всегда можно отредактировать после изменения содержимого пакета. Для получения дополнительных данных об информационных файлах и сценариях установки см. Глава 3, «Расширение функциональности пакета (задачи)».

1 **Определите, какие объекты пакета будут абсолютными, а какие - перемещаемыми.**

Для получения информации о выполнении этого этапа см. раздел «Поле *path*» на стр. 36.

2 **Упорядочите объекты пакета таким образом, чтобы они повторяли свое расположение в целевой системе.**

Если пакеты уже упорядочены так, как описано в разделе «Упорядочение содержимого пакета» на стр. 32, возможно, придется внести в них некоторые изменения в зависимости

от того, какое решение было принято в Шаг 1. Если пакет еще не упорядочен, следует сделать это сейчас. Если пакет не будет упорядочен, использование команды pkgproto для создания файла prototype будет невозможно.

3 **Если пакет содержит коллективно перемещаемые объекты, необходимо отредактировать файл** pkginfo **и присвоить параметру** BASEDIR **соответствующее значение.**

Пример:

```
BASEDIR=/opt
```

Для получения информации о коллективно перемещаемых объектах см. раздел «Коллективно перемещаемые объекты» на стр. 36.

4 **Если пакет содержит индивидуально перемещаемые объекты, необходимо создать сценарий** request**, который будет предлагать установщику ввести соответствующее имя пути. Как вариант, можно создать сценарий** checkinstall**, который определит соответствующий путь на основе системных данных.**

Ниже приведены ссылки на наиболее общие задачи.

- Для создания сценария request см. раздел «Создание сценария request » на стр. 69.
- Для создания сценария checkinstall см. раздел «Сбор данных о файловой системе» на стр. 72.
- Для получения дополнительной информации об индивидуально перемещаемых объектах см. раздел «Индивидуально перемещаемые объекты» на стр. 37.

5 **Измените владельца и группу во всех компонентах пакета таким образом, чтобы они соответствовали владельцу и группе в целевой системе.**

Используйте команды chown -R и chgrp -R в каталоге пакета и каталоге информационных файлов.

6 **Выполните команду** pkgproto **для создания базового файла** prototype**.**

Команда pkgproto сканирует каталоги и создает базовый файл. Пример:

```
$ cd package-directory
$ pkgproto ./package-directory > prototype
```

Файл prototype может располагаться в любом месте системы. Для упрощения доступа и обслуживания информационных файлов и сценариев установки рекомендуется хранить их в одном месте. Для получения дополнительной информации о команде pkgproto см. справочную страницу pkgproto(1).

7 **Откройте файл `prototype` в своем любимом текстовом редакторе и добавьте записи для файлов типа `v`, `e`, `x` и `i`.**

Для получения информации об изменениях, которые, вероятно, придется сделать, см. раздел «Настройка файла `prototype`, созданного с помощью команды `pkgproto`» на стр. 41.

8 **(Необязательное действие) При использовании нескольких классов отредактируйте файлы `prototype` и `pkginfo`. Используйте текстовый редактор для производства необходимых изменений и создания соответствующих сценариев действий над классами.**

Для получения информации об изменениях, которые вероятно придется сделать, см. раздел «Настройка файла `prototype`, созданного с помощью команды `pkgproto`» на стр. 41 и «Создание сценариев действий над классами» на стр. 75.

9 **Откройте файл `prototype` в текстовом редакторе и переопределите имена путей, а также измените другие настройки полей.**

Для получения дополнительной информации см. раздел «Настройка файла `prototype`, созданного с помощью команды `pkgproto`» на стр. 41.

10 **(Необязательно) Откройте файл `prototype` в любом удобном текстовом редакторе и добавьте дополнительные функции к файлу `prototype`.**

Для получения дополнительной информации см. раздел «Добавление функций в файл `prototype`» на стр. 43.

11 **Сохраните изменения и выйдите из редактора.**

См. также Для перехода к следующей задаче перейдите по ссылке «Как собрать пакет» на стр. 49.

Сборка пакета

Для сборки пакета используйте команду `pkgmk`. Команда `pkgmk` выполняет следующие задачи:

- Переводит все объекты, определенные в файле `prototype`, в формат каталога.
- Создает файл `pkgmap`, заменяющий файл `prototype`.
- Создает устанавливаемый пакет, который используется в качестве входных данных команды `pkgadd`.

Использование простейшей команды `pkgmk`

Простейшей формой этой команды является выполнение команды `pkgmk` без каких-либо параметров. Перед использованием команды `pkgmk` без параметров убедитесь, что в

текущем рабочем каталоге содержится файл prototype пакета. Результат выполнения команды, файлы и каталоги записываются в каталог /var/spool/pkg.

Файл pkgmap

При сборке пакета с помощью команды pkgmk она создает файл pkgmap, который заменяет файл prototype. Файл pkgmap из предыдущего примера имеет следующее содержимое:

```
$ more pkgmap
: 1 3170
1 d none SUNWcadap 0755 root sys
1 d none SUNWcadap/demo 0755 root bin
1 f none SUNWcadap/demo/file1 0555 root bin 14868 45617 837527496
1 d none SUNWcadap/lib 0755 root bin
1 f none SUNWcadap/lib/file2 0644 root bin 1551792 62372 837527499
1 d none SUNWcadap/man 0755 bin bin
1 d none SUNWcadap/man/man1 0755 bin bin
1 f none SUNWcadap/man/man1/file3.1 0444 bin bin 3700 42989 837527500
1 f none SUNWcadap/man/man1/file4.1 0444 bin bin 1338 44010 837527499
1 f none SUNWcadap/man/windex 0644 root other 157 13275 837527499
1 d none SUNWcadap/srcfiles 0755 root bin
1 f none SUNWcadap/srcfiles/file5 0555 root bin 12208 20280 837527497
1 f none SUNWcadap/srcfiles/file6 0555 root bin 12256 63236 837527497
1 i pkginfo 140 10941 837531104
$
```

Формат этого файла очень схож с форматом файла prototype. Однако файл pkgmap содержит следующую информацию:

- В первой строке указывается количество томов, на которых расположен пакет, и примерный размер пакета после его установки.

  Например, `: 1 3170` означает, что пакет содержится на одном томе и будет занимать примерно 3170 512-байтовых блоков после установки.
- Имеются три дополнительных поля, в которых указываются размер, контрольная сумма и время изменения каждого объекта пакета.
- Объекты пакета упорядочены в алфавитном порядке по классу и по имени пути, чтобы сократить время установки пакета.

▼ Как собрать пакет

1 **Создайте файл pkginfo, если таковой отсутствует.**

Поэтапные указания приведены в разделе «Как создать файл pkginfo» на стр. 32.

2 Создайте файл `prototype`, если таковой отсутствует.

Поэтапные указания приведены в разделе «Создание файла `prototype` с помощью команды `pkgproto`» на стр. 46.

3 Сделайте каталог, содержащий файл `prototype`, текущим рабочим каталогом.

4 Выполните сборку пакета.

```
$ pkgmk [-o] [-a arch] [-b base-src-dir] [-d device]
   [-f filename] [-l limit] [-p pstamp] [-r rootpath]
   [-v version] [PARAM=value] [pkginst]
```

`-o`	Перезаписывает текущую версию пакета.
`-a` *arch*	Отменяет информацию об архитектуре, содержащуюся в файле `pkginfo`.
`-b` *base-src-dir*	Запрашивает добавление параметра *base-src-dir* к началу перемещаемых имен путей, когда команда `pkgmk` осуществляет поиск объектов в системе разработчика.
`-d` *device*	Указывает, что пакет должен быть скопирован на устройство *device*. В качестве устройства может выступать абсолютное имя пути каталога, дискета или съемный диск.
`-f` *filename*	Называет файл *filename*, который используется в качестве файла `prototype`. Имена по умолчанию - `prototype` или `Prototype`.
`-l` *limit*	Определяет максимальный размер устройства вывода в 512-байтных блоках.
`-p` *pstamp*	Отменяет определение производственной маркировки в файле `pkginfo`.
`-r` *rootpath*	Требует, чтобы для обнаружения объектов в системе разработчика использовался корневой каталог *rootpath*.
`-v` *version*	Отменяет информацию о версии, содержащуюся в файле `pkginfo`.
PARAM=value	Устанавливает глобальные переменные среды. Переменные, начинающиеся со строчной буквы, вычисляются во время сборки пакета. Переменные, начинающиеся с заглавной буквы, помещаются в файл `pkginfo` и используются во время установки.
pkginst	Ссылается на пакет по его аббревиатуре или определенному экземпляру (например, `SUNWcadap.4`).

Для получения дополнительной информации см. справочную страницу `pkgmk(1)`.

5 Проверьте содержимое пакета.

```
$ pkgchk -d device-name pkg-abbrev
Checking uninstalled directory format package pkg-abbrev
from device-name
## Checking control scripts.
## Checking package objects.
## Checking is complete.
$
```

`-d` *device-name* — Указывает расположение пакета. Обратите внимание, что *device-name* может быть как полным именем пути каталога, так и идентификатором в случае использования ленточного накопителя или съемного диска.

pkg-abbrev — Указывает имя одного или нескольких пакетов (разделенных пробелами), которые следует проверить. Если этот параметр отсутствует, команда `pkgchk` проверяет все имеющиеся пакеты.

Команда `pkgchk` выводит на печать перечень проверяемых параметров пакета и выводит на экран предупреждения или сообщения об ошибках. Для получения дополнительной информации о команде `pkgchk` см. раздел «Проверка целостности пакета» на стр. 96.

Внимание – К ошибкам следует относиться очень серьезно. Ошибка может означать, что необходимо внести изменения в сценарий. Проверьте все ошибки и продолжите работу, если не согласны с результатом выполнения команды `pkgchk`.

Пример 2–2 Сборка пакета

В данном примере используется файл `prototype`, созданный в разделе «Настройка файла `prototype`, созданного с помощью команды `pkgproto`» на стр. 41.

```
$ cd /home/jane/InfoFiles
$ pkgmk
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter  set to "system990708093144"
WARNING: parameter  set to "none"
## Attempting to volumize 13 entries in pkgmap.
part  1 -- 3170 blocks, 17 entries
## Packaging one part.
/var/spool/pkg/SUNWcadap/pkgmap
/var/spool/pkg/SUNWcadap/pkginfo
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/demo/file1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/lib/file2
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file3.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file4.1
```

```
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/windex
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file5
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file6
## Validating control scripts.
## Packaging complete.
$
```

Пример 2–3 Определение исходного каталога для перемещаемых файлов

Если пакет содержит перемещаемые файлы, то можно использовать параметр `-b` *base-src-dir* в команде `pkgmk` для указания имени пути, которое следует добавить к началу перемещаемых имен путей во время создания пакета. Этот параметр может быть полезен, если для перемещаемых файлов не использовался формат *path1=path2* или командой `!search` не был указан путь поиска в файле `prototype`.

Команда выполняет сборку пакета со следующими характеристиками:

- Сборка пакета осуществляется с помощью пробного файла `prototype`, созданного командой `pkgproto`. Для получения дополнительной информации см. «Пример. Создание файла `prototype`с помощью команды `pkgproto`» на стр. 40.
- Сборка пакета осуществляется без изменения полей *path*.
- Пакет добавляет запись в файл `pkginfo`.

```
$ cd /home/jane/InfoFiles
$ pkgmk -o -b /home/jane
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter  set to "system960716102636"
WARNING: parameter  set to "none"
## Attempting to volumize 13 entries in pkgmap.
part  1 -- 3170 blocks, 17 entries
## Packaging one part.
/var/spool/pkg/SUNWcadap/pkgmap
/var/spool/pkg/SUNWcadap/pkginfo
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/demo/file1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/lib/file2
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file3.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file4.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/windex
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file5
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file6
## Validating control scripts.
## Packaging complete.
```

В этом примере пакет собирается в каталоге по умолчанию `/var/spool/pkg` с помощью параметра `-o`. Этот параметр перезаписывает пакет, созданный в Пример 2–2.

Пример 2–4 Указание других исходных каталогов для информационных файлов и объектов пакета

Если разместить информационные файлы пакета (например, `pkginfo` и `prototype`) и объекты пакета в двух разных каталогах, то можно создать пакет с помощью параметров `-b` *base-src-dir* и `-r` *rootpath* команды `pkgmk`. Если объекты пакета находятся в каталоге `/product/pkgbin`, а другие информационные файлы пакета в каталоге `/product/pkgsrc`, то можно использовать следующую команду для помещения пакета в каталог `/var/spool/pkg`:

```
$ pkgmk -b /product/pkgbin -r /product/pkgsrc -f /product/pkgsrc/prototype
```

При желании можно использовать указанные ниже команды для достижения того же результата:

```
$ cd /product/pkgsrc
$ pkgmk -o -b /product/pkgbin
```

В данном примере команда `pkgmk` использует текущий рабочий каталог для поиска оставшихся частей пакета (таких, как информационные файлы `prototype` и `pkginfo`).

См. также Для получения информации о добавлении необязательных информационных файлов и сценариев установки к пакету см. Глава 3, «Расширение функциональности пакета (задачи)». После сборки пакета следует проверить его целостность. В Глава 4, «Проверка и запись пакета» рассказано, как это сделать. В этой главе представлены поэтапные указания по записи проверенного пакета на распространяемый носитель.

◆ ◆ ◆ ГЛАВА 3

Расширение функциональности пакета (задачи)

В этой главе описываются способы создания необязательных информационных файлов и сценариев установки пакета. В Глава 2, «Сборка пакета» были приведены минимальные требования к системе для сборки пакета. В данной главе приведены дополнительные функции, которые можно встроить в пакет. Эти дополнительные функции зависят от критериев, которые учитывались при разработке пакета. Для получения дополнительных сведений см. раздел «Что следует принять во внимание перед сборкой пакета» на стр. 17.

Ниже следует общий перечень вопросов, рассматривающихся в данной главе.

Создание информационных файлов и сценариев установки (карта задач)

В карте задач ниже содержится описание дополнительных функций, которые можно внедрить в пакет.

ТАБЛИЦА 3–1 Создание информационных файлов и сценариев установки (карта задач)

Задача	Описание	Инструкции
1. Создание информационных файлов	*Определение зависимостей пакета* Определение зависимостей пакета позволяет указать, будет ли пакет совместим с предыдущими версиями, будет ли он являться зависимым от других пакетов, или будут ли другие пакеты зависимы от него.	«Определение зависимостей пакета» на стр. 58
	Создайте сообщение об авторских правах Файл `copyright` позволит обеспечить защиту вашего приложения с юридической точки зрения.	«Написание сообщения об авторских правах» на стр. 60
	Зарезервируйте дополнительное пространство на целевой системе. Файл `space` резервирует блоки на целевой системе, что позволяет создать во время установки файлы, не указанные в файле `pkgmap`.	«Резервирование дополнительного дискового пространства на целевой системе» на стр. 62
2. Создание сценариев установки	*Получение информации из программы установки* Сценарий `request` позволяет получать информацию от лица, производящего установку пакета.	«Создание сценария `request` » на стр. 69
	Получение требуемых для установки данных о файловой системе Сценарий `checkinstall` позволяет выполнить анализ целевой системы и правильную настройку переменных или останов процесса установки.	«Сбор данных о файловой системе» на стр. 72
	Написание процедурных сценариев Процедурные сценарии позволяют создать специальные инструкции по установке во время конкретных фаз установки или удаления.	«Создание процедурных сценариев » на стр. 74
	Написание сценариев действия над классами Сценарии действия над классами позволяют установить набор инструкций, которые должны выполняться над определенными объектами пакета во время его установки и удаления.	«Создание сценариев действий над классом» на стр. 83

Создание информационных файлов

В этом разделе содержится описание необязательных информационных файлов пакета. С помощью этих файлов можно определить зависимости пакета, предоставить для пользователей сообщение об авторских правах, а также зарезервировать дополнительное дисковое пространство на целевой системе.

Определение зависимостей пакета

Необходимо определить, будет ли ваш пакет иметь зависимости от других пакетов или другие пакеты будут зависеть от вашего пакета. Зависимости и несовместимые версии можно определить с помощью двух дополнительных информационных файлов пакета, `compver` и `depend`.

Файл `compver` позволяет указать предыдущие версии пакета, совместимые с устанавливаемой версией.

Поставка файла `depend` позволяет определить три типа зависимостей, связанных с создаваемым пакетом. Эти типы зависимости перечислены ниже:

- *Требуемый пакет* – Создаваемый пакет зависит от существования другого пакета
- *Обратная зависимость* – Другой пакет зависит от существования этого пакета

Примечание – Использовать обратную зависимость следует только в том случае, если от существующего пакета зависит пакет, в котором нет файла `depend`.

- *Несовместимый пакет* – Созданный пакет несовместим с указанными пакетами

Файл `depend` разрешает только самые основные зависимости. Если пакет имеет зависимость от конкретного файла, его содержимого или поведения, файл `depend` не может обеспечить необходимую точность. В этом случае для подробной проверки зависимостей необходимо использовать сценарий `request` или сценарий `checkinstall`. Сценарий `checkinstall` также является единственным сценарием, который может без ошибок остановить процесс установки пакета.

Примечание – Убедитесь в том, что файлы `depend` и `compver` имеют записи в файле `prototype`. Файл должен иметь тип `i` (для информационного файла пакета).

Для получения дополнительной информации см. справочную страницу `depend(4)` и `compver(4)`.

▼ Определение зависимостей пакета

1 **Сделайте каталог, где содержатся информационные файлы, текущим рабочим каталогом.**

2 **Если имеются предыдущие версии пакета и необходимо указать, что новый пакет с ними совместим, с помощью любого текстового редактора создайте файл с именем** `compver`**.**

Перечислите версии, совместимые с создаваемым пакетом. Используйте следующий формат:

string string . . .

Значение строки *string* является идентичным значению, присвоенному параметру `VERSION` в файле `pkginfo`, для каждого совместимого пакета.

3 **Сохраните изменения и выйдите из редактора.**

4 **Если создаваемый пакет имеет зависимость от существования других пакетов или другие пакеты имеют зависимости от существования вашего, а также если ваш пакет несовместим с другими пакетами, создайте файл с именем** `depend` **в любом текстовом редакторе.**

Добавьте запись для каждой зависимости. Используйте следующий формат:

type pkg-abbrev pkg-name
(*arch*) *version*
(*arch*) *version . . .*

type — Определяет тип зависимости. Должен быть выражен одним из следующих символов: `P` (требуемый пакет), `I` (несовместимый пакет) или `R` (обратная зависимость).

pkg-abbrev — Указывает сокращенное имя (аббревиатуру) пакета, например `SUNWcadap`.

pkg-name — Указывает полное описание пакета, например: `Chip designers need CAD application software to design abc chips. Runs only on xyz hardware and is installed in the usr partition.`

(*arch*) — Необязательный параметр. Указывает тип оборудования, на котором выполняется пакет. Например, `sparc` или `x86`. При указании архитектуры системы в качестве разделителя необходимо использовать скобки.

version — Необязательный параметр. Указывает значение, назначенное параметру `VERSION` в файле `pkginfo`.

Для получения дополнительных сведений см. страницу depend(4).

5 **Сохраните изменения и выйдите из редактора.**

6 **Выполните *одну* из следующих задач.**

- Если необходимо создать дополнительные информационные файлы и сценарии установки, перейдите к следующей задаче: «Написание сообщения об авторских правах» на стр. 60.
- Если файл `prototype` еще не создан, выполните процедуру «Создание файла `prototype` с помощью команды `pkgproto`» на стр. 46. Перейдите к Шаг 7.
- Если файл`prototype` уже создан, измените его, добавив запись для каждого созданного файла.

7 Выполните сборку пакета.

В случае необходимости см. главу «Как собрать пакет» на стр. 49.

Пример 3–1 Файл `compver`

В этом примере имеются четыре версии пакета: 1.0, 1.1, 2.0, и новая версия пакета - 3.0. Новый пакет совместим со всеми тремя версиями. Файл `compver` для новой версии может выглядеть следующим образом:

```
release 3.0
release 2.0
version 1.1
1.0
```

Записи не обязательно располагать в последовательном порядке. Однако они должны в точности соответствовать определению параметра `VERSION` в файле `pkginfo` каждого пакета. В этом примере разработчики пакета используют различные форматы первых трех версий.

Пример 3–2 Файл `depend`

В этом примере предполагается, что пример пакета `SUNWcadap` требует наличия установленных в целевой системе пакетов `SUNWcsr` и `SUNWcsu`. Файл `depend` для пакета `SUNWcadap` выглядит следующим образом:

```
P SUNWcsr Core Solaris, (Root)
P SUNWcsu Core Solaris, (Usr)
```

См. также После сборки пакета установите его для подтверждения правильности выполнения установки и проверьте его целостность. Глава 4, «Проверка и запись пакета» содержатся пояснения по этим задачам и поэтапные указания по записи проверенного пакета на распространяемый носитель.

Создание сообщения об авторских правах

Вам необходимо решить, будет ли при установке пакета отображаться сообщение об авторских правах. Если оно должно отображаться, создайте файл `copyright`.

Примечание – Чтобы обеспечить юридическую защиту для создаваемого приложения, создайте файл `copyright` . Для того, чтобы получить текст содержимого файла, обратитесь в юридический отдел своей компании.

Чтобы донести до пользователей сообщение об авторских правах, создайте файл с именем `copyright`. Во время установки пакета сообщение будет показываться точно в том виде, в каком оно представлено в файле (без учета форматирования). Для получения дополнительной информации см. справочную страницу `copyright(4)`.

Примечание – Убедитесь, что файл `copyright` имеет запись в файле `prototype`. Файл должен иметь тип `i` (для информационного файла пакета).

▼ Написание сообщения об авторских правах

1 **Сделайте каталог, где содержатся информационные файлы, текущим рабочим каталогом.**

2 **Создайте файл с именем `copyright` с помощью любого текстового редактора.**

Введите текст сообщения об авторских правах точно в том виде, в каком оно должно отображаться при установке пакета.

3 **Сохраните изменения и выйдите из редактора.**

4 **Выполните *одну* из следующих задач.**

- Если необходимо создать дополнительные информационные файлы и сценарии установки, перейдите к следующей задаче «Резервирование дополнительного дискового пространства на целевой системе» на стр. 62.
- Если вы *не* создали файл `prototype`, выполните процедуру «Создание файла `prototype` с помощью команды `pkgproto`» на стр. 46. Перейдите к Шаг 5.
- Если файл `prototype` уже создан, измените его, добавив записи для созданных информационных файлов.

5 **Выполните сборку пакета.**

В случае необходимости см. главу «Как собрать пакет» на стр. 49.

Пример 3–3 Файл `copyright`

Например, часть сообщения об авторских правах может выглядеть следующим образом:

```
Copyright (c) 2003 Company Name
All Rights Reserved

This product is protected by copyright and distributed under
licenses restricting copying, distribution, and decompilation.
```

См. также После сборки пакета установите его для подтверждения правильности выполнения установки и проверьте его целостность. Глава 4, «Проверка и запись пакета» содержатся пояснения по этим задачам и поэтапные указания по записи проверенного пакета на распространяемый носитель.

Резервирование дополнительного места на диске на целевой системе

Вам необходимо принять решение о том, требуется ли для вашего пакета дополнительное место на диске на целевой системе. Это дополнительное место, которое необходимо для объектов пакета. В этом случае создайте информационный файл `space` . Эта задача отличается от создания пустых файлов и каталогов во время установки, как указывается в разделе «Определение дополнительных объектов, которые будут создаваться во время установки» на стр. 44.

Команда `pkgadd` позволяет проверить наличие достаточного места на диске на основе определений объектов в файле `pkgmap`. Однако для пакета может требоваться дополнительное место на диске помимо того, что определено для объектов в файле `pkgmap`. Например, пакет должен создать после установки файл, в котором может быть база данных, файлы журнала или растущий файл иного рода, расходующий свободное место на диске. Чтобы зарезервировать достаточный объем дискового пространства, добавьте файл `space`, в котором указываются требования к объему дискового пространства. С помощью команды `pkgadd` производится проверка дополнительного дискового пространства, указанного в файле `space`. Для получения дополнительной информации см. справочную страницу `space(4)`.

Примечание – Убедитесь, что для файла `space` имеется запись в файле `prototype`. Файл должен иметь тип `i` (для информационного файла пакета).

▼ Резервирование дополнительного дискового пространства на целевой системе

1 **Сделайте каталог, где содержатся информационные файлы, текущим рабочим каталогом.**

2 **С помощью любого текстового редактора создайте файл с именем** `space`**.**

Укажите требования к дополнительному месту на диске, которое требуется для вашего пакета. Используйте следующий формат:

pathname blocks inodes

pathname Указывает имя каталога, которое может или не может использоваться в качестве точки монтирования для файловой системы.

blocks Указывает место для резервирования, выраженное количеством блоков по 512 байт.

inodes Указывает количество требуемых индексных дескрипторов.

Для получения дополнительной информации см. справочную страницу `space(4)`.

3 **Сохраните изменения и выйдите из редактора.**

4 **Выполните одну из следующих задач.**

- Если необходимо создать сценарии установки, перейдите к следующей задаче: «Создание сценария `request` » на стр. 69.
- Если файл `prototype` не создан, выполните процедуру, указанную в разделе «Создание файла `prototype` с помощью команды `pkgproto`» на стр. 46. Перейдите к Шаг 5.
- Если файл `prototype` уже создан, измените его, добавив записи для созданных информационных файлов.

5 **Выполните сборку пакета.**

В случае необходимости см. главу «Как собрать пакет» на стр. 49.

Пример 3–4 Файл `space`

В этом примере файла `space` указаны 1000 блоков по 512 байт и 1 индексный дескриптор, которые будут зарезервированы в каталоге `/opt` целевого компьютера.

```
/opt   1000   1
```

См. также После сборки пакета установите его для подтверждения правильности выполнения установки и проверьте его целостность. В Глава 4, «Проверка и запись пакета» содержатся пояснения по этим задачам и поэтапные указания по записи проверенного пакета на распространяемый носитель.

Создание сценариев установки

В этом разделе описываются дополнительные сценарии установки пакетов. С помощью команды `pkgadd` автоматически выполняются все действия, необходимые для установки пакета, с использованием информационных файлов в качестве входных данных. Предоставлять сценарии установки пакета *нет необходимости*. Однако если для пакета требуется создать специальные процедуры установки, это можно сделать с помощью сценариев установки. Сценарии установки:

- Должны выполняться в интерпретаторе команд `sh`
- Должны содержать команды интерпретатора sh и текст
- Могут не содержать идентификатор интерпретатора команд `#!/bin/sh`
- Не обязательно должны быть исполняемым файлом

Выполнять пользовательские действия можно при помощи четырех типов сценариев установки:

- Сценарий `request`

  Сценарий `request` запрашивает данные у администратора, выполняющего установку пакета, с целью изменения или назначения переменных среды.

- Сценарий `checkinstall`

  Сценарий `checkinstall` осуществляет проверку целевого компьютера на необходимые данные, может устанавливать или изменять переменные среды пакета и определять, можно ли продолжать процесс установки.

  Примечание – Сценарий `checkinstall` доступен начиная с выпуска Solaris 2.5 и совместимых выпусков.

- Процедурные сценарии

  Процедурные сценарии определяют процедуру, которая будет вызвана перед или после удаления или установки пакета. Четыре процедурных сценария - это `preinstall`, `postinstall`, `preremove` и `postremove`.

- Сценарии действий над классами

Сценарии действия над классами определяют действие или набор действий, которые должны применяться к классу или файлам во время установки или удаления. Можно определить свои собственные классы. Кроме того, можно также использовать один из четырех стандартных классов (`sed`, `awk`, `build` и `preserve`).

Обработка сценария во время установки пакета

Тип используемых сценариев зависит от того, в какой момент процесса установки необходимо действие над классом. После установки пакета с помощью команды `pkgadd` выполняются следующие действия.

1. Выполняется сценарий `request`

   Пакет может запросить данные от администратора, который выполняет установку, только во время этого действия.

2. Выполняется сценарий `checkinstall`

   Сценарий `checkinstall` производит сбор данных о файловой системе и может создавать или изменять переменные среды для управления последующей установкой. Для получения дополнительной информации о переменных среды пакета см. раздел «Переменные среды пакета» на стр. 26.

3. Выполняется сценарий `preinstall`

4. Выполняется установка объектов пакета для каждого устанавливаемого класса.

   Установка этих файлов производится по классам. Сценарии действий над классами выполняются соответствующим образом. Список классов, над которыми выполняется действие, и порядок их установки изначально определяется с помощью параметра `CLASSES` в файле `pkginfo`. Однако сценарий `request` или `checkinstall` может изменять значение параметра `CLASSES`. Для получения дополнительной информации об обработки классов во время установки см. раздел «Обработка классов во время установки пакета» на стр. 76.

   a. Выполняется создание символьных ссылок, устройств, именованных каналов и необходимых каталогов.

   b. Выполняется установка стандартных файлов (типы файлов `e`, `v`, `f`) на основе их классов

      Сценарий действия над классом разрешает установку только обычных файлов. Все другие объекты пакета создаются автоматически на основе информации в файле `pkgmap`.

   c. Создаются все жесткие ссылки.

5. Выполняется сценарий `postinstall`.

Обработка сценариев во время удаления пакета

При удалении пакета команда `pkgrm` выполняет следующие действия:

1. Выполняется сценарий `preremove`.
2. Выполняется удаление объектов пакета для каждого класса

   Удаление также производится по классам. Обработка сценариев удаления производится в порядке, обратному порядку установки, на основе последовательности, определенной в параметре `CLASSES`. Для получения дополнительной информации об обработки классов во время установки см. раздел «Обработка классов во время установки пакета» на стр. 76.

   a. Выполняется удаление жестких ссылок.
   b. Выполняется удаление стандартных файлов.
   c. Выполняется удаление символических ссылок, устройств и именованных каналов.
3. Выполняется сценарий `postremove`.

Обработка сценария `request` во время удаления пакета не производится. Однако результат выполнения сценария сохраняется в установленном пакете и предоставляется для сценариев удаления. Результат выполнения сценария `request` - это список переменных среды.

Доступные для сценариев переменные среды пакета

Для всех сценариев установки доступны следующие группы переменных среды. Некоторые из переменных среды можно изменить с помощью сценария `request` или сценария `checkinstall`.

- Сценарий `request` или сценарий `checkinstall` может установить или изменить любой из стандартных параметров в файле `pkginfo`, за исключением необходимых параметров. Параметры стандартной установки подробно описаныя на справочной странице `pkginfo(4)`.

  Примечание – Параметр `BASEDIR` можно изменить только в выпусках начиная с Solaris 2.5, а также в совместимых выпусках.

- Можно определить собственные переменные среды установки путем установки значений в файле `pkginfo`. Эти переменные среды должны быть указаны алфавитно-цифровыми символами. Первый символ должен быть заглавным. Любая из переменных среды может быть изменена с помощью сценария `request` или `checkinstall`.

- Оба сценария - `request` и `checkinstall` - могут определять новые переменные среды путем установки данных и помещения в среду установки.
- В следующей таблице перечислены переменные среды, которые доступны для сценариев установки в среде. Ни одна из этих переменных не может быть изменена в ходе выполнения сценария.

Переменная среды	Описание
`CLIENT_BASEDIR`	Основной каталог по отношению к целевому компьютеру. Переменная `BASEDIR` используется при ссылке на определенный объект из устанавливающей системы (обычно это сервер), а `CLIENT_BASEDIR` - это путь, включающий файлы, которые будут размещены на клиентской системе. Параметр `CLIENT_BASEDIR` существует при наличии параметра `BASEDIR` и полностью идентичен параметру `BASEDIR` при отсутствии параметра `PKG_INSTALL_ROOT`.
`INST_DATADIR`	Каталог находится в том расположении, где производится чтение пакета. Если чтение пакета производится с ленты, эта переменная будет иметь значение расположения временного каталога, куда пакет переносится в формате каталога. Другими словами, в предположении, что расширения имени пакета не существует (например, `SUNWstuff.d`), сценарий `request` для пакета можно найти в местоположении `$INST_DATADIR/$PKG/install`.
`PATH`	Для поиска команд при вызове сценария интерпретатор `sh` использует список поиска. Параметр `PATH` обычно имеет следующий формат: `/sbin:/usr/sbin:/usr/bin:/usr/sadm/install/bin`.
`PKGINST`	Идентификатор экземпляра устанавливаемого пакета. Если другое экземпляр пакета не установлен, это значение является сокращенным именем (аббревиатурой) пакета (например,`SUNWcadap`). В противном случае это значение является аббревиатурой пакета с добавлением суффикса, например `SUNWcadap.4`.
`PKGSAV`	Каталог, в которым могут сохраняться файлы для использования сценариями удаления, или где можно найти файлы, сохраненные ранее. Доступны только в выпуске Solaris 2.5 и совместимых выпусках.
`PKG_CLIENT_OS`	Операционная система клиента, в которую устанавливается пакет. Значение переменной - `Solaris`.
`PKG_CLIENT_VERSION`	Версия ОС Solaris в формате `x.y`.
`PKG_CLIENT_REVISION`	Версия сборки ОС Solaris.
`PKG_INSTALL_ROOT`	Корневая файловая система на целевом компьютере, куда производится установка пакета. Эта переменная существует, только если производился вызов команд `pkgadd` и `pkgrm` с параметром `-R`. Это условное наличие облегчает использование в процедурных сценариях в форме `${PKG_INSTALL_ROOT}/`*некий_путь*.
`PKG_NO_UNIFIED`	Переменная среды, которая получает значение при вызове команд `pkgadd` и `pkgrm` с помощью параметров `-M` и `-R`. Эта переменная среды используется для сценариев установки или команд пакета, которые являются частью переменой пакета.

Переменная среды	Описание
UPDATE	Эта переменная среды не существует в большинстве сред установки. Если эта переменная существует (со значением yes), это может означать одно из следующего: Наличие установленного пакета с тем же именем, версией и архитектурой. Пакет устанавливается с замещением установленного пакета с тем же именем по требованию администратора. В этих случаях всегда используется основной каталог.

Получение информации пакета для сценария

Для получения информации о пакете можно использовать две команды:

- Команда pkginfo возвращает информацию о пакетах приложения, например, идентификатор экземпляра и имя пакета.
- Команда pkgparam возвращает значения для запрошенных переменных среды.

  Для получения дополнительной информации см. справочную страницу pkginfo(1), справочную страницу pkgparam(1) и Глава 4, «Проверка и запись пакета».

Коды выхода для сценария

Каждый сценарий должен завершать работу с использованием кодов выхода, перечисленных в таблице ниже.

ТАБЛИЦА 3–2 Коды выхода сценария установки

Код	Описание
0	Успешное выполнение сценария.
1	Неустранимый сбой. При этом производится прерывание процесса установки.
2	Предупреждение или состояние возможной ошибки. Процесс установки продолжается. По завершении установки показывается сообщение с предупреждением.
3	Выполнение команды pkgadd останавливается. Это код возвращается только сценарием checkinstall.
10	По завершении установки всех пакетов необходимо перезагрузить компьютер. (Это значение должно быть добавлено к одному из одноразрядных числовых кодов выхода.)
20	По завершении установки текущего пакета следует немедленно перезагрузить компьютер. (Это значение должно быть добавлено к одному из одноразрядных числовых кодов выхода.)

Примеры кодов выхода, возвращаемых сценариями установки, приведены в Глава 5, «Практические примеры создания пакета».

Примечание – Все включенные в пакет сценарии установки должны иметь записи в файле `prototype`. Файл должен иметь тип `i` (для сценария установки пакета).

Создание сценария `request`

Сценарий `request` представляет собой единственный способ взаимодействия администратора с устанавливаемым пакетом. Это сценарий можно использовать, например, чтобы предоставить администратору возможность выбора установки дополнительных компонентов пакета.

Результат выполнения сценария `request` должен представлять собой список переменных и их значений. В этот список могут входить любые параметры, созданные в файле `pkginfo`, а также параметры `CLASSES` и `BASEDIR`. В этом списке могут быть указаны переменные среды, которые не назначены в других компонентах. Однако файл `pkginfo` должен всегда содержать умолчания значений (когда это практически необходимо). Для получения дополнительной информации о переменных среды пакета см. раздел «Переменные среды пакета» на стр. 26.

Когда сценарий `request` производит назначение значений для переменной, они должны быть предоставлены для использования командой `pkgadd` и других сценариев установки.

Поведение сценария `request`

- Сценарий `request` не может изменять файлы. Этот сценарий только позволяет осуществлять взаимодействие с администратором, осуществляющим установку, и создавать список значений переменных среды на основе результатов этого взаимодействия. Сценарий `request` выполняется от имени непривилегированного пользователя `install`, если такой пользователь существует. В противном случае сценарий выполняется от имени пользователя `root`.
- С помощью команды `pkgadd` производится вызов сценария `request` с одним аргументом, который указывает имя файла ответов для сценария. В файле ответов хранятся ответы администратора.
- При удалении пакета сценарий `request` не выполняется. Однако назначенные сценарием переменные среды сохраняются и доступны при удалении пакета.

Правила разработки сценариев `request`

- Для каждого пакета может существовать только один сценарий `request`. Этот сценарий должен иметь имя `request`.
- В среду установки также необходимо добавить значения переменных среды для использования командой `pkgadd` и других сценариев пакета, добавив их в файл ответов (который сценарий опознает как `$1`).

- Переменные среды системы и стандартные переменные среды установки (за исключением параметров `CLASSES` и `BASEDIR`) невозможно изменить с помощью сценария `request`. Можно изменить все остальные созданные переменные среды.

Примечание – Сценарий `request` может изменить только параметр `BASEDIR`, начиная с ОС Solaris 2.5 и совместимых выпусков.

- Каждой переменной среды, с которая может работать сценарий `request`, в файле `pkginfo` должно быть присвоено значение по умолчанию.
- Формат выходного списка должен быть следующим: *PARAM=value*. Пример:

```
CLASSES=none class1
```

- В качестве стандартного вход для сценария `request` определен терминал администратора.
- Не выполняйте какого-либо специального анализа целевого компьютера с помощью сценария `request`. Проверять компьютер на наличие определенных файлов или поведения, а также устанавливать переменные среды на основе такого анализа рискованно. Гарантий, что сценарий `request` будет выполняться в момент установки пакета, не существует. Администратор, который выполняет установку пакета, может предоставить файл ответов, который произведет вставку переменных среды даже без вызова сценария `request`. Если сценарий `request` также производит оценку целевой файловой системы, эта оценка может быть не произведена. Анализ целевого компьютера для специальной обработки лучше оставить сценарию `checkinstall`.

Примечание – Если администратор, который будет устанавливать пакет, использует решение JumpStart™, то установка пакета не должна осуществляться в интерактивном режиме. Необходимо или не предоставлять для своего пакета сценарий `request`, или сообщать администраторам, чтобы до установки пакета они использовали команду `pkgask`. Команда `pkgask` сохраняет ответы администраторов в сценарий`request`. Для получения дополнительных сведений о команде `pkgask` см. справочную страницу `pkgask(1M)`.

▼ Создание сценария `request`

1 **Сделайте каталог, где содержатся информационные файлы, текущим рабочим каталогом.**

2 **Создайте файл `request` с помощью любого текстового редактора.**

3 **По завершении сохраните изменения и закройте редактор.**

4 **Выполните одну из следующих задач.**

- Если необходимо создать дополнительные сценарии установки, перейдите к следующей задаче, «Сбор данных о файловой системе» на стр. 72.
- Если файл `prototype` еще не создан, выполните процедуру «Создание файла `prototype` с помощью команды `pkgproto`» на стр. 46. Перейдите к Шаг 5.
- Если файл `prototype` уже создан, измените его, добавив запись для созданного сценария установки.

5 **Выполните сборку пакета.**

В случае необходимости см. главу «Как собрать пакет» на стр. 49.

Пример 3–5 Создание сценария `request`

Когда сценарий `request` присваивает значения переменным среды, они должны быть доступны для команды `pkgadd`. В примере ниже показан сегмент сценария `request`, который выполняет эту задачу для четырех переменных среды: `CLASSES`, `NCMPBIN`, `EMACS` и `NCMPMAN`. Предположим, что эти переменные были определены во время интерактивного сеанса сценария ранее.

```
# make environment variables available to installation
# service and any other packaging script we might have

cat >$1 <<!
CLASSES=$CLASSES
NCMPBIN=$NCMPBIN
EMACS=$EMACS
NCMPMAN=$NCMPMAN
!
```

См. также После сборки пакета установите его для подтверждения правильности выполнения установки и проверьте его целостность. В Глава 4, «Проверка и запись пакета» содержатся пояснения по этим задачам и поэтапные указания по записи проверенного пакета на распространяемый носитель.

Сбор данных о файловой системе с помощь сценария `checkinstall`

Сценарий `checkinstall` выполняется вскоре после необязательного сценария `request`. Сценарий `checkinstall` выполняется от имени пользователя `install`, если такой существует, или от имени пользователя `nobody`. Сценарий`checkinstall` не имеет полномочий на изменение данных файловой системы. Однако в зависимости от

собранной информации сценарий может создавать или изменять переменные среды для контроля за ходом итоговой установки. Сценарий также может безопасно остановить процесс установки.

Сценарий `checkinstall` предназначен для выполнения основной проверки файловой системы, которая будет стандартной для выполнения команды `pkgadd`. Например, этот сценарий может использоваться для предварительной проверки возможности перезаписи существующих файлов или управления общими зависимостями программ. Файл `depend` производит управление только зависимостями на уровне пакета.

В отличие от сценария `request`, сценарий `checkinstall` выполняется вне зависимости от наличия файла с результатами выполнения. Наличие сценария не позволяет считать пакет интерактивным. Сценарий `checkinstall` можно использовать, если сценарий `request` запрещен или взаимодействие с администраторам нецелесообразно.

Примечание – Сценарий `checkinstall` доступен, начиная с Solaris 2.5 и совместимых выпусков.

Поведение сценария `checkinstall`

- Сценарий `checkinstall` не может изменять файлы. Этот сценарий только проводит анализ состояния системы и создание списка значений переменных среды на основе этого действия. Для принудительного выполнения этого ограничения сценарий `checkinstall` выполняется от имени непривилегированного пользователя `install` (если этот пользователь существует). В противном случае сценарий выполняется от имени непривилегированного пользователя `nobody`. Сценарий `checkinstall` не имеет полномочий суперпользователя.
- Команда `pkgadd` производит вызов сценария `checkinstall` с использованием одного аргумента, содержащего информацию файла ответов сценария. Файл с результатами выполнения сценария - это файл, где хранятся данные выбора администратора.
- Сценарий `checkinstall` не выполняется при удалении пакета. Однако назначенные сценарием переменные среды сохраняются и доступны при удалении пакета.

Правила разработки сценария `checkinstall`

- Для каждого пакета может использоваться только один сценарий `checkinstall`. Сценарий должен иметь имя `checkinstall`.
- В среду установки также необходимо добавить значения переменных среды для использования командой `pkgadd` и других сценариев пакета, добавив их в файл ответов (который сценарий опознает как `$1`).
- Переменные системной среды и переменные стандартной установки, за исключением параметров `CLASSES` и `BASEDIR`, невозможно изменить с помощью сценария `checkinstall`. Можно изменить все остальные созданные переменные.

- Каждая переменная среды, которая может управляться сценарием `checkinstall` должна иметь значение умолчания, назначенное в файле `pkginfo`.
- Список результирующих параметров должен иметь следующий формат *PARAM=value*. Пример:

```
CLASSES=none class1
```

- Во время выполнения сценария `checkinstall` взаимодействие с администратором не осуществляется. Все взаимодействие с оператором ограничивается сценарием `request`.

▼ Сбор данных о файловой системе

1. **Сделайте каталог, где содержатся информационные файлы, текущим рабочим каталогом.**
2. **С помощью любого текстового редактора создайте файл с именем `checkinstall`.**
3. **По завершении сохраните изменения и закройте редактор.**
4. **Выполните одну из следующих задач.**
   - Если необходимо создать дополнительные сценарии установки, перейдите к следующей задаче: «Создание процедурных сценариев » на стр. 74.
   - Если файл `prototype` еще не создан, выполните процедуру «Создание файла `prototype` с помощью команды `pkgproto`» на стр. 46. Перейдите к Шаг 5.
   - Если файл `prototype` уже создан, измените его, добавив запись для созданного сценария установки.
5. **Выполните сборку пакета.**

   В случае необходимости см. главу «Как собрать пакет» на стр. 49.

Пример 3–6 Создание сценария `checkinstall`

В этом примере сценария `checkinstall` производится проверка, установлено ли требуемое для пакета `SUNWcadap` ПО базы данных.

```
# checkinstall script for SUNWcadap
#
# This confirms the existence of the required specU database

# First find which database package has been installed.
pkginfo -q SUNWspcdA    # try the older one
```

```
if [ $? -ne 0 ]; then
   pkginfo -q SUNWspcdB    # now the latest

      if [ $? -ne 0 ]; then    # oops
            echo "No database package can be found. Please install the"
            echo "SpecU database package and try this installation again."
            exit 3        # Suspend
      else
            DBBASE="`pkgparam SUNWsbcdB BASEDIR`/db"    # new DB software
      fi
else
      DBBASE="`pkgparam SUNWspcdA BASEDIR`/db"    # old DB software
fi

# Now look for the database file we will need for this installation
if [ $DBBASE/specUlatte ]; then
      exit 0        # all OK
else
      echo "No database file can be found. Please create the database"
      echo "using your installed specU software and try this"
      echo "installation again."
      exit 3        # Suspend
fi
```

См. также После сборки пакета установите его для подтверждения правильности выполнения установки и проверьте его целостность. В Глава 4, «Проверка и запись пакета» содержатся пояснения по этим задачам и поэтапные указания по записи проверенного пакета на распространяемый носитель.

Создание процедурных сценариев

Процедурные сценарии позволяют установить набор инструкций, которые выполняются в конкретные моменты при установке или удалении пакета. Каждый из четырех процедурных сценариев должен иметь одно из четырех заранее определенных имен в зависимости от выполняемых инструкций. Сценарии выполняются без аргументов.

- Сценарий `preinstall`

  Выполняется перед запуском установки класса. При выполнении этого сценария установка файлов не производится.

- Сценарий `postinstall`

  Выполняется после установки всех томов.

- Сценарий `preremove`

Выполняется перед запуском удаления класса. При выполнении этого сценария удаление файлов не производится.

- Сценарий `postremove`

  Выполняется после удаления всех классов.

Поведение сценариев процедур

Процедурные сценарии выполняются от имени `uid=root` и `gid=other`.

Правила разработки процедурных сценариев

- Каждый сценарий должен выполняться неоднократно, поскольку он выполняется по одному разу на каждый том пакета. Это означает, что выполнение сценария неограниченное количество раз с одними вводными данными позволяет получить те же результаты, что и при однократном выполнении.
- Каждый процедурный сценарий, который выполняет установку объектов пакета не в файл `pkgmap`, должен использовать команду `installf`, чтобы передать сведения в БД пакета об изменении или добавлении путевого имени. По завершении внесения всех изменений или добавлений следует вызвать эту команду с параметром `-f`. Таким способом устанавливать объекты пакета могут только сценарии `postinstall` и `postremove`. Для получения дополнительной информации см. справочную страницу `installf`(1M) и Глава 5, «Практические примеры создания пакета».
- При выполнении процедурных сценариев взаимодействие с администратором не допускается. Все взаимодействие с оператором ограничивается сценарием `request`.
- Каждый процедурный сценарий, который производит удаление файлов, не имеющих записи в файле `pkgmap`, должен использовать команду `removef` для передачи данных в БД пакета о том, что он не производит удаление путевого имени. По завершении удаления следует вызвать эту команду с параметром `-f`. Более подробная информация и примеры приведены на справочной странице `removef`(1M) и в Глава 5, «Практические примеры создания пакета».

Примечание – Команды `installf` и `removef` использовать не следует, поскольку процедурные сценарии не связываются автоматически с путевыми именами, указанными в файле `pkgmap`.

▼ Создание процедурных сценариев

1 Сделайте каталог, где содержатся информационные файлы, текущим рабочим каталогом.

2 **Создайте один или более процедурный сценарий с помощью любого текстового редактора.**

Процедурный сценарий должен иметь одно из ранее определенных имен: `preinstall`, `postinstall`, `preremove` или `postremove`.

3 **Сохраните изменения и выйдите из редактора.**

4 **Выполните одну из следующих задач.**

- Если необходимо создать сценарии действий над классами, перейдите к следующей задаче: «Создание сценариев действий над классом» на стр. 83.
- Если файл `prototype` еще не создан, выполните процедуру «Создание файла `prototype` с помощью команды `pkgproto`» на стр. 46. Перейдите к Шаг 5.
- Если файл `prototype` уже создан, измените его, добавив запись для каждого созданного сценария установки.

5 **Выполните сборку пакета.**

В случае необходимости см. главу «Как собрать пакет» на стр. 49.

См. также После сборки пакета установите его для подтверждения правильности выполнения установки и проверьте его целостность. В Глава 4, «Проверка и запись пакета» содержатся пояснения по этим задачам и поэтапные указания по записи проверенного пакета на распространяемый носитель.

Создание сценариев действий над классами

Определение классов объекта

Классы объектов позволяют выполнять серию действий над группой объектов пакета при установке или удалении. Назначение объектов классу производится в файле `prototype`. Все объекты пакета должны иметь определенный класс, а для объектов, не требующих специального действия, по умолчанию используется класс `none`.

Параметр установки `CLASSES`, назначенный в файле `pkginfo` представляет собой список устанавливаемых классов (включая класс `none`).

Примечание – Объекты, определенные в файле `pkgmap`, которые принадлежат классу, не указанному в этом параметре в файле `pkginfo`, установлены *не будут*.

Порядок установки определяется списком `CLASSES`. Класс `none`, если он имеется, всегда устанавливается первым и удаляется последним. Поскольку каталоги являются основной структурой поддержки для всех других объектов файловой системы, они

должны иметь класс none. Возможны исключения, однако, как правило, класс none является самым безопасным. Эта стратегия позволяет обеспечить создание каталогов до объектов, которые в них будут содержаться. Кроме того, каталоги удаляться не будут до тех пор, пока не будут удалены содержащиеся в них объекты.

Обработка классов во время установки пакета

Ниже приведено описание действий системы при установке класса. Эти действия повторяются для каждого тома пакета во время его установки.

1. Команда pkgadd позволяет создать список путевых имен.

   С помощью команды pkgadd производится создание списка путевых имен, над которыми производит действие сценарий. В каждой строке этого списка указывается исходный и целевой пути, разделенные пробелом. Исходный путь указывает, где в установочном томе расположен устанавливаемый объект . Целевой путь указывает местоположение в целевой системе, куда будет устанавливаться объект. Содержимое списка ограничивается следующими критериями.

   - В списке содержатся только путевые имена, принадлежащие связанному классу.
   - При сбое создания объекта пакета каталоги, именованные каналы, символьные устройства, блочные устройства и символические ссылки включаются в список с исходными путевым именем, равным /dev/null. Обычно эти элементы автоматически создаются с помощью команды pkgadd (если еще не существуют) и наделяются правильными атрибутами (режим, владелец, группа) согласно определению в файле pkgmap.
   - В список ни в коем случае не включаются связанные файлы типа l. Жесткие ссылки в данном классе создаются в элементе 4.

2. Если для установки конкретного класса не предоставлен сценарий действия над классом, путевые имена в создаваемом списке копируются из тома в соответствующее целевое местоположение.

3. Выполняется сценарий действия над классом (если существует).

   При вызове сценария действия над классом на его стандартный вход подается список, созданный в элементе 1. Если этот том является последним томом пакета или при отсутствии других объектов в этом классе сценарий выполняется с единственным аргументом ENDOFCLASS.

   Примечание – Даже если в пакете больше нет обычных файлов этого класса, производится вызов сценария действия над классом как минимум один раз с пустым списком и аргументом ENDOFCLASS.

4. Командаpkgadd осуществляет аудит содержимого и атрибутов, а также создание жестких ссылок.

После успешного выполнения элемента 2 или 3 команда pkgadd производит аудит содержимого и информации атрибутов для списка путевых имен. Команда pkgadd автоматически создает ссылки, связанные с классом. Обнаруженные несовпадения атрибутов в созданном списке исправляются для всех путевых имен.

Обработка классов при удалении пакета

Объекты удаляются по классам. Сначала удаляются классы, существующие для пакетов, однако не перечисленные в параметре CLASSES (например, объект, установленный с помощью команды installf). Классы, указанные в параметре CLASSES, удаляются в обратном порядке. Класс none всегда удаляется в последнюю очередь. Ниже описаны действия, которые производятся в системе при удалении класса.

1. Создание списка путевых имен с помощью команды pkgrm.

   Производится создание списка установленных путевых имен, принадлежащих указанному классу, с помощью команды pkgrm. Путевые имена, на которые ссылается другой пакет, из этого списка исключаются, если они не принадлежат к типу e. Тип файлов e означает, что по установки или удалении файл должен быть изменен.

   Если удаляемый пакет произвел изменение каких-либо файлов типа e во время установки, то при удалении производится удаление только добавленных строк. Не удаляйте непустой редактируемый файл. Удалите строки, которые были добавлены пакетом.

2. Если сценария действия над классом не существует, производится удаление путевых имен.

   Если в пакете отсутствуют сценарии действий над классами для этого класса, производится удаление всех путевых имен в списке, созданном с помощью команды pkgrm.

   Примечание – Файлам, принадлежащим к типу e (редактируемые), не назначается класс и связанный с ним сценарий действия над классом. Эти файлы удаляются в данный момент, даже если их путевые имена используются другими пакетами.

   ---

3. Если сценарий действия над классом уже существует, производится выполнение сценария.

   Командаpkgrm производит вызов сценария действия над классом, на стандартный вход которого подается список, созданный в элементе 1.

4. Командаpkgrm выполняет аудит.

   После успешного выполнения сценария действия над классом команда pkgrm производит удаление ссылок на путевые имена из БД пакета, если на них нет ссылок других пакетов.

Сценарий действия над классом

Сценарий действия над классом определяет набор действия для выполнения при установке или удалении пакета. Действия выполняются над группой путевых имен в зависимости от их определения класса. Примеры сценариев действий над классами приведены в Глава 5, «Практические примеры создания пакета».

Имя сценария действия над классом основывается на классе, над которым должно производиться действие, а также на том, должны ли эти действия производиться при установке или удалении пакета. В следующей таблице показаны два формата имен:

Формат имени	Описание
`i.`*класс*	Производит действия над путевыми именами в указанном классе при установке пакета
`r.`*класс*	Производит действия над путевыми именами в указанном классе при удалении пакета

Например, имя сценария установки для класса с именем `manpage` будет`i.manpage`. Сценарий удаления должен иметь имя `r.manpage`.

Примечание – Этот формат имен файлов не используется для файлов, которые принадлежат к системным классам `sed`, `awk` или `build`. Для получения дополнительной информации по этим классам см. раздел «Специальные системные классы » на стр. 79.

Поведение сценариев действий над классами

- Сценарии действий над классами выполняются под именем `uid=root` и `gid=other`.
- Сценарий выполняется для всех файлов данного класса на текущем томе.
- Команды `pkgadd` и `pkgrm` производят создание списка всех объектов, указанных в файле `pkgmap`, которые принадлежат классу. В результате сценарий действия над классом может выполнять операции только над путевыми именами, определенными в файле `pkgmap`, которые принадлежат конкретному классу.
- При последнем выполнении сценария действия над классом (т.е. файлов, принадлежащих этому классу, больше нет) сценарий действия над классом выполняется однократно с ключевым аргументом `ENDOFCLASS`.
- При выполнении сценариев действий над классами взаимодействие с администратором не допускается.

Правила разработки сценариев действий над классами

- Если пакет распределен более чем на один том, для каждого тома, содержащего хотя бы один файл, принадлежащий этому классу, выполняется сценарий действия над классом. Соответственно, каждый сценарий должен иметь возможность неоднократного выполнения. Это означает, что выполнение сценария неограниченное количество раз с одними вводными данными должны получаться те же результаты, что и при однократном выполнении.
- Если файл является частью класса, имеющего сценарий действия над классом, сценарий должен производить установку файла. Команда`pkgadd` не производит установку файлов, для которых существует сценарий действия над классом, однако с ее помощью производится проверка установки.
- Сценарий действия над классом ни в коем случае не должен производить добавление, изменение или удаление путевого имени или системного атрибута, отсутствующего в списке, который создан с помощью команды `pkgadd`. Для получения дополнительной информации о списке см. пункт 1 раздела «Обработка классов во время установки пакета» на стр. 76.
- Когда сценарий обнаруживает аргумент `ENDOFCLASS`, добавьте в сценарий действия после обработки, например очистку.
- Все взаимодействие с оператором ограничивается сценарием `request`. Не пытайтесь получить информацию от администратора с помощью сценария действия над классом.

Специальные системные классы

В системе имеются четыре специальных класса:

- Класс `sed`

  Обеспечивает метод для использования команд `sed` для изменения файлов после удаления или установки пакета.

- Класс `awk`

  Обеспечивает метод для использования инструкций `awk` для изменения файлов после удаления или установки пакета.

- Класс `build`

  Обеспечивает метод динамического создания или изменения файла с помощью команд интерпретатора sh

- Класс `preserve`

  Обеспечивает метод сохранения файлов, которые не должны перезаписываться при будущих установках пакета.

- Класс `manifest`

Обеспечивает автоматическую установку и удаление служб SMF (подсистемы управления службами, связанных с манифестом Для всех манифестов SMF в пакете должен использоваться класс `manifest`.

Если для некоторых файлов необходима специальная обработка, выполнение которой может быть невозможно с помощью команд `sed`, `awk` или`sh`, установка с помощью системных классов может производиться быстрее, чем с помощью нескольких классов и соответствующих сценариев действий над классами.

Сценарий класса `sed`

Класс `sed` обеспечивает метод изменения существующего объекта в целевой системе. Сценарий действия над классом `sed` выполняется при установке автоматически, если имеется файл, принадлежащий классу `sed`. Имя сценария действия над классом `sed` должно быть тем же, что и имя файла, над которым выполняются команды.

Сценарий действия над классом `sed` обеспечивает передачу инструкций `sed` в следующем формате:

Время, когда должны выполняться команды, показывается следующими двумя командами. При установке пакета производится выполнение команды программы `sed`, которая следует за командой `!install`. При удалении пакета производится выполнение команды программы `sed`, которая следует за командой `!remove`. Порядок, в котором используются эти команды, не учитывается.

Для получения дополнительной информации об командах программы `sed` см. справочную страницу `sed(1)`. Примеры сценариев действий над классами `sed` приведены в Глава 5, «Практические примеры создания пакета».

Сценарий класса `awk`

Класс `awk` обеспечивает метод изменения существующего объекта в целевой системе. Изменения производятся согласно командам программы `awk` в сценарии действия над классом `awk` .

Сценарий действия над классом `awk` выполняется при установке автоматически, если имеется файл, принадлежащий классу `awk`. В этом файле содержатся команды для сценария действия над классом `awk` в следующем формате:

Момент, когда должны выполняться команды, указывается следующими двумя командами. При установке пакета производится выполнение команд программы `awk`, которые следуют за командой `!install`. При удалении пакета производится автоматические выполнение команд, которые следуют за командой `!remove`. Эти команды могут выполняться в любом порядке.

Имя сценария действия над классом `awk` должно быть тем же, что и имя файла, над которым выполняются команды программы awk.

В качестве входных данных для команды awk используется файл, который необходимо изменить. В результате выполнения сценария производится полная замена исходного объекта. С помощью этого синтаксиса передать переменные среды для команды awk невозможно.

Для получения дополнительной информации о командах программы sed см. справочную страницу awk(1).

Сценарий класса build

Класс build позволяет создавать или изменять файл объекта пакета путем выполнения инструкций интерпретатора команд sh. Эти команды предоставляются как объект пакета. Команды выполняются автоматически при установке, если объект пакета принадлежит к классу build.

Имя сценария действия над классом build должно быть тем же, что и имя файла, над которым выполняются команды. Имя должно допускать выполнение действия с помощь команды sh. Результат выполнения сценария становится новой версией измененного или созданного файла Если при выполнении сценария результат отсутствует, создание или изменение файла не производится. Таким образом, сценарий может создать или изменять сам файл.

Например, если в пакете поставляется файл /etc/randomtable по умолчанию, и если в целевой системе файл еще отсутствует, то запись файла prototype может иметь следующий вид:

```
e build /etc/randomtable ? ? ?
```

Объект пакета /etc/randomtable может выглядеть следующим образом:

```
!install
# randomtable builder
if [ -f $PKG_INSTALL_ROOT/etc/randomtable ]; then
        echo "/etc/randomtable is already in place.";
        else
        echo "# /etc/randomtable" > $PKG_INSTALL_ROOT/etc/randomtable
        echo "1121554    # first random number" >> $PKG_INSTALL_ROOT/etc/randomtable
fi

!remove
# randomtable deconstructor
if [ -f $PKG_INSTALL_ROOT/etc/randomtable ]; then
        # the file can be removed if it's unchanged
        if [ egrep "first random number" $PKG_INSTALL_ROOT/etc/randomtable ]; then
            rm $PKG_INSTALL_ROOT/etc/randomtable;
        fi
fi
```

Еще один пример использования класса `build` приведен в Глава 5, «Практические примеры создания пакета».

Сценарий класса `preserve`

Класс `preserve` позволяет сохранить файл объекта пакета путем определения необходимости перезаписи существующего файла при установке пакета. При использовании сценария класса `preserve` возможны два варианта действий:

- Если устанавливаемый файл в целевом каталоге отсутствует, он будет установлен в стандартном порядке.
- Если устанавливаемый файл в целевом каталоге уже имеется, появится сообщение о том, что файл уже существует. Установка файла производиться не будет.

Для сценария `preserve` оба варианта являются успешными. Ошибка происходит только во втором варианте, если выполнить копирование файла в целевой каталог невозможно.

Начиная с Solaris 7, сценарий `i.preserve` и его копия - `i.CONFIG.prsv` - находятся в каталоге `/usr/sadm/install/scripts`, где имеются и другие сценарии действий над классами.

Измените сценарий, добавив имена файлов, изменение которых необходимо предотвратить.

Сценарий класса `manifest`

Класс `manifest` служит для автоматической установки и удаления служб SMF (подсистемы управления службами), связанных с манифестом SMF. Для получения ознакомительной информации относительно управления службами с помощью SMF см. Глава 17, «Managing Services (Overview),» в *System Administration Guide: Basic Administration*.

Все манифесты служб в пакетах должны быть обозначены классом `manifest`. В подсистему пакетов включены сценарии действий классов, предназначенные для установки и удаления манифестов служб При вызове команды `pkgadd`(1M) выполняется импорт манифеста. При вызове команды `pkgrm`(1M) выполняется удаление отключенных экземпляров в манифесте служб. Все службы в манифесте, для которых не осталось экземпляров, также удаляются. Если командам `pkgadd` (1M) или `pkgrm` (1M) передается параметр `-R`, то соответствующие действия манифеста служб выполняются при следующей перезагрузке системы с альтернативным корневым путем.

В приведенном ниже фрагменте кода из файла информации о пакете демонстрируется использование класса `manifest`.

```
# packaging files
i pkginfo
i copyright
```

```
i depend
i preinstall
i postinstall
i i.manifest
i r.manifest
#
# source locations relative to the prototype file
#
d none var 0755 root sys
d none var/svc 0755 root sys
d none var/svc/manifest 0755 root sys
d none var/svc/manifest/network 0755 root sys
d none var/svc/manifest/network/rpc 0755 root sys
f manifest var/svc/manifest/network/rpc/smserver.xml 0444 root sys
```

▼ Создание сценариев действий над классом

1 **Сделайте каталог, где содержатся информационные файлы, текущим рабочим каталогом.**

2 **Назначьте для объектов пакета необходимые имена классов в файле** `prototype`**.**

Например, при назначении объектам классов `application`и `manpage` строка будет выглядеть следующим образом:

```
f manpage /usr/share/man/manl/myappl.1l
f application /usr/bin/myappl
```

3 **Измените параметр** `CLASSES` **в файле** `pkginfo`**, добавив в него имена классов, которые необходимо использовать в пакете.**

Например, записи для классов `application`и `manpage` будут выглядеть следующим образом:

```
CLASSES=manpage application none
```

Примечание – Класс `none` всегда устанавливается первым и удаляется последним вне зависимости от места, где он указан в определении параметра `CLASSES`.

---

4 **Если вы создаете сценарий действия над классом для файла, принадлежащего классу** `sed`**,** `awk` **или** `build`**, сделайте каталог, содержащий объект пакета, текущим рабочим каталогом.**

5 **Создайте сценарии действий над классами или объекты пакета (для файлов, которые принадлежат классу** `sed`**,** `awk` **или** `build`**).**

Например, сценарий установки для класса с именем `application` будет иметь имя `i.application`, а сценарий удаления будет иметь имя `r.application`.

Помните о том, что если файл является частью класса, который имеет сценарий действия над классом, то этот файл должен быть установлен сценарием. Командаpkgadd не производит установку файлов, для которых существует сценарий действия над классом, однако с ее помощью производится проверка установки. Кроме того, если вы определите класс, однако не создадите сценарий действия над классом, единственное действие, которое будет возможно для этого класса - копирование компонентов с установочного носителя в целевую систему (поведения по умолчанию команды `pkgadd`).

6 **Выполните *одну* из следующих задач.**

- Если файл `prototype` еще *не создан*, выполните процедуру «Создание файла `prototype` с помощью команды `pkgproto`» на стр. 46 и перейдите к Шаг 7.
- Если файл `prototype` уже создан, измените его, добавив запись для каждого созданного сценария установки.

7 **Выполните сборку пакета.**

В случае необходимости см. главу «Как собрать пакет» на стр. 49.

Дополнительные сведения

Что делать дальше

После сборки пакета установите его для подтверждения правильности выполнения установки и проверьте его целостность. В Глава 4, «Проверка и запись пакета» содержатся пояснения по выполнению этой задачи и поэтапные указания по записи проверенного пакета на носитель для распространения.

Создание подписанных пакетов

Процесс создания подписанных пакетов включает в себя несколько этапов. Для него требуется знание новых концепций и терминологии. В этом разделе приводится информация о подписанных пакетах, терминологии, а также сведения об управлении сертификатами. В этом разделе также представлены поэтапные указания по созданию подписанного пакета.

Подписанные пакеты

Подписанный пакет - это стандартный пакет формата потока, который имеет цифровую подпись (цифровую подпись PKCS7 с кодировкой PEM, определение которой приводится ниже). При его установке производит следующая проверка:

- Пакет получен от того, кем он был подписан
- Факт подписывания пакета этим лицом
- Пакет не изменялся с того момента, когда он был подписан

- Лицо или компания, подписавшая пакет, является доверенным (-ой)

Подписанный пакет идентичен неподписанному, за исключением наличия подписи. Подписанный пакет имеет двоичную совместимость с неподписанным пакетом. Поэтому подписанный пакет может использоваться совместно со старыми версиями средств для работы с пакетами. Тем не менее, в этом случае подпись игнорируется.

В технологии создания подписанных пакетов используются новые приемы и терминология, которые описаны в приведённой ниже таблице.

Термин	Определение
ASN.1	Абстрактная синтаксическая нотация версии 1 - Способ выражения абстрактных объектов. Например, ASN.1 определяет сертификат открытого ключа, все объекты, составляющие сертификат и порядок, в котором эти объекты установлены. Однако ASN.1 не указывает, каким образом эти объекты сериализуются для хранения или передачи.
X.509	ITU-T Рекомендация X.509 - Описывает распространенный синтаксис сертификата ключа X.509.
DER	Правила различенной кодировки - Двоичное представление объекта ASN.1. Определяет, каким образом объект ASN.1 сериализуется для хранения или передачи в вычислительной среде.
PEM	Сообщение о улучшенной защите - Способ кодирования файла (в DER или другом двоичном формате) с помощью кодировки Base64 и некоторых дополнительных заголовков. PEM обычно использовался для кодирования электронных сообщений типа MIME. PEM также широко используется для кодирования сертификатов и открытых ключей в файле, существующем в файловой системе или в сообщении электронной почты.
PKCS7	Стандарт шифрования с открытым ключом #7 - В этом стандарте описываются общие требования к синтаксису данных, которые могут быть зашифрованы, например цифровые подписи и цифровые конверты. Подписанный пакет содержит встроенную подпись PKCS7. В этой подписи в минимальных объемах содержатся сводные данные по пакету, а также данные сертификата открытого ключа в формате X.509 того, кто его подписал. В подписанном пакете могут также содержаться цепочки сертификатов. Цепочки сертификатов могут использоваться при создании цепочки доверия от сертификата лица, подписавшего пакет, до локально хранимых доверенных сертификатов.

Термин	Определение
PKCS12	Стандарт шифрования с открытым ключом #12 - В этом стандарте содержатся требования к синтаксису для хранения зашифрованных объектов на диске. В этом формате поддерживается хранилище ключей.
Хранилище ключей пакета	Хранилище сертификатов и ключей, к которым можно обращаться с помощью средств для работы с пакетами.

Управление сертификатом

Перед созданием подписанного пакета необходимо создать хранилище ключей пакета. Сертификаты в этом хранилище ключей сертификата хранятся в виде объектов. В хранилище ключей пакета есть два типа объектов:

Доверенный сертификат
: Доверенный сертификат, содержащий один сертификат открытого ключа, принадлежащий другой организации. Доверенный сертификат называется таковым, поскольку владелец хранилища ключей считает, что открытый ключ в сертификате принадлежит организации (лицу), указанному в поле “тема” (владелец) сертификата. Издатель сертификата делает его доверенным, подписывая сертификат.

  При проверке подписей и подключении к защищенному серверу (по протоколу SSL) используются доверенные сертификаты.

Пользовательский ключ
: Пользовательский ключ, содержащий секретную информацию ключа шифрования. Эта информация хранится в защищенном формате для предотвращения несанкционированного использования. Пользовательский ключ состоит из секретного ключа пользователя и соответствующего ему сертификата открытого ключа.

  Пользовательские ключи используются при создании подписанного пакета.

По умолчанию хранилище ключей расположено в каталоге `/var/sadm/security`. Отдельные пользователи могут также иметь собственные хранилища ключей, которые по умолчанию располагаются в каталоге `$HOME/.pkg/security`.

На диске хранилище ключей может иметь два формата: формат одного файла и формат нескольких файлов. При использовании формата нескольких файлов объекты хранятся в нескольких файлах. Каждый тип объекта хранится в отдельном файле. Все эти файлы

должны быть зашифрованы с помощью одного пароля. При использовании хранилища в формате одного файла все объекты хранятся в одном файле файловой системы.

Основным средством для управления сертификатами и хранилищем ключей пакета является команда `pkgadm`. Более распространенные задачи, которые используются при управлении хранилищем ключей пакета, представлены в подразделах ниже.

Добавление доверенных сертификатов в хранилище ключей пакета

Доверенный сертификат можно добавить в хранилище пакета с помощью команды `pkgadm`. Сертификат может иметь формат PEM или DER. Пример:

```
$ pkgadm addcert -t /tmp/mytrustedcert.pem
```

В этом примере сертификат в формате PEM имеет имя `mytrustedcert.pem` и добавлен в хранилище ключей пакета.

Добавление в хранилище пакета пользовательского сертификата и секретного ключа

С помощью команды `pkgadm` невозможно создание пользовательских сертификатов или секретных ключей. Пользовательские сертификаты и секретные ключи обычно можно получить от центра сертификации, например Verisign. Их также можно создать локально в виде самоподписанных сертификатов. После получения ключа и сертификата их можно импортировать в хранилище ключей пакета с помощью команды `pkgadm`. Пример:

```
pkgadm addcert -n myname -e /tmp/myprivkey.pem /tmp/mypubcert.pem
```

В этом примере используются следующие параметры:

`-n` *myname*	Указывает организацию или лицо (*myname*) в хранилище данных пакета, над которым необходимо выполнять действие. Организация *myname* становится псевдонимом, под которым хранятся объекты.
`-e /tmp/`*myprivkey.pem*	Указывает файл, содержащий секретный ключ. В этом случае этот файл - *myprivkey.pem*, который расположен в каталоге `/tmp`.
`/tmp/`*mypubcert.pem*	Указывает сертификат формата PEM, который имеет имя *mypubcert.pem* .

Проверка содержимого хранилища ключей пакета

Команда `pkgadm` также используется для просмотра содержимого хранилища ключей пакета. Пример:

```
$ pkgadm listcert
```

С помощью этой команды производится отображение доверенных сертификатов и секретных ключей, которые находятся в хранилище ключей пакета.

Удаление из хранилища пакета пользовательских сертификатов и секретных ключей

Чтобы удалить доверенные сертификаты и закрытые ключи из хранилища ключей пакета, можно использовать команду `pkgadm`.

При удалении пользовательских сертификатов необходимо указывать псевдоним пары сертификат/ключ. Пример:

```
$ pkgadm removecert -n myname
```

Псевдоним сертификата - это его общее имя, которое может быть указано с помощью команды `pkgadm listcert`. Например, с помощью этой команды производится удаление доверенного сертификата с именем Trusted CA Cert 1:

```
$ pkgadm removecert -n "Trusted CA Cert 1"
```

Примечание – Если под одним псевдонимом хранятся доверенный сертификат и пользовательский сертификат, при использовании параметра `-n` оба они удаляются.

Создание подписанных пакетов

Процесс создания подписанного пакета включает в себя три основных этапа:

1. Создание неподписанного пакета в формате каталога.
2. Импорт подписанного сертификата, сертификатов центров сертификации и секретного ключа в хранилище ключей пакета.
3. Подписывание пакета из Действия 1 с помощью сертификата из Действия 2.

Примечание – Средства пакетирования не создают сертификаты. Эти сертификаты необходимо получать от центра сертификации, например Verisign или Thawte.

Все действия по созданию подписанных пакетов описаны в следующих процедурах.

▼ Создание неподписанного пакета в формате каталога

Процедура создания неподписанных пакетов в формате каталогов та же, что и при создании стандартного пакета, как описано в этом руководстве ранее. Ниже приводится процедура создания неподписанного пакета в формате каталога. Если необходима дополнительная информация, см. предыдущие разделы, посвященные сборке пакетов.

1 Создание файла `pkginfo`.

Базовое содержимое файла `pkginfo` должно быть следующим:

```
PKG=SUNWfoo
BASEDIR=/
NAME=My Test Package
ARCH=sparc
VERSION=1.0.0
CATEGORY=application
```

2 Создание файла `prototype`.

Базовое содержимое файла `prototye` должно быть следующим:

```
$cat prototype
i pkginfo
d none usr 0755 root sys
d none usr/bin 0755 root bin
f none usr/bin/myapp=/tmp/myroot/usr/bin/myapp 0644 root bin
```

3 Укажите список содержимого исходного каталога объектов.

Пример:

```
$ ls -lR /tmp/myroot
```

В результате должно получиться следующее:

```
/tmp/myroot:
total 16
drwxr-xr-x   3 abc      other        177 Jun  2 16:19 usr

/tmp/myroot/usr:
total 16
drwxr-xr-x   2 abc      other        179 Jun  2 16:19 bin

/tmp/myroot/usr/bin:
total 16
-rw-------   1 abc      other       1024 Jun  2 16:19 myapp
```

4 Создайте неподписанный пакет.

```
pkgmk -d ‘pwd‘
```

В результате должно получиться следующее:

```
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter <PSTAMP> set to "syrinx20030605115507"
WARNING: parameter <CLASSES> set to "none"
## Attempting to volumize 3 entries in pkgmap.
part  1 -- 84 blocks, 7 entries
## Packaging one part.
/tmp/SUNWfoo/pkgmap
/tmp/SUNWfoo/pkginfo
/tmp/SUNWfoo/reloc/usr/bin/myapp
## Validating control scripts.
## Packaging complete.
```

В текущем каталоге теперь размещен пакет.

▼ Импорт сертификатов в хранилище ключей пакета

Сертификат и секретный ключ для импорта должны быть представлены в виде сертификата X.509 в кодировке PEM или DER и секретного ключа. Кроме того, перед подписыванием пакета необходимо выполнить импорт всех промежуточных сертификатов или их цепочек, связывающих подписанные сертификаты с центром сертификации.

Примечание – Каждый центр сертификации может издавать сертификаты в различных форматах. Чтобы извлечь сертификаты и секретные ключи из файла PKCS12 и вставить в файл X.509, зашифрованный с помощью алгоритма PEM (который можно импортировать в хранилище ключей пакета), воспользуйтесь таким бесплатным средством преобразования, как OpenSSL.

Если секретный ключ зашифрован (как обычно и должно быть), выдается запрос на ввод пароля. Кроме того, выдается запрос на ввод пароля для защиты хранилища ключей пакета, которое будет получено в результате. Пароль можно не указывать, однако при этом хранилище ключей пакета зашифровано не будет.

В следующей процедуре описывается способ импорта сертификата с помощью команды `pkgadm` после того, как сертификат будет в нужном формате.

1 Выполните импорт всех сертификатов центра сертификации, которые находятся в файле сертификата X.509 с кодировкой PEM или DER.

Например, для импорта всех сертификатов центра сертификации, которые находятся в файле `ca.pem`, необходимо ввести следующую команду:

```
$ pkgadm addcert -k ~/mykeystore -ty ca.pem
```

В результате должно получиться следующее:

```
Trusting certificate <VeriSign Class 1 CA Individual \
Subscriber-Persona Not Validated>
Trusting certificate </C=US/O=VeriSign, Inc./OU=Class 1 Public \
Primary Certification Authority
Type a Keystore protection Password.
Press ENTER for no protection password (not recommended):
For Verification: Type a Keystore protection Password.
Press ENTER for no protection password (not recommended):
Certificate(s) from <ca.pem> are now trusted
```

Чтобы импортировать ключ подписи в хранилище ключей пакета, необходимо указать псевдоним, который будет использоваться позднее при подписании пакета. Этот псевдоним также можно использовать в случае, если необходимо удалить ключ из хранилища ключей пакета.

Например, чтобы импортировать ключ подписи из файла `sign.pem`, необходимо ввести следующее:

```
$ pkgadm addcert -k ~/mykeystore -n mycert sign.pem
```

В результате должно получиться следующее:

```
Enter PEM passphrase:
Enter Keystore Password:
Successfully added Certificate <sign.pem> with alias <mycert>
```

2 **Убедитесь, что сертификат импортирован в хранилище ключей пакета.**

Например, чтобы просмотреть сертификаты в хранилище ключей, созданном в ходе предыдущего действия, введите следующее:

```
$ pkgadm listcert -k ~/mykeystore
```

▼ Подписывание пакета

После того, как сертификаты импортированы в хранилище пакета, можно подписать пакет. Фактически подписание пакета производится с помощью команды `pkgtrans`.

- **Подпишите пакет с помощь команды `pkgtrans`. Укажите расположение подписанного пакета, а также псевдоним ключа для подписания.**

Например, используя примеры процедур выше, для создания подписанного пакета с именем `SUNWfoo.signed` необходимо ввести следующее:

```
$ pkgtrans -g -k ~/mykeystore -n mycert . ./SUNWfoo.signed SUNWfoo
```

В результате выполнения этой команды получится следующее:

```
Retrieving signing certificates from keystore </home/user/mykeystore>
Enter keystore password:
Generating digital signature for signer <Test User>
Transferring <SUNWfoot> package instance
```

Подписанный пакет создан в файле `SUNWfoo.signed`. Пакет имеет формат потока пакета. Подписанный пакет можно скопировать на веб-сайт и установить с помощью команды `pkgadd`, указав URL-адрес.

◆ ◆ ◆ ГЛАВА 4

Проверка и запись пакета

В данной главе описано, как проверить целостность пакета и записать его на распространяемый носитель (дискету или компакт-диск).

Ниже приведен общий перечень вопросов, рассмотренных в данной главе.

Проверка и запись пакета (карта задач)

В приведенной ниже таблице описаны действия, которые необходимо выполнить для проверки целостности пакета и переноса его на распространяемый носитель.

ТАБЛИЦА 4–1 Проверка и запись пакета (карта задач)

Задача	Описание	Инструкции
1. Выполните сборку пакета	Выполните сборку пакета на диске.	Глава 2, «Сборка пакета»
2. Установите пакет	Сделайте пробную установку пакета и убедитесь, что при установке нет ошибок.	« Как устанавливать пакеты на независимой системе или сервере» на стр. 95
3. Проверьте целостность пакета	Проверьте целостность пакета с помощью команды pkgchk.	«Как проверить целостность пакета» на стр. 97

ТАБЛИЦА 4–1 Проверка и запись пакета (карта задач) *(Продолжение)*

Задача	Описание	Инструкции
4. Соберите дополнительную информацию о пакете	*Необязательная задача.* С помощью команд `pkginfo` и `pkgparam` выполните проверку конкретного пакета.	«Отображение дополнительной информации об установленных пакетах» на стр. 98
5. Удалите установленный пакет	Удалите установленный пакет из системы с помощью команды `pkgrm`.	«Как удалить пакет» на стр. 104
6. Запишите пакет на распространяемый носитель	Перенесите пакет (в формате пакета) на распространяемый носитель с помощью команды `pkgtrans`.	«Как записать пакет на распространяемый носитель» на стр. 105

Установка пакетов ПО

Пакеты программного обеспечения устанавливаются с помощью команды `pkgadd`. Данная команда переносит содержимое пакета программного обеспечения с распространяемого носителя или из каталога и устанавливает его на целевую систему.

В данном разделе приводятся базовые инструкции по установке пакета с целью проверки правильности установки.

База данных устанавливаемого ПО

Информация о всех пакетах, установленных в системе, хранится в базе данных устанавливаемого ПО. Она содержит запись для каждого объекта пакета, а также такую информацию, как имя компонента, его местоположение и тип. Каждая запись содержит данные о пакете, к которому принадлежит тот или иной компонент, данные о других пакетах, которые могут ссылаться на данный компонент, а также такую информацию, как имя пути, местоположение компонента и его тип. Записи автоматически добавляются и удаляются с помощью команд `pkgadd` и `pkgrm`. С помощью команд `pkgchk` и `pkginfo` можно просмотреть информацию, содержащуюся в базе данных.

С каждым компонентом пакета связаны два типа информации. Атрибутивная информация описывает сам компонент. Примерами атрибутивной информации служат права доступа к компоненту, идентификатор владельца и идентификатор группы. Информация о содержимом описывает содержимое компонента, например размер файла и время последнего изменения.

База данных устанавливаемого ПО отслеживает состояние пакета. Пакет может быть полностью установлен (успешно завершен процесс установки) или установлен частично (процесс установки не был успешно завершен).

Когда пакет установлен частично, части этого пакета могли быть установлены до того, как процесс установки был прерван. Таким образом, часть пакета установлена и запись

об этом внесена в базу данных, а часть нет. При переустановке пакета выдается запрос на начало с того места, где установка была прервана, поскольку команда pkgadd имеет доступ к базе данных и может определить, какие части уже были установлены. Кроме того, установленные части пакета можно удалить на основе информации, содержащейся в базе данных устанавливаемого ПО, с помощью команды pkgrm.

Взаимодействие с командой pkgadd

Если команда pkgadd сталкивается с проблемой, то первым делом она проверяет файл администрирования установки на наличие соответствующих инструкций. (Для получения дополнительной информации см. страницу admin(4).) Если инструкции отсутствуют, или если соответствующий параметр в файле администрирования установлен на значение ask (спросить), команда pkgadd выдает сообщение с описанием проблемы и предлагает ввести ответ. Обычно запрос выглядит так: Do you want to continue with this installation? (Продолжить установку?). На этот запрос необходимо ответить yes (да), no (нет) или quit (выйти).

Если указано более одного пакета, то в случае ответа no установка текущего пакета будет остановлена, однако команда pkgadd продолжит установку других пакетов. Ответ выйти указывает команде pkgadd, что необходимо прекратить установку всех пакетов.

Установка пакетов на независимых системах или серверах в однородной вычислительной среде

В данном разделе описывается, как производить установку пакетов на независимых системах или серверах в однородной вычислительной среде.

▼ Как устанавливать пакеты на независимой системе или сервере

1. **Выполните сборку пакета.**

   См. раздел «Сборка пакета» на стр. 48.

2. **Войдите в систему как суперпользователь.**

3. **Добавьте пакет программного обеспечения в систему.**

   # **pkgadd -d** *device-name* [*pkg-abbrev* . . .]

-d *device-name*	Указывает расположение пакета. Обратите внимание, что *device-name* может быть как полным именем пути каталога, так и идентификатором в случае использования ленточного накопителя, дискеты или съемного диска.
pkg-abbrev	Имя одного или нескольких пакетов (разделенных пробелом), которые следует добавить. Если имя не указано, команда pkgadd устанавливает все имеющиеся пакеты.

Пример 4–1 Установка пакетов на независимых системах или серверах

Для установки пакета ПО с именем pkgA с ленточного носителя с именем /dev/rmt/0 необходимо ввести следующую команду:

```
# pkgadd -d /dev/rmt/0 pkgA
```

Можно устанавливать несколько пакетов одновременно, отделяя их имена друг от друга с помощью пробела:

```
# pkgadd -d /dev/rmt/0 pkgA pkgB pkgC
```

Если не указано имя устройства, на котором расположен пакет, команда проверяет буферный каталог, установленный по умолчанию (/var/spool/pkg). Если пакета там нет, установка завершается сбоем.

См. также Для перехода к следующей задаче откройте ссылку «Как проверить целостность пакета» на стр. 97.

Проверка целостности пакета

С помощью команды pkgchk можно проверить целостность пакетов независимо от того, установлены ли они в системе или находятся в пакетном формате (готовы к установке командой pkgadd). Команда подтверждает структуру пакета, установленных файлов и каталогов или отображает информацию об объектах пакета. Команда pkgchk может вывести список или проверить следующее:

- Сценарии установки пакета.
- Содержимое и/или атрибуты объектов, установленных в системе.
- Содержимое помещенного в буфер, удаленного пакета.
- Содержимое и/или атрибуты объектов, описанных в указанном файле pkgmap.

Для получения дополнительной информации см. страницу pkgchk(1M).

Команда pkgchkвыполняет два типа проверки. Она проверяет атрибуты файла (права доступа и принадлежность файла, а также старшие и младшие номера блочных или

символьных устройств) и содержимое файла (размер, контрольная сумма и дата изменения). По умолчанию команда проверяет как атрибуты, так и содержимое файла.

Кроме того, команда `pkgchk` сравнивает атрибуты файла и содержимое установленного пакета с базой данных устанавливаемого ПО. Записи о пакете могли измениться со времени установки. Например, другой пакет мог изменить какой-то компонент пакета. Подобные изменения отражаются в базе данных.

▼ Как проверить целостность пакета

1 Установите пакет.

См. раздел « Как устанавливать пакеты на независимой системе или сервере» на стр. 95

2 Проверьте целостность пакета.

```
# pkgchk [-v] [-R root-path] [pkg-abbrev...]
```

`-v`	Выводит список файлов по мере их обработки.
`-R` *root-path*	Указывает расположение корневой файловой системы клиента.
pkg-abbrev	Имя одного или нескольких пакетов (разделенных пробелом), которые следует проверить. Если имя не указано, команда `pkgchk` проверяет все имеющиеся пакеты.

Пример 4–2 Проверка целостности пакета

В данном примере показана команда, которую следует использовать для проверки целостности установленного пакета.

```
$ pkgchk pkg-abbrev
$
```

При наличии ошибок команда `pkgchk` выводит их на печать. В противном случае данные не выводятся, и возвращается код выхода 0. Если сокращение пакета не указано, осуществляется проверка всех пакетов в системе.

При необходимости можно использовать параметр `-v` для вывода на печать списка файлов пакета при отсутствии в нем ошибок. Пример:

```
$ pkgchk -v SUNWcadap
/opt/SUNWcadap
/opt/SUNWcadap/demo
/opt/SUNWcadap/demo/file1
/opt/SUNWcadap/lib
```

```
/opt/SUNWcadap/lib/file2
/opt/SUNWcadap/man
/opt/SUNWcadap/man/man1
/opt/SUNWcadap/man/man1/file3.1
/opt/SUNWcadap/man/man1/file4.1
/opt/SUNWcadap/man/windex
/opt/SUNWcadap/srcfiles
/opt/SUNWcadap/srcfiles/file5
/opt/SUNWcadap/srcfiles/file6
$
```

Если требуется проверить пакет, установленный в корневой файловой системе клиента, используйте следующую команду:

```
$ pkgchk -v -R root-path pkg-abbrev
```

См. также Для перехода к следующей задаче откройте ссылку «Как получить информацию с помощью команды pkginfo» на стр. 103.

Отображение дополнительной информации об установленных пакетах

Для отображения информации об установленных пакетах можно использовать еще две команды:

- Команда `pkgparam` отображает значения параметров.
- Команда `pkginfo` отображает информацию из базы данных устанавливаемого ПО.

Команда `pkgparam`

Команда `pkgparam` позволяет вывести на экран значения параметров, указанных в командной строке. Эти значения извлекаются либо из файла `pkginfo` конкретного пакета, либо из указанного вами файла. В каждой строке отображается одно значение параметра. Можно выводить на экран только значения или параметры и их значения.

▼ Как получить информацию с помощью команды `pkgparam`

1 Установите пакет.

См. раздел « Как устанавливать пакеты на независимой системе или сервере» на стр. 95

2 Отобразите дополнительную информацию о пакете.

```
# pkgparam [-v] pkg-abbrev [param...]
```

`-v`	Отображает имя параметра и его значение.
pkg-abbrev	Имя конкретного пакета.
param	Указывает один или несколько параметров, значения которых отображаются.

Пример 4–3 Получение информации с помощью команды pkgparam

Например, для отображения только значений используйте следующую команду.

```
$ pkgparam SUNWcadap
none
/opt
US/Mountain
/sbin:/usr/sbin:/usr/bin:/usr/sadm/install/bin
/usr/sadm/sysadm
SUNWcadap
Chip designers need CAD application software to design abc
chips.  Runs only on xyz hardware and is installed in the usr
partition.
system
release 1.0
SPARC
venus990706083849
SUNWcadap
/var/sadm/pkg/SUNWcadap/save
Jul 7 1999 09:58
$
```

Для отображения параметров и их значений используйте следующую команду:

```
$ pkgparam -v SUNWcadap
pkgparam -v SUNWcadap
CLASSES='none'
BASEDIR='/opt'
TZ='US/Mountain'
PATH='/sbin:/usr/sbin:/usr/bin:/usr/sadm/install/bin'
OAMBASE='/usr/sadm/sysadm'
PKG='SUNWcadap'
NAME='Chip designers need CAD application software to design abc chips.
Runs only on xyz hardware and is installed in the usr partition.'
CATEGORY='system'
VERSION='release 1.0'
```

```
ARCH='SPARC'
PSTAMP='venus990706083849'
PKGINST='SUNWcadap'
PKGSAV='/var/sadm/pkg/SUNWcadap/save'
INSTDATE='Jul 7 1999 09:58'
$
```

Для отображения значения конкретного параметра используйте следующий формат:

```
$ pkgparam SUNWcadap BASEDIR
/opt
$
```

Для получения дополнительной информации см. страницу pkgparam(1).

См. также Для перехода к следующей задаче воспользуйтесь ссылкой «Как удалить пакет» на стр. 104.

Команда pkginfo

С помощью команды pkginfo можно отобразить информацию об установленных пакетах. У данной команды есть несколько параметров, которые позволяют настроить формат и содержимое выводимой на экран информации.

Можно запросить информацию о любом количестве экземпляров пакета.

Вывод команды pkginfo по умолчанию

Когда команда pkginfo выполняется без параметров, она отображает категорию, экземпляр пакета и имена всех пакетов, которые были полностью установлены в системе. Отображение упорядочено по категориям, как показано в следующем примере:

```
$ pkginfo
.
.
.
system      SUNWinst       Install Software
system      SUNWipc        Interprocess Communications
system      SUNWisolc      XSH4 conversion for ISO Latin character sets
application SUNWkcspf      KCMS Optional Profiles
application SUNWkcspg      KCMS Programmers Environment
application SUNWkcsrt      KCMS Runtime Environment
.
.
.
$
```

Настройка формата отображения результатов выполнения команды `pkginfo`

Результаты выполнения команды `pkginfo` могут отображаться в любом из трех форматов: кратком, извлеченном и длинном.

Значением по умолчанию является краткий формат. В этом формате отображается только категория, аббревиатура и полное имя пакета, как показано в разделе «Вывод команды `pkginfo` по умолчанию» на стр. 100.

В извлеченном формате отображаются аббревиатура и имя пакета, архитектура пакета (если есть) и версия пакета (если есть). Для получения выходных данных в извлеченном формате используйте параметр `-x`, как показано в следующем примере:

```
$ pkginfo -x
.
.
.
SUNWipc         Interprocess Communications
                (sparc) 11.8.0,REV=1999.08.20.12.37
SUNWisolc       XSH4 conversion for ISO Latin character sets
                (sparc) 1.0,REV=1999.07.10.10.10
SUNWkcspf       KCMS Optional Profiles
                (sparc) 1.1.2,REV=1.5
SUNWkcspg       KCMS Programmers Environment
                (sparc) 1.1.2,REV=1.5
.
.
.
$
```

При указании параметра `-l` выдаются данные в длинном формате, в котором показана вся имеющаяся информация о пакете:

```
$ pkginfo -l SUNWcadap
   PKGINST:  SUNWcadap
      NAME:  Chip designers need CAD application software to
design abc chips.  Runs only on xyz hardware and is installed
in the usr partition.
  CATEGORY:  system
      ARCH:  SPARC
   VERSION:  release 1.0
   BASEDIR:  /opt
    PSTAMP:  system980706083849
  INSTDATE:  Jul 7 1999 09:58
    STATUS:  completely installed
     FILES:     13 installed pathnames
                 6 directories
```

```
               3 executables
            3121 blocks used (approx)
$
```

Описание параметров для длинного формата команды `pkginfo`

В приведенной ниже таблице описаны параметры пакета, которые могут быть отображены для каждого пакета. Параметр и его значение отображаются только в том случае, если параметр имеет назначенное ему значение.

ТАБЛИЦА 4–2 Параметры пакета

Параметр	Описание
ARCH	Архитектура, поддерживаемая данным пакетом.
BASEDIR	Базовый каталог, в котором размещен пакет ПО (отображается если пакет является перемещаемым).
CATEGORY	Категория или категории программного обеспечения, частью которых является этот пакет (например, `system` или `application`).
CLASSES	Список классов, определенных для пакета. Порядок следования классов в этом списке определяет последовательность их установки. Классы, приведенные в начале списка, будут установлены первыми (из расчета "носитель за носителем"). Этот параметр может быть изменен сценарием request.
DESC	Текст с описанием пакета.
EMAIL	Адрес электронной почты, используемый для запросов от пользователей.
HOTLINE	Информация о том, как получить оперативную помощь по пакету.
INTONLY	Указывает, что пакет следует устанавливать в интерактивном режиме только при наличии любого не пустого значения.
ISTATES	Список разрешенных состояний выполнения для установки пакета (например, `S s 1`).
MAXINST	Максимальное количество экземпляров пакета, которые могут одновременно находиться на компьютере. По умолчанию разрешен только один экземпляр пакета.
NAME	Имя пакета - обычно текст, описывающий аббревиатуру пакета.
ORDER	Список классов, определяющий порядок, в котором классы будут переноситься на носитель. Используется командой `pkgmk` при создании пакета. Классы, не определенные в этом параметре, размещаются на носителе в соответствии со стандартными процедурами сортировки.
PKGINST	Аббревиатура устанавливаемого пакета.

ТАБЛИЦА 4–2 Параметры пакета *(Продолжение)*

Параметр	Описание
PSTAMP	Производственная марка пакета.
RSTATES	Список разрешенных состояний выполнения для удаления пакета (например, `S s 1`).
ULIMIT	Если этот параметр установлен, он передается в качестве аргумента команде `ulimit`, которая определяет максимальный размер файла в ходе установки. Применимо только к файлам, созданным с помощью процедурных сценариев.
VENDOR	Имя поставщика пакета программного обеспечения.
VERSION	Версия пакета.
VSTOCK	Инвентарный номер, предоставленный поставщиком.

Для получения дополнительной информации о команде `pkginfo` см. справочную страницу `pkginfo(1)`.

▼ Как получить информацию с помощью команды `pkginfo`

1 Установите пакет.

См. раздел « Как устанавливать пакеты на независимой системе или сервере» на стр. 95

2 Отобразите дополнительную информацию о пакете.

```
# pkginfo [-x | -l] [pkg-abbrev]
```

`-x`	Отображает информацию о пакете в извлеченном формате.
`-l`	Отображает информацию о пакете в длинном формате.
pkg-abbrev	Имя конкретного пакета. Если имя не указано, команда `pkginfo` отображает информацию обо всех установленных пакетах в формате по умолчанию.

Дополнительные сведения

Что делать дальше

Для перехода к следующей задаче откройте ссылку «Как удалить пакет» на стр. 104.

Удаление пакета

Поскольку команда `pkgrm` обновляет информацию в базе данных ПО, для удаления пакета важно использовать именно команду `pkgrm`, а не команду `rm`. Например, команду `rm` можно использовать для удаления двоичного исполняемого файла, но это не то же самое, что использование команды `pkgrm` для удаления пакета ПО, содержащего двоичный исполняемый файл. Использование команды `rm` для удаления файлов пакета приведет к повреждению базы данных ПО. (Если действительно требуется удалить только один файл, можно использовать команду `removef`, которая корректно обновит базу данных ПО.

▼ Как удалить пакет

1 **Войдите в систему как суперпользователь.**

2 **Удалите установленный пакет.**

```
# pkgrm pkg-abbrev ...
```

pkg-abbrev	Имя одного или нескольких пакетов (разделенных пробелом). Если имя не указано, команда `pkgrm` удаляет все имеющиеся пакеты.

3 **Проверьте, что пакет был успешно удален. Используйте для этого команду** `pkginfo`**.**

```
$ pkginfo | egrep pkg-abbrev
```

Если пакет *pkg-abbrev* установлен, команда `pkginfo` возвращает строку с информации о нем. В противном случае команда `pkginfo` возвращает системную подсказку.

Запись пакета на распространяемый носитель

Команда `pkgtrans` перемещает пакеты и производит преобразование формата пакета. Для выполнения преобразований устанавливаемого пакета можно использовать команду `pkgtrans`. При ее выполнении можно совершить следующие преобразования:

- Преобразование формата файловой системы в формат потока данных
- Преобразование формата потока данных в формат файловой системы
- Преобразование формата одной файловой системы в формат другой файловой системы

▼ Как записать пакет на распространяемый носитель

1 **Выполните сборку пакета путем создания пакета в формате каталога, если этого не было сделано ранее.**

Для получения дополнительной информации см. раздел «Как собрать пакет» на стр. 49.

2 **Установите пакет, чтобы проверить, что установка выполняется корректно.**

См. раздел « Как устанавливать пакеты на независимой системе или сервере» на стр. 95

3 **Проверьте целостность пакета**

При необходимости см. разделы «Как проверить целостность пакета» на стр. 97, «Как получить информацию с помощью команды `pkginfo`» на стр. 103 и «Как получить информацию с помощью команды `pkgparam`» на стр. 98.

4 **Удалите установленный пакет из системы.**

См. раздел «Как удалить пакет» на стр. 104.

5 **Запишите пакет (в формате пакета) на распространяемый носитель.**

Для выполнения базового преобразования выполните следующую команду:

```
$ pkgtrans device1 device2 [pkg-abbrev...]
```

device1	Имя устройства, на котором в данный момент расположен пакет.
device2	Имя устройства, на которое будет записан преобразованный пакет.
[*pkg-abbrev*]	Одна или несколько аббревиатур пакета.

Если имена пакета не указаны, то все пакеты, находящиеся на устройстве *device1*, преобразуются и записываются на устройство *device2*.

Примечание – Если на устройстве *device1* находится более одного экземпляра пакета, необходимо использовать идентификатор экземпляра пакета. Описание идентификатора пакета содержится на веб-странице «Определение экземпляра пакета» на стр. 29. Если на устройстве *device2* уже имеется экземпляр преобразуемого пакета, то команда `pkgtrans` не выполняет преобразование. Для перезаписи всех имеющихся экземпляров на устройстве назначения можно использовать параметр `-o`команды `pkgtrans` и параметр `-n` для создания нового экземпляра, если на этом устройстве уже существуют экземпляры пакета. Обратите внимание, что данная проверка неприменима, если устройство *device2* поддерживает формат потока данных.

Дополнительные сведения

Что делать дальше

К данному моменту мы завершили этапы, необходимые для разработки, сборки, проверки и записи пакета. Для изучения практических примеров см. Глава 5, «Практические примеры создания пакета». Дополнительные идеи по разработке пакетов приведены в Глава 6, «Дополнительные методы создания пакетов».

◆ ◆ ◆ ГЛАВА 5

Практические примеры создания пакета

В данной главе приводятся практические примеры случаев создания пакета, например, условная установка объектов, определение количества создаваемых файлов во время выполнения программы и изменение существующего файла данных во время установки и удаления пакета.

Рассмотрение каждого практического примера начинается с описания, затем следует перечень использованных методов создания пакета, описания подходов, использованных в этих методах, а также примеры файлов и сценариев, связанных с практическим примером.

В данной главе будут рассмотрены практические примеры.

Запрос ввода у администратора

В данном практическом примере пакет содержатся три типа объектов. Администратор может выбрать, какой из этих трех типов следует установить и в каком месте эти объекты будут располагаться на целевом компьютере.

Методы

В практическом примере применяются следующие методы:

- Использование параметрических имен путей (переменные в именах путей объекта), применяемых для создания нескольких базовых каталогов

  Для получения информации о параметрических именах пути см. раздел «Параметрические имена путей» на стр. 37.

- Использование сценария `request` для запроса ввода у администратора

  Для получения информации о сценариях `request` см. раздел «Создание сценария `request` » на стр. 68.

- Определение условных значений для установочного параметра.

Подход

Для настройки выборочной установки в этом практическом примере необходимо выполнить следующие задачи.

- Определить класс для каждого типа объекта, который может быть установлен.

  В данном практическом примере тремя типами объекта являются исполняемые файлы пакета, справочные страницы и исполняемые файлы `emacs`. Каждому типу присвоен собственный класс: `bin`, `man` и `emacs` соответственно. Обратите внимание, что в файле `prototype` все файлы объектов принадлежат к одному из этих трех классов.

- Инициализируйте параметр `CLASSES` в файле `pkginfo`, установив для него пустое значение.

  Обычно при определении класса следует внести этот класс в параметр `CLASSES` файла `pkginfo`. В противном случае ни один объект этого класса установлен не будет. В нашем практическом примере параметру первоначально присвоено пустое значение, что означает, что никаких объектов установлено не будет. Параметр `CLASSES` будет изменен сценарием `request` в соответствии с выбором, сделанным администратором. Таким образом параметр `CLASSES` принимает только те типы объектов, которые хочет установить администратор.

  Примечание – Обычно рекомендуется устанавливать для параметров значение по умолчанию. Если в пакете содержатся компоненты, общие для всех трех типов объектов, можно назначить их классу `none` и затем установить параметр `CLASSES` равным `none`.

- Вставьте параметрические имена путей в файл `prototype`.

Сценарий `request` устанавливает для этих переменных среды значение, предоставленное администратором. После этого в ходе установки команда `pkgadd` вычисляет эти переменные среды и определяет, куда установить пакет.

Три переменные среды, используемые в данном примере, установлены на соответствующие значения по умолчанию в файле `pkginfo` и служат следующим целям:

- `$NCMPBIN` определяет местоположение исполняемых файлов объекта
- `$NCMPMAN` определяет местоположение справочных страниц
- `$EMACS` определяет местоположение исполняемых файлов `emacs`

В примере файла `prototype` показано, как определить имена пути объекта при помощи переменных.

- Создайте сценарий `request`, который будет запрашивать администратора, какие части пакета следует установить и где они должны быть размещены.

  Сценарий `request` в данном пакете задает администратору два вопроса:

  - Следует ли установить эту часть пакета?

    Если администратор отвечает "да", то к параметру `CLASSES` добавляется соответствующее имя класса. Например, если администратор решает установить справочные страницы этого пакета, то класс `man` добавляется к параметру `CLASSES`.

  - Если да, где следует разместить эту часть пакета?

    Ответ на этот вопрос присваивается соответствующей переменной среды. В примере со справочной страницей ответ на запрос присваивается переменной `$NCMPMAN`.

  Эти два вопроса повторяются для каждого из трех типов объекта.

  В конце сценария `request` эти параметры становятся доступны установочной среде для выполнения команды `pkgadd` и других сценариев пакета. Сценарий `request` записывает эти определения в файл, предоставленный вызывающей служебной программой. В нашем практическом примере другие сценарии отсутствуют.

  При рассмотрении сценария `request` для данного практического примера обратите внимание, что вопросы создаются средствами проверки данных `ckyorn` и `ckpath`. Для получения дополнительной информации об этих средствах см. `ckyorn(1)` и `ckpath(1)`.

Файлы практических примеров

Файл `pkginfo`

```
PKG=ncmp
NAME=NCMP Utilities
CATEGORY=application, tools
```

```
BASEDIR=/
ARCH=SPARC
VERSION=RELEASE 1.0, Issue 1.0
CLASSES=""
NCMPBIN=/bin
NCMPMAN=/usr/man
EMACS=/usr/emacs
```

Файл prototype

```
i pkginfo
i request
x bin $NCMPBIN 0755 root other
f bin $NCMPBIN/dired=/usr/ncmp/bin/dired 0755 root other
f bin $NCMPBIN/less=/usr/ncmp/bin/less 0755 root other
f bin $NCMPBIN/ttype=/usr/ncmp/bin/ttype 0755 root other
f emacs $NCMPBIN/emacs=/usr/ncmp/bin/emacs 0755 root other
x emacs $EMACS 0755 root other
f emacs $EMACS/ansii=/usr/ncmp/lib/emacs/macros/ansii 0644 root other
f emacs $EMACS/box=/usr/ncmp/lib/emacs/macros/box 0644 root other
f emacs $EMACS/crypt=/usr/ncmp/lib/emacs/macros/crypt 0644 root other
f emacs $EMACS/draw=/usr/ncmp/lib/emacs/macros/draw 0644 root other
f emacs $EMACS/mail=/usr/ncmp/lib/emacs/macros/mail 0644 root other
f emacs $NCMPMAN/man1/emacs.1=/usr/ncmp/man/man1/emacs.1 0644 root other
d man $NCMPMAN 0755 root other
d man $NCMPMAN/man1 0755 root other
f man $NCMPMAN/man1/dired.1=/usr/ncmp/man/man1/dired.1 0644 root other
f man $NCMPMAN/man1/ttype.1=/usr/ncmp/man/man1/ttype.1 0644 root other
f man $NCMPMAN/man1/less.1=/usr/ncmp/man/man1/less.1 0644 inixmr other
```

Сценарий request

```
trap 'exit 3' 15
# determine if and where general executables should be placed
ans=`ckyorn -d y \
-p "Should executables included in this package be installed"
` || exit $?
if [ "$ans" = y ]
then
   CLASSES="$CLASSES bin"
   NCMPBIN=`ckpath -d /usr/ncmp/bin -aoy \
   -p "Where should executables be installed"
   ` || exit $?
fi
# determine if emacs editor should be installed, and if it should
# where should the associated macros be placed
ans=`ckyorn -d y \
-p "Should emacs editor included in this package be installed"
```

```
‘ || exit $?
if [ "$ans" = y ]
then
   CLASSES="$CLASSES emacs"
   EMACS=‘ckpath -d /usr/ncmp/lib/emacs -aoy \
   -p "Where should emacs macros be installed"
   ‘ || exit $?
fi
```

Обратите внимание, что выполнение сценария `request` может завершиться таким образом, что ни один из файлов не останется в файловой системе. Для установки в системе Solaris версий ниже 2.5 и совместимых версий (где нельзя использовать сценарий `checkinstall`), сценарий `request` подходит наилучшим образом для тестирования файловой системы с тем, чтобы обеспечить успешную установку. Если выполнение сценария `request` завершается с кодом 1, произойдет чистый выход из процесса установки.

В данных файлах примеров показано, как использовать параметрические пути для создания нескольких базовых каталогов. Однако предпочтительнее использовать параметр `BASEDIR`, который управляется и проверяется командой `pkgadd`. При использовании нескольких базовых каталогов необходимо особо тщательно следить за установкой нескольких версий и архитектур на одну и ту же платформу.

Создание файла во время установки и сохранение его во время удаления

В данном практическом примере в ходе установки будет создан файл базы данных и его копия сохранена после удаления пакета.

Методы

В практическом примере применяются следующие ниже методы.

- Использование классов и сценариев действий над классами для выполнения особых действий на различных наборах объектов

  Для получения дополнительной информации см. раздел «Создание сценариев действий над классами» на стр. 75.

- Использование файла `space` для сообщения команде `pkgadd` о необходимости дополнительного дискового пространства для корректной установки данного пакета

  Для получения дополнительной информации о файле `space` см. раздел «Резервирование дополнительного места на диске на целевой системе» на стр. 61.

- Использование команды `installf` для установки файла, не определенного в файлах `prototype` и `pkgmap`.

Подход

Чтобы создать файл базы данных в ходе установки и сохранить его копию при удалении, необходимо выполнить следующие ниже задачи.

- Определите три класса.

  Для данного практического примера пакета требуется определение следующих трех классов в параметре `CLASSES`:

  - Стандартный класс `none`, содержащий набор процессов, принадлежащий подкаталогу `bin`.
  - Класс `admin`, содержащий исполняемый файл `config`, а также каталог с файлами данных.
  - Класс `cfgdata`, содержащий каталог.
- Сделайте пакет коллективно перемещаемым.

  Обратите внимание, что в файле `prototype` ни одно из имен путей не начинается с косой черты или с переменной среды. Это означает, что они являются коллективно перемещаемыми.
- Вычислите объем дискового пространства, требуемого для файла базы данных, и создайте файл `space`, который будет поставляться в составе пакета. Данный файл уведомляет команду `pkgadd`, что для пакета требуется дополнительное пространство и указывает его объем.
- Создайте сценарий действия над классом для класса `admin` (`i.admin`).

  Пример сценария инициализирует базу данных, используя файлы данных, принадлежащие классу `admin`. Для выполнения этой задачи сценарий выполняет следующие действия.

  - Копирует исходный файл данных в предназначенное место
  - Создает пустой файл с именем `config.data` и назначает его классу `cfgdata`
  - Выполняет команду `bin/config` (поставляемую с пакетом и уже установленную) для наполнения данными файла базы данных `config.data` с помощью файлов данных, принадлежащих классу `admin`
  - Выполняет команду `installf -f` для завершения установки файла `config.data`

  Никакие особые действия для класса `admin` во время удаления пакета не требуются, поэтому сценарий действия над классом во время удаления не создается. Это означает, что все файлы и каталоги в классе `admin` удаляются из системы.
- Создайте сценарий действия над классом для класса `cfgdata` (`r.cfgdata`).

  Сценарий удаления создает копию файла базы данных перед его удалением. Никакие особые действия для этого класса во время установки не требуется, поэтому сценарий действия над классом во время установки не нужен.

Помните, что в качестве входных данных для сценария удаления выступает список имен путей, которые необходимо удалить. Имена путей всегда идут в обратном алфавитном порядке. Сценарий удаления копирует файлы в каталог с именем `$PKGSAV`. После обработки всех имен путей сценарий возвращается назад и удаляет все каталоги и файлы, связанные с классом `cfgdata`.

Результатом выполнения этого сценария удаления является копирование файла `config.data` в каталог `$PKGSAV` и последующее удаление файла `config.data` и каталога с данными.

Файлы практических примеров

Файл `pkginfo`

```
PKG=krazy
NAME=KrAzY Applications
CATEGORY=applications
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1
CLASSES=none cfgdata admin
```

Файл `prototype`

```
i pkginfo
i request
i i.admin
i r.cfgdata
d none bin 555 root sys
f none bin/process1 555 root other
f none bin/process2 555 root other
f none bin/process3 555 root other
f admin bin/config 500 root sys
d admin cfg 555 root sys
f admin cfg/datafile1 444 root sys
f admin cfg/datafile2 444 root sys
f admin cfg/datafile3 444 root sys
f admin cfg/datafile4 444 root sys
d cfgdata data 555 root sys
```

Файл `space`

```
# extra space required by config data which is
# dynamically loaded onto the system
data 500 1
```

Сценарий действия над классом `i.admin`

```
# PKGINST parameter provided by installation service
# BASEDIR parameter provided by installation service
while read src dest
do
   cp $src $dest || exit 2
done
# if this is the last time this script will be executed
# during the installation, do additional processing here.
if [ "$1" = ENDOFCLASS ]
then
# our config process will create a data file based on any changes
# made by installing files in this class; make sure the data file
# is in class ‘cfgdata’ so special rules can apply to it during
# package removal.
   installf -c cfgdata $PKGINST $BASEDIR/data/config.data f 444 root
   sys || exit 2
   $BASEDIR/bin/config > $BASEDIR/data/config.data || exit 2
   installf -f -c cfgdata $PKGINST || exit 2
fi
exit 0
```

Здесь продемонстрирован редкий случай, когда команда `installf` может быть необходима для сценария действия над классом. Поскольку файл `space` использовался для резервирования пространства в конкретной файловой системе, этот новый файл может быть добавлен в пакет несмотря на то, что он не включен в файл `pkgmap`.

Сценарий удаления `r.cfgdata`

```
# the product manager for this package has suggested that
# the configuration data is so valuable that it should be
# backed up to $PKGSAV before it is removed!
while read path
do
# path names appear in reverse lexical order.
   mv $path $PKGSAV || exit 2
   rm -f $path || exit 2
done
exit 0
```

Определение совместимости и зависимостей пакета

Пакет в данном практическом примере использует необязательные информационные файлы для определения совместимости и зависимостей пакета, а также для вывода сообщения об авторских правах во время установки.

Методы

В практическом примере применяются следующие ниже методы.

- Использование файла `copyright`
- Использование файла `compver`
- Использование файла `depend`

Для получения дополнительной информации об этих файлах см. раздел «Создание информационных файлов» на стр. 57.

Подход

Для обеспечения соответствия требованиям, указанным в описании, необходимо выполнить следующие действия.

- Создать файл `copyright`.

  Файл `copyright`содержит сообщение об авторских правах в текстовом формате с кодировкой ASCII. Сообщение, приведенное в файле примера, отображается на дисплее компьютера в ходе установки пакета.

- Создать файл `compver`.

  Файл `pkginfo`, приведенный ниже, определяет версию пакета как версию 3.0. Файл `compver` определяет версию 3.0 как совместимую с версиями 2.3, 2.2, 2.1, 2.1.1, 2.1.3 и 1.7.

- Создать файл `depend`.

  Файлы, перечисленные в файле `depend`, должны уже быть установлены в системе во время установки пакета. Файл примера содержит 11 пакетов, которые уже должны быть установлены в системе во время установки.

Файлы практических примеров

Файл `pkginfo`

```
PKG=case3
NAME=Case Study #3
CATEGORY=application
```

```
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 3.0
CLASSES=none
```

Файл copyright

```
Copyright (c) 1999 company_name
All Rights Reserved.
THIS PACKAGE CONTAINS UNPUBLISHED PROPRIETARY SOURCE CODE OF
company_name.
The copyright notice above does not evidence any
actual or intended publication of such source code
```

Файл compver

```
Version 3.0
Version 2.3
Version 2.2
Version 2.1
Version 2.1.1
Version 2.1.3
Version 1.7
```

Файл depend

```
P acu Advanced C Utilities
Issue 4 Version 1
P cc C Programming Language
Issue 4 Version 1
P dfm Directory and File Management Utilities
P ed Editing Utilities
P esg Extended Software Generation Utilities
Issue 4 Version 1
P graph Graphics Utilities
P rfs Remote File Sharing Utilities
Issue 1 Version 1
P rx Remote Execution Utilities
P sgs Software Generation Utilities
Issue 4 Version 1
P shell Shell Programming Utilities
P sys System Header Files
Release 3.1
```

Изменение файла с помощью стандартных классов и сценариев действий над классами

В данном практическом примере происходит изменение существующего файла в ходе установки пакета с помощью стандартных классов и сценариев действий над классами. Здесь используется один из трех способов изменения файла. Два других способа описаны в разделах «Изменение файла с помощью класса sed и сценария postinstall» на стр. 120 и «Изменение файла с помощью класса build» на стр. 122. Изменяемый файл называется `/etc/inittab`.

Методы

В данном практическом примере показано, как использовать сценарии действий над классами в ходе установки и удаления пакета. Для получения дополнительной информации см. раздел «Создание сценариев действий над классами» на стр. 75.

Подход

Для изменения файла `/etc/inittab` в ходе установки с помощью классов и сценариев действий над классами необходимо выполнить следующие ниже задачи.

- Создать класс.

  Создайте класс с именем `inittab`. Для этого класса необходимо создать сценарий действия над классом в ходе установки и удаления. Определите `inittab` в параметре `CLASSES` файла `pkginfo`.

- Создать файл `inittab`.

  В данном файле содержится информация для записи, которая будет добавлена в файл `/etc/inittab`. Обратите внимание в примере файла `prototype`, что файл `inittab` является членом класса `inittab` и принадлежит к файлу типа `e` (редактируемый).

- Создать сценарий действия над классом во время установки (`i.inittab`).

  Помните, что сценарии действий над классами должны выдавать одинаковые результаты при каждом выполнении. Сценарий действия над классом выполняет следующие действия.

  - Проверяет, не была ли данная запись добавлена ранее
  - Если это так, удаляет все предыдущие версии записи
  - Редактирует файл `inittab` и добавляет строки с комментариями о происхождении этой записи
  - Перемещает временный файл обратно в файл `/etc/inittab`
  - Выполняет команду `init q` после получения признака `ENDOFCLASS`

Обратите внимание, что в данном сценарии установки можно выполнять команду `init q`. При таком подходе состоящий из одной строки сценарий `postinstall` не понадобится.

- Создать сценарий действия над классом во время удаления (`r.inittab`).

  Сценарий удаления очень похож на сценарий установки. Информация, добавленная сценарием установки, удаляется и выполняется команда `init q`.

Данный практический пример более сложный, чем следующий; см. «Изменение файла с помощью класса sed и сценария postinstall» на стр. 120. Требуются три файла вместо двух, и поставляемый файл `/etc/inittab` в действительности является лишь местозаполнителем, содержащим фрагмент записи, которая будет вставлена. Эту запись можно было бы разместить в файле `i.inittab`, однако для этого команде `pkgadd` нужен файл, который будет передан в файл `i.inittab`. Кроме того, процедура удаления в этом случае должна быть размещена в отдельном файле (`r.inittab`). Хотя данный способ работает хорошо, лучше использовать его для случаев, когда требуется очень сложная установка большого количества файлов. См. раздел «Изменение файлов crontab в ходе установки» на стр. 124.

Программа `sed`, используемая в разделе«Изменение файла с помощью класса sed и сценария postinstall» на стр. 120, поддерживает большое количество экземпляров пакета, поскольку комментарий в конце записи `inittab` основан на экземпляре пакета. В практическом примере в разделе «Изменение файла с помощью класса build» на стр. 122 используется более простой подход для редактирования файла `/etc/inittab` в ходе установки.

Файлы практических примеров

Файл `pkginfo`

```
PKG=case5
NAME=Case Study #5
CATEGORY=applications
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1d05
CLASSES=inittab
```

Файл `prototype`

```
i pkginfo
i i.inittab
i r.inittab
e inittab /etc/inittab ? ? ?
```

Сценарий действия над классом при установке `i.inittab`

```
# PKGINST parameter provided by installation service
while read src dest
do
# remove all entries from the table that
# associated with this PKGINST
sed -e "/^[^:]*:[^:]*:[^:]*:[^#]*#$PKGINST$/d" $dest >
/tmp/$$itab ||
exit 2
sed -e "s/$/#$PKGINST" $src >> /tmp/$$itab ||
exit 2
mv /tmp/$$itab $dest ||
exit 2
done
if [ "$1" = ENDOFCLASS ]
then
/sbin/init q ||
exit 2
fi
exit 0
```

Сценарий действия над классом при удалении `r.inittab`

```
# PKGINST parameter provided by installation service
while read src dest
do
# remove all entries from the table that
# are associated with this PKGINST
sed -e "/^[^:]*:[^:]*:[^:]*:[^#]*#$PKGINST$/d" $dest >
/tmp/$$itab ||
exit 2
mv /tmp/$$itab $dest ||
exit 2
done
/sbin/init q ||
exit 2
exit 0
```

Файл `inittab`

```
rb:023456:wait:/usr/robot/bin/setup
```

Изменение файла с помощью класса sed и сценария postinstall

В данном практическом примере происходит изменение файла, существующего на целевом компьютере в ходе установки пакета. Здесь используется один из трех способов изменения файла. Два других способа описаны в разделах«Изменение файла с помощью стандартных классов и сценариев действий над классами» на стр. 117 и «Изменение файла с помощью класса build» на стр. 122. Изменяемый файл называется /etc/inittab.

Методы

В практическом примере применяются следующие ниже методы.

- Использование класса sed

  Для получения дополнительной информации о классе sed см. раздел «Сценарий класса sed » на стр. 80.

- Использование сценария postinstall

  Для получения дополнительной информации об этом сценарии см. раздел «Создание процедурных сценариев » на стр. 73.

Подход

Для изменения файла /etc/inittab в ходе установки с помощью класса sed, необходимо выполнить следующие задачи:

- Добавьте сценарий класса sed в файл prototype.

  Название сценария должно совпадать с именем файла, который предстоит редактировать. В нашем случае редактируемый файл называется /etc/inittab, и сценарий sed также будет называться /etc/inittab. В сценарии sed отсутствуют требования по режиму, владельцу и группе (они представлены в образце файла prototype вопросительными знаками). Файл сценария sed должен относиться к типу e (редактируемый).

- Включите в параметр CLASSES класс sed.

  Как показано в примере ниже, sed является единственным устанавливаемым классом. Однако количество классов может быть любым.

- Создать сценарий действия над классом sed.

  В пакет не может быть включена копия файла /etc/inittabв необходимом виде, поскольку /etc/inittab является динамическим файлом и неизвестно, как он будет выглядеть в период установки пакета. Однако сценарий sed позволяет изменять файл /etc/inittab в ходе установки пакета.

- Создайте сценарий postinstall.

  Необходимо выполнить команду init q для извещения системы о том, что файл /etc/inittab был изменен. Единственное место, где можно выполнить это действие в нашем примере, это сценарий postinstall. При рассмотрении примера сценария postinstall видно, что его единственное предназначение - выполнение команды init q.

Данный подход к изменению файла /etc/inittab в ходе установки имеет один недостаток - приходится поставлять полный сценарий (сценарий postinstall) лишь для того, чтобы выполнить команду init q.

Файлы практических примеров

Файл pkginfo

```
PKG=case4
NAME=Case Study #4
CATEGORY=applications
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1d05
CLASSES=sed
```

Файл prototype

```
i pkginfo
i postinstall
e sed /etc/inittab ? ? ?
```

Сценарий действия над классом sed (/etc/inittab)

```
!remove
# remove all entries from the table that are associated
# with this package, though not necessarily just
# with this package instance
/^[^:]*:[^:]*:[^:]*:[^#]*#ROBOT$/d
!install
# remove any previous entry added to the table
# for this particular change
/^[^:]*:[^:]*:[^:]*:[^#]*#ROBOT$/d
# add the needed entry at the end of the table;
# sed(1) does not properly interpret the '$a'
# construct if you previously deleted the last
# line, so the command
# $a\
```

```
# rb:023456:wait:/usr/robot/bin/setup #ROBOT
# will not work here if the file already contained
# the modification. Instead, you will settle for
# inserting the entry before the last line!
$i\
rb:023456:wait:/usr/robot/bin/setup #ROBOT
```

Сценарий postinstall

```
# make init re-read inittab
/sbin/init q ||
exit 2
exit 0
```

Изменение файла с помощью класса build

В данном практическом примере происходит изменение файла, существующего на целевом компьютере в ходе установки пакета. Здесь используется один из трех способов изменения файла. Два других способа описаны в разделах «Изменение файла с помощью стандартных классов и сценариев действий над классами» на стр. 117 и «Изменение файла с помощью класса sed и сценария postinstall» на стр. 120. Изменяемый файл называется /etc/inittab.

Методы

В данном практическом примере показано, как использовать класс build. Для получения дополнительной информации о классе build см. раздел «Сценарий класса build » на стр. 81.

Подход

Данный подход к изменению /etc/inittab использует класс build. Сценарий класса build выполняется как сценарий интерпретатора команд, и результатом его выполнения становится новая версия исполняемого файла. Другими словами, файл данных /etc/inittab, поставляемый с этим пакетом, будет выполнен, и результатом этого выполнения станет файл /etc/inittab.

Сценарий класса build выполняется в ходе установки и удаления пакета. Параметр install передается файлу, если сценарий выполняется в ходе установки. В сценарии класса build обратите внимание на то, что действия по установке определяются посредством проверки этого параметра.

Для изменения файла /etc/inittab с помощью класса build необходимо выполнить следующие задачи.

- Определить файл build в файле prototype.

  Запись для файла build в файле prototype должна поместить последний в класс build и определить его тип как e (редактируемый). Убедитесь, что параметр CLASSES в файле pkginfo определен как build.

- Создать сценарий класса build.

  Приведенный в примере сценарий класса build делает следующее:

  - Изменяет файл /etc/inittab таким образом, чтобы позволить ему удалить любые существующие изменения этого пакета. Обратите внимание, что имя файла /etc/inittab жестко прописано в команде sed.
  - Если пакет устанавливается, добавляет новую строку к концу файла /etc/inittab. В этой строке дается комментарий с описанием происхождения данной записи.
  - Выполняет команду init q.

Данное решение устраняет недостатки, описанные в практических примерах «Изменение файла с помощью стандартных классов и сценариев действий над классами» на стр. 117 и «Изменение файла с помощью класса sed и сценария postinstall» на стр. 120. Требуется лишь один короткий файл (помимо файлов pkginfo и prototype). Этот файл работает с несколькими экземплярами пакета, поскольку используется параметр PKGINST, а сценарий postinstall не требуется, так как команду init q можно выполнить из сценария класса build.

Файлы практических примеров

Файл pkginfo

```
PKG=case6
NAME=Case Study #6
CATEGORY=applications
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1d05
CLASSES=build
```

Файл prototype

```
i pkginfo
e build /etc/inittab ? ? ?
```

Файл Build

```
# PKGINST parameter provided by installation service
# remove all entries from the existing table that
# are associated with this PKGINST
```

```
sed -e "/^[^:]*:[^:]*:[^:]*:[^#]*#$PKGINST$/d" /etc/inittab ||
exit 2
if [ "$1" = install ]
then
# add the following entry to the table
echo "rb:023456:wait:/usr/robot/bin/setup #$PKGINST" ||
exit 2
fi
/sbin/init q ||
exit 2
exit 0
```

Изменение файлов crontab в ходе установки

В данном практическом примере происходит изменение файлов crontab в ходе установки пакета.

Методы

В практическом примере применяются следующие методы:

- Использование классов и сценариев действий над классами

  Для получения дополнительной информации см. раздел «Создание сценариев действий над классами» на стр. 75.

- Использование команды crontab в сценарии действия над классом.

Подход

Наиболее эффективным способом изменить несколько файлов в ходе установки является определение класса и создание сценария действия над классом. При использовании подхода с классом build необходимо поставлять один сценарий класса build для каждого изменяемого файла crontab. Более общим подходом будет создание класса cron. Для изменения файлов crontab при этом подходе необходимо:

- Определить файлы crontab, которые будут изменяться, в файле prototype.

  Создайте запись в файле prototype для каждого файла crontab, который будет изменен. Для каждого файла определите класс как cron и тип файла как e. Используйте действительное имя изменяемого файла.

- Создать для пакета файлы crontab.

  В этих файлах содержится информация, которую необходимо добавить к одноименным существующим файлам crontab.

- Создать установочный сценарий действия над классом для класса cron.

  Приведенный в примере сценарий класса i.cron делает следующее:

  - Определяет идентификатор пользователя (UID).

    Сценарий i.cron устанавливает значение переменной *user* равной базовому имени выполняемого сценария класса cron. Это имя и является идентификатором пользователя (UID). Например, базовым именем /var/spool/cron/crontabs/root является root, который одновременно представляет собой UID.

  - Выполняет команду crontab с использованием UID и параметра -l.

    Использование параметра -l означает для команды crontab, что следует направить содержимое файла crontab для определенного пользователя на стандартный вывод.

  - Передает результаты выполнения команды crontab в сценарий sed, который удаляет все предыдущие записи, добавленные с помощью данного метода установки.
  - Помещает измененное содержимое во временный файл.
  - Добавляет файл данных для корневого UID (поставленного с пакетом) во временный файл и добавляет метку о происходжении записей.
  - Выполняет команду crontabс тем же UID и передает результаты во временный файл в качестве входных данных.

- Создать удаляющий сценарий действия над классом для класса cron.

  Сценарий r.cron похож на установочный сценарий, за исключением отсутствия процедуры добавления данных в файл crontab.

  Эти процедуры выполняются для каждого файла в классе cron.

Файлы практических примеров

Сценарии i.cron и r.cron, приведенные ниже, выполняются суперпользователем. Изменение суперпользователем файла crontab другого пользователя может привести к непредсказуемым результатам. При необходимости измените следующую запись в каждом сценарии:

```
crontab $user < /tmp/$$crontab ||
```

на

```
su $user -c "crontab /tmp/$$crontab" ||
```

Команда pkginfo

```
PKG=case7
NAME=Case Study #7
CATEGORY=application
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1.0
CLASSES=cron
```

Файл prototype

```
i pkginfo
i i.cron
i r.cron
e cron /var/spool/cron/crontabs/root ? ? ?
e cron /var/spool/cron/crontabs/sys ? ? ?
```

Установочный сценарий действия над классом i.cron

```
# PKGINST parameter provided by installation service
while read src dest
do
user=‘basename $dest‘ ||
exit 2
(crontab -l $user |
sed -e "/#$PKGINST$/d" > /tmp/$$crontab) ||
exit 2
sed -e "s/$/#$PKGINST/" $src >> /tmp/$$crontab ||
exit 2
crontab $user < /tmp/$$crontab ||
exit 2
rm -f /tmp/$$crontab
done
exit 0
```

Сценарий действия над классом при удалении r.cron

```
# PKGINST parameter provided by installation service
while read path
do
user=‘basename $path‘ ||
exit 2
(crontab -l $user |
sed -e "/#$PKGINST$/d" > /tmp/$$crontab) ||
exit 2
crontab $user < /tmp/$$crontab ||
exit 2
```

```
rm -f /tmp/$$crontab
done
exit
```

`crontab` Файл #1

```
41,1,21 * * * * /usr/lib/uucp/uudemon.hour > /dev/null
45 23 * * * ulimit 5000; /usr/bin/su uucp -c
"/usr/lib/uucp/uudemon.cleanup" >
/dev/null 2>&1
11,31,51 * * * * /usr/lib/uucp/uudemon.poll > /dev/null
```

`crontab` Файл #2

```
0 * * * 0-6 /usr/lib/sa/sa1
20,40 8-17 * * 1-5 /usr/lib/sa/sa1
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A
```

Примечание – Если при изменении группы файлов общий размер файлов увеличится более чем на 10 КБ, создайте файл `space` для того, чтобы команда `pkgadd` приняла во внимание это увеличение. Для получения дополнительной информации о файле `space` см. раздел «Резервирование дополнительного места на диске на целевой системе» на стр. 61.

Установка и удаление драйвера с помощью процедурных сценариев

Данный пакет производит установку драйвера.

Методы

В практическом примере применяются следующие методы:

- Установка и загрузка драйвера с помощью сценария `postinstall`
- Выгрузка драйвера с помощью сценария `preremove`

Для получения дополнительной информации об этих сценариях см. раздел «Создание процедурных сценариев » на стр. 73.

Подход

- Создайте сценарий `request`.

Сценарий `request` определяет место, куда администратор хочет установить объекты драйвера, задавая администратору вопросы и назначая ответы параметру `$KERNDIR`.

Сценарий заканчивается подпрограммой, делающей два параметра - `CLASSES` и `KERNDIR` - доступными для среды установки и для сценария`postinstall`.

- Создайте сценарий `postinstall`.

  Сценарий `postinstall` производит непосредственную установку драйвера. Сценарий выполняется после того, как будут установлены два файла: `buffer` и `buffer.conf`. Файл `postinstall`, представленный в данном примере, выполняет следующие действия:

  - Использует команду `add_drv` для загрузки драйвера в систему.
  - Создает ссылку для устройства с помощью команды `installf`.
  - Завершает установку с помощью команды `installf -f`.
  - Создает сценарий `preremove`.

    Сценарий `preremove` использует команду `rem_drv` для выгрузки драйвера из системы, а затем удаляет ссылку `/dev/buffer0`.

Файлы практических примеров

Файл `pkginfo`

```
PKG=bufdev
NAME=Buffer Device
CATEGORY=system
BASEDIR=/
ARCH=INTEL
VERSION=Software Issue #19
CLASSES=none
```

Файл `prototype`

Для установки драйвера в ходе установки пакета необходимо включить объект и файлы настройки драйвера в файл `prototype`.

В данном примере исполняемый модуль драйвера называется `buffer`. Команда `add_drv` работает с этим файлом. Ядро системы использует файл настройки `buffer.conf` для помощи в настройке драйвера.

```
i pkginfo
i request
i postinstall
i preremove
f none $KERNDIR/buffer 444 root root
f none $KERNDIR/buffer.conf 444 root root
```

При рассмотрении файла `prototype` в этом примере обратите внимание на следующее:

- Поскольку особой обработки объектов пакета не требуется, их можно поместить в стандартный класс `none`. Параметру `CLASSES` присваивается значение `none` в файле `pkginfo`.
- Имена пути файлов `buffer` и `buffer.conf` начинаются с переменной `$KERNDIR`. Эта переменная устанавливается в сценарии `request` и позволяет администратору решить, куда устанавливать файлы драйвера. Каталог по умолчанию - `/kernel/drv`.
- Существует запись для сценария `postinstall` (сценария, который будет производить установку драйвера).

Сценарий `request`

```
trap 'exit 3' 15
# determine where driver object should be placed; location
# must be an absolute path name that is an existing directory
KERNDIR=`ckpath -aoy -d /kernel/drv -p \
"Where do you want the driver object installed"` || exit $?

# make parameters available to installation service, and
# so to any other packaging scripts
cat >$1 <<!

CLASSES='$CLASSES'
KERNDIR='$KERNDIR'
!
exit 0
```

Сценарий `postinstall`

```
# KERNDIR parameter provided by 'request' script
err_code=1                    # an error is considered fatal
# Load the module into the system
cd $KERNDIR
add_drv -m '* 0666 root sys' buffer || exit $err_code
# Create a /dev entry for the character node
installf $PKGINST /dev/buffer0=/devices/eisa/buffer*:0 s
installf -f $PKGINST
```

Сценарий `preremove`

```
err_code=1                    # an error is considered fatal
# Unload the driver
rem_drv buffer || exit $err_code
# remove /dev file
removef $PKGINST /dev/buffer0 ; rm /dev/buffer0
removef -f $PKGINST
```

Установка драйвера с помощью класса sed и процедурных сценариев

В данном практическом примере описывается, как установить драйвер с помощью класса sed и процедурных сценариев. Он отличается от предыдущего практического примера (см. раздел «Установка и удаление драйвера с помощью процедурных сценариев» на стр. 127), поскольку данный пакет состоит как из абсолютных, так и перемещаемых объектов.

Методы

В практическом примере применяются следующие методы:

- Создание файла prototype с абсолютными и перемещаемыми объектами.

  Для получения дополнительной информации о создании файла prototype см. раздел «Создание файла prototype» на стр. 34.

- Использование сценария postinstall

  Для получения дополнительной информации об этом сценарии см. раздел «Создание процедурных сценариев » на стр. 73.

- Использование сценария preremove

  Для получения дополнительной информации об этом сценарии см. раздел «Создание процедурных сценариев » на стр. 73.

- Использование файла copyright

  Для получения информации об этом файле см. раздел «Создание сообщения об авторских правах» на стр. 60.

Подход

- Создайте файл prototype, содержащий абсолютные и перемещаемые объекты пакета.

  Подробности создания такого файла приведены в разделе «Файл prototype» на стр. 131.

- Добавьте сценарий класса sed в файл prototype.

  Название сценария должно совпадать с именем файла, который предстоит редактировать. В данном случае изменяемый файл называется /etc/devlink.tab, поэтому сценарий sed также будет называться /etc/devlink.tab. В сценарии sed отсутствуют требования по режиму, владельцу и группе (они представлены в образце файла prototype вопросительными знаками). Файл сценария sed должен относиться к типу e (редактируемый).

- Включите в параметр CLASSES класс sed.

- Создайте сценарий действия над классом sed (/etc/devlink.tab).
- Создайте сценарий postinstall.

  Для добавления драйвера устройства в систему, сценарию postinstall необходимо выполнить команду add_drv.
- Создайте сценарий preremove.

  Для удаления драйвера устройства из системы, сценарию preremove необходимо выполнить команду rem_drv до удаления пакета.
- Создайте файл copyright.

  Файл copyrightсодержит сообщение об авторских правах в текстовом формате с кодировкой ASCII. Сообщение, приведенное в файле примера, отображается на дисплее компьютера в ходе установки пакета.

Файлы практических примеров

Файл pkginfo

```
PKG=SUNWsst
NAME=Simple SCSI Target Driver
VERSION=1
CATEGORY=system
ARCH=sparc
VENDOR=Sun Microsystems
BASEDIR=/opt
CLASSES=sed
```

Файл prototype

В нашем практическом примере используется иерархическое распределение объектов пакета, как показано на рисунке ниже.

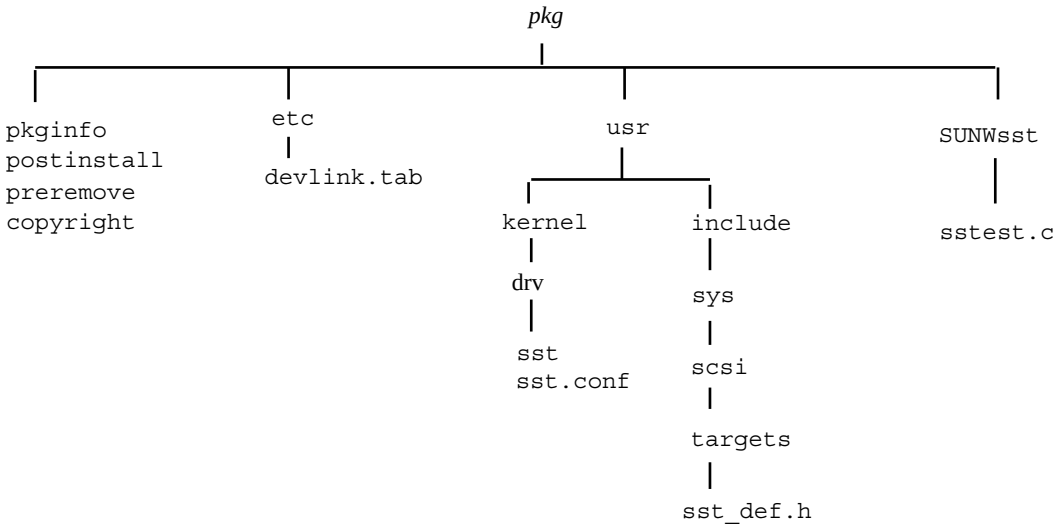


РИСУНОК 5–1 Структура каталогов иерархического пакета

Объекты пакета устанавливаются в то же место, в котором они расположены в каталоге pkgвыше. Модули драйвера (`sst` и `sst.conf`) устанавливаются в каталог `/usr/kernel/drv`, а вложенный файл в каталог `/usr/include/sys/scsi/targets`. Файлы `sst`, `sst.conf` и `sst_def.h` являются абсолютными объектами. Тестовая программа `sstest.c` и ее каталог `SUNWsst` являются перемещаемыми; место их установки определяется параметром `BASEDIR`.

Оставшиеся компоненты пакета (все контрольные файлы) помещаются в каталог верхнего уровня пакета на компьютере разработчика, за исключением сценария класса sed. Сценарий называется `devlink.tab` в соответствии с изменяемым им файлом и помещается в каталог `etc`, содержащий действительный файл `devlink.tab`.

Из каталога pkg выполните команду `pkgproto` следующим образом:

```
find usr SUNWsst -print | pkgproto > prototype
```

Результат выполнения этой команды выглядит так:

```
d none usr 0775 pms mts
d none usr/include 0775 pms mts
d none usr/include/sys 0775 pms mts
d none usr/include/sys/scsi 0775 pms mts
d none usr/include/sys/scsi/targets 0775 pms mts
f none usr/include/sys/scsi/targets/sst_def.h 0444 pms mts
d none usr/kernel 0775 pms mts
d none usr/kernel/drv 0775 pms mts
f none usr/kernel/drv/sst 0664 pms mts
```

```
f none usr/kernel/drv/sst.conf 0444 pms mts
d none SUNWsst 0775 pms mts
f none SUNWsst/sstest.c 0664 pms mts
```

Создание данного файла `prototype` еще не завершено. Чтобы закончить создание этого файла, произведите следующие изменения:

- Вставьте записи для контрольных файлов (тип файла `i`), поскольку в них содержится информация, отличная от других объектов пакета.
- Удалите записи для каталогов, уже существующих в целевой системе.
- Измените права доступа и принадлежность каждой записи.
- Добавьте косую черту к началу абсолютных объектов пакета.

Окончательный файл `prototype` выглядит так:

```
i pkginfo
i postinstall
i preremove
i copyright
e sed /etc/devlink.tab ? ? ?
f none /usr/include/sys/scsi/targets/sst_def.h 0644 bin bin
f none /usr/kernel/drv/sst 0755 root sys
f none /usr/kernel/drv/sst.conf 0644 root sys
d none SUNWsst 0775 root sys
f none SUNWsst/sstest.c 0664 root sys
```

Знаки вопроса в строке для сценария `sed` указывают, что права доступа и принадлежность существующего файла на целевом компьютере не должны изменяться.

Сценарий действия над классом `sed(/etc/devlink.tab)`

В примере с драйвером сценарий класса `sed` используется для добавления записи для драйвера в файл `/etc/devlink.tab`. Этот файл используется командой `devlinks` для создания символической ссылки из `/dev` в `/devices`. Сценарий `sed` выглядит так:

```
# sed class script to modify /etc/devlink.tab
!install
/name=sst;/d
$i\
type=ddi_pseudo;name=sst;minor=character    rsst\\A1

!remove
/name=sst;/d
```

Команда `pkgrm` не выполняет часть сценария, ответственную за удаление. Может понадобится добавление строки в сценарий `preremove` для того, чтобы класс `sed` удалил записи из файла `/etc/devlink.tab` напрямую.

Установочный сценарий postinstall

В данном практическом примере сценарию необходимо лишь выполнить команду add_drv.

```
# Postinstallation script for SUNWsst
# This does not apply to a client.
if [$PKG_INSTALL_ROOT = "/" -o -z $PKG_INSTALL_ROOT]; then
   SAVEBASE=$BASEDIR
   BASEDIR=””; export BASEDIR
   /usr/sbin/add_drv sst
   STATUS=$?
   BASEDIR=$SAVEBASE; export BASEDIR
   if [ $STATUS -eq 0 ]
   then
         exit 20
   else
         exit 2
   fi
else
   echo "This cannot be installed onto a client."
   exit 2
fi
```

Команда add_drv использует параметр BASEDIR, поэтому сценарию необходимо отменить установку параметра BASEDIR перед выполнением команды и восстановить его после выполнения.

Одним из действий, выполняемых командой add_drv, является выполнение команды devlinks, которая использует запись, размещенную в каталоге /etc/devlink.tab сценарием класса sed для создания записей /dev для драйвера.

Очень важен код выхода из сценария postinstall. Код выхода 20 указывает команде pkgadd известить пользователя о необходимости перезагрузки системы после установки драйвера, а код выхода 2 указывает команде pkgadd известить пользователя о том, что установка частично не удалась.

Сценарий удаления preremove

В практическом примере с данным драйвером этот сценарий удаляет ссылки в каталоге /dev и выполняет команду rem_drv на драйвере.

```
# Pre removal script for the sst driver
echo “Removing /dev entries”
/usr/bin/rm -f /dev/rsst*

echo “Deinstalling driver from the kernel”
SAVEBASE=$BASEDIR
```

```
BASEDIR=””; export BASEDIR
/usr/sbin/rem_drv sst
BASEDIR=$SAVEBASE; export BASEDIR

exit
```

Сценарий самостоятельно удаляет записи /dev; записи /devices удаляются командой rem_drv.

Файл copyright

Это простой файл ASCII, содержащий текст уведомления об авторских правах. Уведомление отображается в начале установки пакета точно в таком же виде, как оно представлено в файле.

```
    Copyright (c) 1999 Drivers-R-Us, Inc.
    10 Device Drive, Thebus, IO 80586

All rights reserved. This product and related documentation is
protected by copyright and distributed under licenses
restricting its use, copying, distribution and decompilation.
No part of this product or related documentation may be
reproduced in any form by any means without prior written
authorization of Drivers-R-Us and its licensors, if any.
```

◆ ◆ ◆ ГЛАВА 6

Дополнительные методы создания пакетов

В ОС Solaris в полном объеме реализованы все возможности по созданию пакетов System V, которые представляют собой мощный инстумент для установки программных продуктов. Эти возможности доступны для использования разработчику пакетов. Для настройки установок типа клиент-сервер в пакетах, не являющихся частью ОС Solaris (несобранные пакеты), может использоваться механизм классов. Имеется возможность разработки перемещаемых пакетов для удобства работы администратора. Сложный продукт может поставляться в виде набора составных пакетов, которые автоматически вычисляют зависимости пакета. Разработчик пакета может настроить процесс его обновления и исправления. Исправленные пакеты могут поставляться так же, как и неисправленные пакеты. В продукт могут включаться архивы файлов отката.

Ниже следует общий перечень вопросов, рассмотренных в данной главе.

Определение базового каталога

Существуют несколько методов, позволяющих указать место, куда будет установлен пакет. Важно иметь возможность изменять базовый каталог динамически во время установки. Если эта операция выполнена верно, администратор сможет без труда устанавливать разные версии и различные архитектуры компьютеров.

В начале этого раздела рассматриваются наиболее общие методы, а затем подходы, улучшающие процесс установки в неоднородных операционных средах.

Файл административных значений по умолчанию

Администраторы, ответственные за установку пакетов, могут использовать файлы администрирования для контроля над процессом установки пакета. Разработчику пакета следует знать о файлах администрирования и каким образом администратор может изменить предполагаемую установку пакета.

Файл администрирования сообщает команде `pkgadd`, следует ли осуществлять все проверки или выводить запросы, которые она обычно выводит. Поэтому перед использованием файлов администрирования администраторам необходимо полностью понять процесс установки пакета и работы сценариев.

Базовый файл административных значений по умолчанию поставляется вместе с операционной системой SunOS и находится по следующему пути: `/var/sadm/install/admin/default`. Этот файл устанавливает самый простой уровень административной политики в отношении установки программных продуктов. Поставляемый файл выглядит следующим образом:

```
#ident "@(#)default
1.4 92/12/23 SMI"    /* SVr4.0 1.5.2.1    */
mail=
instance=unique
partial=ask
runlevel=ask
idepend=ask
rdepend=ask
space=ask
setuid=ask
conflict=ask
action=ask
basedir=default
```

Администратор может редактировать этот файл и устанавливать новые значения по умолчанию. Он может также создать новый файл администрирования и указать на его наличие с помощью параметра `-a` команды `pkgadd`.

В файле администрирования можно при необходимости определить одиннадцать параметров. Для получения дополнительной информации см. страницу `admin(4)`.

Параметр `basedir` указывает, каким образом будет извлечен базовый каталог при установке пакета. Большинство администраторов оставляют этот параметр со значением по умолчанию (`default`), однако для параметра `basedir` можно установить одно из следующих значений:

- `ask` - всегда спрашивать администратора о расположении базового каталога
- Абсолютное имя пути

- Абсолютное имя пути, содержащее конструкцию `$PKGINST`, означающую, что следует всегда выполнять установку в базовый каталог, извлеченный из экземпляра пакета

Примечание – При вызове команды `pkgadd` с аргументом `-a none` команда всегда запрашивает администратора о размещении базового каталога. К сожалению, при этом *все* параметры в файле устанавливаются на значение по умолчанию `quit`, что может создать дополнительные сложности.

Устранение неопределенности

Обычно администратор контролирует все пакеты, устанавливаемые в системе, с помощью файла администрирования. К сожалению, *разработчики пакетов* часто предоставляют альтернативные файлы административных значений по умолчанию, обходя тем самым пожелания администратора.

Разработчики пакетов иногда включают в поставку свой файл администрирования с тем, чтобы они, а не администраторы, контролировали установку пакета. Поскольку запись `basedir` в файле административных значений по умолчанию отменяет все другие базовые каталоги, это предоставляет простой способ выбора подходящего базового каталога во время установки. Во всех версиях ОС Solaris до выпуска Solaris 2.5 подобный способ контроля над базовым каталогом считался наиболее простым.

Однако следует принимать во внимание желания администраторов, которые хотят контролировать процесс установки продукта. Предоставление временного файла с административными значениями по умолчанию для контроля над установкой приводит к недоверию со стороны администраторов. Следует использовать сценарии `request` и `checkinstall` с тем, чтобы установка происходила под контролем администратора. Если сценарий `request` вовлекает в процесс установки администратора, организация пакетов System V облегчит работу как разработчиков, так и администраторов.

Использование параметра `BASEDIR`

Файл `pkginfo` для любого перемещаемого пакета должен включать базовый каталог по умолчанию в форме подобной записи:

`BASEDIR=`*`absolute_path`*

Это лишь базовый каталог по умолчанию; он может быть изменен администратором в ходе установки.

Хотя для некоторых пакетов требуется несколько базовых каталогов, преимущество использования этого параметра для размещения пакета состоит в том, что к моменту начала установки базовый каталог гарантированно будет на месте и доступным для записи как допустимый каталог. Правильный путь к базовому каталогу сервера и

клиента доступен всем процедурным сценариям в форме зарезервированных переменных среды. Текущий базовый каталог установки отображается с помощью команды `pkginfo -r SUNWstuf`.

В сценарии `checkinstall` параметр `BASEDIR` существует точно в том виде, как он был определен в файле `pkginfo` (условия еще не были определены). Для проверки целевого базового каталога требуется конструкция `${PKG_INSTALL_ROOT}$BASEDIR`. Это означает, что сценарии `request` и `checkinstall` могут изменять значение параметра `BASEDIR` в среде установки с предсказуемыми результатами. Ко времени вызова сценария `preinstall` параметр `BASEDIR` уже является полностью обусловленным указателем на действительный базовый каталог целевой системы, даже если эта система - клиент.

Примечание – Сценарий `request` использует параметр `BASEDIR` по-разному в разных выпусках операционной системы SunOS. Чтобы осуществить проверку параметра `BASEDIR` в сценарии `request`, следует использовать следующий программный код, позволяющий определить фактический используемый базовый каталог.

```
# request script
constructs base directory
if [ ${CLIENT_BASEDIR} ]; then
      LOCAL_BASE=$BASEDIR
else
      LOCAL_BASE=${PKG_INSTALL_ROOT}$BASEDIR
fi
```

Использование параметрических базовых каталогов

Если для пакета требуется несколько базовых каталогов, их можно создать с помощью параметрических имен путей. Этот способ стал достаточно популярным, несмотря на то, что он имеет следующие ниже недостатки.

- Пакет с параметрическими именами пути обычно ведет себя как абсолютный пакет, однако команда `pkgadd` расценивает его как перемещаемый пакет. Обязательно должен быть определен параметр `BASEDIR`, даже если он и не используется.
- Администратор не может выяснить расположение базового каталога установки пакета с помощью служебных программ System V (команда `pkginfo -r` выполняться не будет).
- Администратор не может использовать стандартный метод перемещения пакета (он называется перемещаемым, но ведет себя как абсолютный).
- Установка на различные архитектуры или установка разных версий требует планирования вариантов действий для каждого целевого базового каталога, что часто означает написание большого количества сложных сценариев действий над классами.

После того, как параметры, устанавливающие базовые каталоги, определены в файле `pkginfo`, их можно изменять с помощью сценария `request`. Это одна из главных причин популярности данного подхода. Тем не менее, перечисленные выше недостатки являются неисправимыми, и данная структура должна рассматриваться лишь как последнее средство.

Примеры. Использование параметрических базовых каталогов

Файл `pkginfo`

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/
EZDIR=/usr/stuf/EZstuf
HRDDIR=/opt/SUNWstuf/HRDstuf
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none
PSTAMP=hubert980707141632
```

Файл `pkgmap`

```
: 1 1758
1 d none $EZDIR 0775 root bin
1 f none $EZDIR/dirdel 0555 bin bin 40 773 751310229
1 f none $EZDIR/usrdel 0555 bin bin 40 773 751310229
1 f none $EZDIR/filedel 0555 bin bin 40 773 751310229
1 d none $HRDDIR 0775 root bin
1 f none $HRDDIR/mksmart 0555 bin bin 40 773 751310229
1 f none $HRDDIR/mktall 0555 bin bin 40 773 751310229
1 f none $HRDDIR/mkcute 0555 bin bin 40 773 751310229
1 f none $HRDDIR/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
1 f none /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

Управление базовым каталогом

Любой пакет, существующий в нескольких версиях или для нескольких архитектур, должен, при необходимости, иметь возможность *увода* базового каталога. Увод базового каталога означает, что если в базовом каталоге уже существует предыдущая версия или другая архитектура устанавливаемого пакета, устанавливаемый пакет решает эту проблему, создавая, например, новый базовый каталог со слегка отличающимся именем. Сценарии `request` и `checkinstall` в Solaris 2.5 и совместимых выпусках имеют возможность изменять переменную среды `BASEDIR`. Однако такая возможность отсутствует в более ранних версиях ОС Solaris.

Даже в более старых версиях операционного окружения Solaris сценарию `request` были доступны полномочия на переопределение каталогов в пределах базового каталога установки. Сценарий `request` может делать это и сейчас, поддерживая большинство административных предпочтений.

Настройка перемещения каталогов

Несмотря на то, что можно выбрать базовые каталоги для различных пакетов таким образом, что они гарантированно будут уникальными для данной архитектуры или версии, это приводит к образованию ненужных уровней иерархии каталога. Например, для продукта, созданного для процессоров архитектуры SPARC и x86, можно было бы упорядочить базовые каталоги по процессору и по версии, как показано ниже.

Базовый каталог	Версия и процессор
`/opt/SUNWstuf/sparc/1.0`	Версия 1.0, SPARC
`/opt/SUNWstuf/sparc/1.2`	Версия 1.2, SPARC
`/opt/SUNWstuf/x86/1.0`	Версия 1.0, x86

Хотя такой подход состоятелен и работает, однако разработчик в данном случае обращается с именами и числами так, как если бы они что-то значили для администратора. Лучшим вариантом было бы делать это автоматически *после* выдачи соответствующих разъяснений администратору и получения разрешения.

Это означает, что можно сделать всю работу в пакете, не вынуждая администратора выполнять ее вручную. Можно произвольно назначить базовый каталог, а затем установить соответствующие клиентские ссылки в сценарии `postinstall` . Для установки части или всего пакета на клиентские машины можно также использовать команду `pkgadd` в сценарии `postinstall`. Можно даже выдать запрос администратору, какие пользователи или клиенты должны знать об этом пакете и автоматически обновить переменные среды `PATH` и файлы `/etc`. Это вполне приемлемо при условии, что какие бы действия пакет ни выполнял при установке, он отменит их при удалении.

Увод базовых каталогов

Можно воспользоваться преимуществами двух методов контроля базового каталога в период установки. Первый метод лучше всего подходит для новых пакетов, которые устанавливаются только на Solaris 2.5 и совместимые выпуски. Этот метод предоставляет администратору очень полезные данные и поддерживает большое количество установленных версий и архитектур при минимальных усилиях. Второй метод может применяться в любом пакете. Он использует внутренний контроль сценария `request` над параметрами сборки пакета для обеспечения успешной установки.

Использование параметра `BASEDIR`

Сценарий `checkinstall` может выбрать соответствующий базовый каталог во время установки, что означает, что базовый каталог может быть размещен на очень низком уровне в дереве каталогов. В данном примере происходит последовательное увеличение номера базового каталога, что приводит к формированию каталогов вида `/opt/SUNWstuf`, `/opt/SUNWstuf.1` и `/opt/SUNWstuf.2`. Администратор может воспользоваться командой `pkginfo` для определения установленной в каждом базовом каталоге архитектуры и версии.

Если в пакете `SUNWstuf` (содержащем набор служебных программ) используется данный метод, то его файлы `pkginfo` и `pkgmap` будут выглядеть следующим образом:

Файл `pkginfo`

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt/SUNWstuf
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none daemon
PSTAMP=hubert990707141632
```

Файл `pkgmap`

```
: 1 1758
1 d none EZstuf 0775 root bin
1 f none EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none EZstuf/usrdel 0555 bin bin 40 773 751310229
```

```
1 f none EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none HRDstuf 0775 root bin
1 f none HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
1 f daemon /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
1 i i.daemon 509 39560 752978103
1 i r.daemon 320 24573 742152591
```

Пример. Аналитические сценарии, осуществляющие увод BASEDIR

Предположим, что SUNWstuf для процессора архитектуры x86 уже установлен на сервере в каталоге /opt/SUNWstuf. Когда администратор выполняет команду pkgadd для установки SPARC, сценарий request должен обнаружить существование версии x86 и начать диалог с администратором касательно данной установки.

Примечание – В сценарии checkinstall увод базового каталога может осуществляться и без взаимодействия с администратором, однако если произвольные операции подобно этой случаются слишком часто, то администраторы начинают испытывать недоверие к происходящему процессу.

Сценарии request и checkinstall для пакета, который регулирует эту ситуацию, может выглядеть следующим образом:

Сценарий request

```
# request script
for SUNWstuf to walk the BASEDIR parameter.

PATH=/usr/sadm/bin:${PATH}    # use admin utilities

GENMSG="The base directory $LOCAL_BASE already contains a \
different architecture or version of $PKG."

OLDMSG="If the option \"-a none\" was used, press the  \
key and enter an unused base directory when it is requested."
```

```
OLDPROMPT="Do you want to overwrite this version? "

OLDHELP="\"y\" will replace the installed package, \"n\" will \
stop the installation."

SUSPEND="Suspending installation at user request using error \
code 1."

MSG="This package could be installed at the unused base directory $WRKNG_BASE."

PROMPT="Do you want to use to the proposed base directory? "

HELP="A response of \"y\" will install to the proposed directory and continue,
\"n\" will request a different directory. If the option \"-a none\" was used,
press the  key and enter an unused base directory when it is requested."

DIRPROMPT="Select a preferred base directory ($WRKNG_BASE) "

DIRHELP="The package $PKG will be installed at the location entered."

NUBD_MSG="The base directory has changed. Be sure to update \
any applicable search paths with the actual location of the \
binaries which are at $WRKNG_BASE/EZstuf and $WRKNG_BASE/HRDstuf."

OldSolaris=""
Changed=""
Suffix="0"

#
# Determine if this product is actually installed in the working
# base directory.
#
Product_is_present () {
      if [ -d $WRKNG_BASE/EZstuf -o -d $WRKNG_BASE/HRDstuf ]; then
            return 1
      else
            return 0
      fi
}

if [ ${BASEDIR} ]; then
      # This may be an old version of Solaris. In the latest Solaris
      # CLIENT_BASEDIR won't be defined yet. In older version it is.
      if [ ${CLIENT_BASEDIR} ]; then
            LOCAL_BASE=$BASEDIR
            OldSolaris="true"
      else    # The base directory hasn't been processed yet
            LOCAL_BASE=${PKG_INSTALL_ROOT}$BASEDIR
```

```
fi

WRKNG_BASE=$LOCAL_BASE

    # See if the base directory is already in place and walk it if
    # possible
while [ -d ${WRKNG_BASE} -a Product_is_present ]; do
         # There is a conflict
         # Is this an update of the same arch & version?
         if [ ${UPDATE} ]; then
               exit 0    # It's out of our hands.
         else
               # So this is a different architecture or
               # version than what is already there.
               # Walk the base directory
               Suffix=`expr $Suffix + 1`
               WRKNG_BASE=$LOCAL_BASE.$Suffix
               Changed="true"
         fi
done

    # So now we can propose a base directory that isn't claimed by
    # any of our other versions.
if [ $Changed ]; then
         puttext "$GENMSG"
         if [ $OldSolaris ]; then
               puttext "$OLDMSG"
               result=`ckyorn -Q -d "a" -h "$OLDHELP" -p "$OLDPROMPT"`
               if [ $result="n" ]; then
                     puttext "$SUSPEND"
                     exit 1    # suspend installation
               else
                     exit 0
               fi
         else    # The latest functionality is available
               puttext "$MSG"
               result=`ckyorn -Q -d "a" -h "$HELP" -p "$PROMPT"`
               if [ $? -eq 3]; then
                     echo quitinstall >> $1
                     exit 0
               fi

               if [ $result="n" ]; then
                     WRKNG_BASE=`ckpath -ayw -d "$WRKNG_BASE" \
                     -h "$DIRHELP" -p "$DIRPROMPT"`
               else if [ $result="a" ]
                     exit 0
               fi
```

```
            fi
            echo "BASEDIR=$WRKNG_BASE" >> $1
            puttext "$NUBD_MSG"
      fi
fi
exit 0
```

Сценарий `checkinstall`

```
# checkinstall
script for SUNWstuf to politely suspend

grep quitinstall $1
if [ $? -eq 0 ]; then
    exit 3        # politely suspend installation
fi

exit 0
```

Данный подход работал бы не очень хорошо, если бы базовым каталогом был просто `/opt`. Данный пакет должен вызывать `BASEDIR` более конкретно, поскольку осуществить увод в каталоге `/opt` достаточно сложно. На самом деле, в зависимости от схемы монтирования, это может оказаться невозможным. В приведенном примере увод базового каталога осуществляется путем создания нового каталога, который не представляет никаких проблем, под каталогом `/opt`.

В примере используются сценарии `request` и `checkinstall` несмотря на то, что версии Solaris до выпуска 2.5 не допускали выполнения сценария `checkinstall`. Сценарий `checkinstall` в данном примере используется для приостановки процесса установки в ответ на частное сообщение в форме строки `quitinstall`. Если этот сценарий выполняется в Solaris 2.3, сценарий `checkinstall` игнорируется, и сценарий `request` приостанавливает установку и выдает сообщение об ошибке.

Помните, что в выпусках до Solaris 2.5 и совместимых выпусках параметр `BASEDIR` был параметром только для чтения и не мог быть изменен сценарием `request`. По этой причине при обнаружении (путем проверки обусловленной переменной среды `CLIENT_BASEDIR`) старой версии операционной системы SunOS сценарий `request` имеет только две возможности: продолжить работу и завершить ее.

Использование относительных параметрических путей

Если программный продукт необходимо установить на старые версии операционной системы SunOS, вся необходимая работа должна производиться в сценарии `request`. Этот подход можно также использовать для управления большим количеством каталогов. Если потребуются дополнительные каталоги, их необходимо включить под одним базовым каталогом для облегчить администрирование продукта. Несмотря на то, что параметр `BASEDIR` не предоставляет ту степень детализации, которая доступна в

последних выпусках Solaris, пакет все равно может осуществлять увод базового каталога с помощью сценария `request`, который будет осуществлять управление параметрическими путями. Файлы `pkginfo` и `pkgmap` могут выглядеть следующим образом:

Файл `pkginfo`

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt
SUBBASE=SUNWstuf
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none daemon
PSTAMP=hubert990707141632
```

Файл `pkgmap`

```
: 1 1758
1 d none $SUBBASE/EZstuf 0775 root bin
1 f none $SUBBASE/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none $SUBBASE/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none $SUBBASE/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none $SUBBASE/HRDstuf 0775 root bin
1 f none $SUBBASE/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none $SUBBASE/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none $SUBBASE/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none $SUBBASE/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
1 f daemon /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
1 i i.daemon 509 39560 752978103
1 i r.daemon 320 24573 742152591
```

Данный пример не совершенен. Команда `pkginfo -r` возвращает в качестве базы установки каталог `/opt`, что очень расплывчато. Каталог `/opt` содержит много пакетов,

но по крайней мере это значащий каталог. Как и в примере выше, данный пример полностью поддерживает большое количество архитектур и версий. Сценарий `request` можно подогнать под потребности определенного пакета и проанализировать применимые зависимости.

Пример. Сценарий `request`, осуществляющий увод по относительному параметрическому пути

```
# request script
for SUNWstuf to walk a parametric path

PATH=/usr/sadm/bin:${PATH}    # use admin utilities

MSG="The target directory $LOCAL_BASE already contains \
different architecture or version of $PKG. This package \
could be installed at the unused target directory $WRKNG_BASE."

PROMPT="Do you want to use to the proposed directory? "

HELP="A response of \"y\" will install to the proposed directory \
and continue, \"n\" will request a different directory. If \
the option \"-a none\" was used, press the <RETURN> key and \
enter an unused base directory when it is requested."

DIRPROMPT="Select a relative target directory under $BASEDIR/"

DIRHELP="The package $PKG will be installed at the location entered."

SUSPEND="Suspending installation at user request using error \
code 1."

NUBD_MSG="The location of this package is not the default. Be \
sure to update any applicable search paths with the actual \
location of the binaries which are at $WRKNG_BASE/EZstuf \
and $WRKNG_BASE/HRDstuf."

Changed=""
Suffix="0"

#
# Determine if this product is actually installed in the working
# base directory.
#
Product_is_present () {
      if [ -d $WRKNG_BASE/EZstuf -o -d $WRKNG_BASE/HRDstuf ]; then
            return 1
      else
            return 0
```

```
      fi
}

if [ ${BASEDIR} ]; then
      # This may be an old version of Solaris. In the latest Solaris
      # CLIENT_BASEDIR won't be defined yet. In older versions it is.
      if [ ${CLIENT_BASEDIR} ]; then
            LOCAL_BASE=$BASEDIR/$SUBBASE
      else    # The base directory hasn't been processed yet
            LOCAL_BASE=${PKG_INSTALL_ROOT}$BASEDIR/$SUBBASE
      fi

WRKNG_BASE=$LOCAL_BASE

# See if the base directory is already in place and walk it if
# possible
while [ -d ${WRKNG_BASE} -a Product_is_present ]; do
         # There is a conflict
         # Is this an update of the same arch & version?
         if [ ${UPDATE} ]; then
               exit 0    # It's out of our hands.
         else
               # So this is a different architecture or
               # version than what is already there.
               # Walk the base directory
               Suffix=`expr $Suffix + 1`
               WRKNG_BASE=$LOCAL_BASE.$Suffix
               Changed="true"
         fi
done

# So now we can propose a base directory that isn't claimed by
# any of our other versions.
if [ $Changed ]; then
         puttext "$MSG"
         result=`ckyorn -Q -d "a" -h "$HELP" -p "$PROMPT"`
         if [ $? -eq 3 ]; then
               puttext "$SUSPEND"
               exit 1
         fi

         if [ $result="n" ]; then
               WRKNG_BASE=`ckpath -lyw -d "$WRKNG_BASE" -h "$DIRHELP" \
               -p "$DIRPROMPT"`

            elif [ $result="a" ]; then
                  exit 0
```

```
            else
                  exit 1
            fi
            echo SUBBASE=$SUBBASE.$Suffix >> $1
            puttext "$NUBD_MSG"
      fi
fi
exit 0
```

Поддержка перемещения в неоднородной вычислительной среде

Исходная концепция пакетирования в System V предполагало наличие одной архитектуры на систему. В этом проектном решении концепция сервера не играла никакой роли. В настоящее время один сервер может осуществлять поддержку нескольких архитектур, то есть на сервере могут присутствовать несколько копий одного и того же программного обеспечения, разработанного для разных архитектур. Несмотря на то, что пакеты Solaris изолированы в рекомендованных рамках файловой системы (например, `/` и `/usr`) с размещением баз данных продукта на сервере и на каждом из клиентов, не все типы установок поддерживают это разделение. Некоторые разработки поддерживают совершенно другую структуру и предполагают наличие общей базы данных продуктов. Хотя направление клиента на другие версии представляется самым простым решением, в действительности установка пакетов System V в другие базовые каталоги может создать сложности для администратора.

При разработке пакета необходимо принять во внимание те методы, которые используют администраторы при внедрении новых версий ПО. Часто администраторы хотят устанавливать и тестировать новейшие версии наряду с уже установленной версией. Данная процедура включает в себя установку новой версии в базовый каталог, отличный от каталога, в котором расположена существующая версия, и направление нескольких некритических клиентов на новую версию в качестве теста. По мере нарастания уверенности в работе новой версии, администратор перенаправляет на нее все большее и большее число клиентов. В конечном итоге администратор оставляет старую версию только для чрезвычайных ситуаций, а позднее и полностью ее удаляет.

Все это подводит нас к тому, что пакеты, разработанные для современных неоднородных систем, должны поддерживать истинное перемещение в том смысле, что администратор может поместить их в любое допустимое место в файловой системе и при этом получить полную функциональность. Solaris 2.5 и совместимые выпуски предлагают ряд полезных средств, которые позволяют осуществлять чистую установку нескольких архитектур и версий на одну систему. Solaris 2.4 и совместимые версии также поддерживают истинное перемещение, однако решение этой задачи не совсем очевидно.

Традиционный подход

Перемещаемые пакеты

Двоичный интерфейс приложений System V предполагает, что первоначальным намерением при разработке перемещаемых пакетов было облегчить труд администратора по установке. В настоящее время потребности в перемещаемых пакетах идут гораздо дальше. Удобство уже не является единственной проблемой, поскольку вполне возможно, что в процессе установки активный программный продукт уже установлен в каталоге по умолчанию. Пакет, не спроектированный на решение данной ситуации, либо перезаписывает существующий продукт, либо установка не удается. С другой стороны пакет, разработанный для работы с многими архитектурами и версиями, может легко быть установлен и одновременно предлагает администратору широкий выбор возможностей, полностью совместимых с существующими традициями администрирования.

В какой-то степени проблема нескольких архитектур и проблема нескольких версий представляют собой одно и то же. Должна существовать возможность установки варианта существующего пакета наряду с другими вариантами и направления клиентов или автономных потребителей экспортированных файловых систем на любой из этих вариантов без потери функциональности. Несмотря на то, что компания Sun разработала способы обращения с множественными архитектурами на сервере, администратор может и не придерживаться этих рекомендаций. Все пакеты должны быть в состоянии соответствовать разумным пожеланиям администраторов касательно установки.

Пример. Традиционный перемещаемый пакет

В данном примере показано, как может выглядеть традиционный перемещаемый пакет. Пакет должен располагаться в каталоге `/opt/SUNWstuf`, а его файлы `pkginfo` и `pkgmap` могут выглядеть следующим образом:

Файл `pkginfo`

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
```

```
MAXINST=1000
CLASSES=none
PSTAMP=hubert990707141632
```

Файл pkgmap

```
: 1 1758
1 d none SUNWstuf 0775 root bin
1 d none SUNWstuf/EZstuf 0775 root bin
1 f none SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none SUNWstuf/HRDstuf 0775 root bin
1 f none SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

Этот метод называется традиционным потому что каждый объект пакета устанавливается в базовый каталог, определенный параметром `BASEDIR` в файле `pkginfo`. Например, первый объект в файле `pkgmap` устанавливается как каталог `/opt/SUNWstuf`.

Абсолютные пакеты

Абсолютным является пакет, устанавливаемый в конкретную корневую (`/`) файловую систему. С точки зрения множественных версий и архитектур эти пакеты являются сложными в обращении. Как правило все пакеты должны быть перемещаемыми. Существуют, однако, довольно веские причины включать абсолютные элементы в перемещаемые пакеты.

Пример. Традиционный абсолютный пакет

Если бы пакет `SUNWstuf` являлся абсолютным, то определения параметра `BASEDIR` в файле `pkginfo` не потребовалось, а файл `pkgmap` выглядел бы так:

Файл pkgmap

```
: 1 1758
1 d none /opt ? ? ?
1 d none /opt/SUNWstuf 0775 root bin
1 d none /opt/SUNWstuf/EZstuf 0775 root bin
1 f none /opt/SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
```

```
1 f none /opt/SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none /opt/SUNWstuf/HRDstuf 0775 root bin
1 f none /opt/SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

В данном примере если администратор в процессе установки указывает альтернативный базовый каталог, он будет проигнорирован командой `pkgadd`.. Данный пакет всегда будет устанавливаться в каталог `/opt/SUNWstuf` целевой системы.

Параметр `-R` команды `pkgadd` работает как и ожидалось. Например,

```
pkgadd -d . -R /export/opt/client3 SUNWstuf
```

устанавливает объекты в каталог `/export/opt/client3/opt/SUNWstuf`. Здесь пакет наиболее близко подходит к определению "перемещаемый".

Обратите внимание на использование знака вопроса (`?`) в каталоге `/opt` файла `pkgmap`. Он означает, что существующие атрибуты не должны изменяться. Он не означает "создать каталог с атрибутами по умолчанию", хотя при некоторых обстоятельствах это может случиться. Для любого каталога, относящегося к новому пакету, необходимо явно указывать все атрибуты.

Составные пакеты

Любой пакет, содержащий перемещаемые объекты, называется перемещаемым пакетом. Данное определение может ввести в заблуждение, поскольку в файле `pkgmap` перемещаемого пакета могут содержаться абсолютные пути. Использование корневой (`/`) записи в файле `pkgmap` может усилить перемещаемые аспекты пакета. Пакеты, содержащие одновременно перемещаемые и корневые записи, называются *составными* пакетами.

Пример. Традиционное решение

Предположим, что один объект в пакете `SUNWstuf` является сценарием запуска, выполняемым на уровне выполнения 2. Файл `/etc/rc2.d/S70dostuf` должен быть установлен как часть пакета, однако он не может быть помещен в базовый каталог. При условии, что единственным решением здесь будет перемещаемый пакет, файлы `pkginfo` и `pkgmap` могут иметь следующий вид:

Файл pkginfo

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none
PSTAMP=hubert990707141632
```

Файл pkgmap

```
: 1 1758
1 d none opt/SUNWstuf/EZstuf 0775 root bin
1 f none opt/SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none opt/SUNWstuf/HRDstuf 0775 root bin
1 f none opt/SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none etc    ? ? ?
1 d none etc/rc2.d ? ? ?
1 f none etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

Нет большой разницы между этим подходом и подходом, применяемым в абсолютном пакете. В действительности абсолютный пакет был бы здесь предпочтительнее - если администратор указал альтернативный базовый каталог для этого пакета, то он не будет работать!

На самом деле только один файл в этом пакете должен быть связан с корнем - остальные файлы можно было бы переместить в любое место. В оставшейся части раздела будет показано, как решить эту проблему с помощью составного пакета.

За пределами традиции

Подход, описанный в данном разделе, не применим ко всем пакетам, но он приводит к увеличению производительности при установке в неоднородную среду. Приведенная здесь информация практически не относится к пакетам, поставляемым как часть ОС Solaris (собранные пакеты); однако на несобранных пакетах можно попрактиковаться в нетрадиционных способах создания пакетов.

Причиной стимулирования использования перемещаемых пакетов является поддержка следующего требования:

При добавлении или удалении пакета существующее желательное поведение установленных программных продуктов не должно измениться.

Несвязанные пакеты должны размещаться в каталоге `/opt` с тем, чтобы гарантировать отсутствие влияния нового пакета на существующие программные продукты.

Другой взгляд на составные пакеты

Существуют два правила, которым необходимо следовать при создании полнофункционального составного пакета:

- Разместить базовый каталог там, где находится абсолютное большинство объектов пакета.
- Если объект пакета размещается в обычном каталоге, не являющимся базовым каталогом (например, `/etc`), укажите этот объект в виде абсолютного имени пути в файле `prototype`.

Иными словами, поскольку "перемещаемый" означает, что объект может быть установлен в любом месте и по-прежнему работать, ни один стартовый сценарий, запускаемый с помощью `init` во время загрузки, не может считаться перемещаемым! И, хотя в поставляемом пакете вполне можно указать `/etc/passwd` в качестве относительного пути, есть только одно место, где он может быть размещен.

Представление абсолютных имен путей в виде перемещаемых

При создании составного пакета абсолютные пути должны работать так, чтобы не влиять на уже установленное программное обеспечение. Пакет, полностью размещаемый в каталоге `/opt`, обходит эту проблему, поскольку никакие существующие файлы ему не мешают. При включении файла из каталога `/etc` в пакет необходимо гарантировать, что абсолютные имена путей будут вести семя так же, как это ожидается от относительных имен путей. Рассмотрите следующие примеры.

Пример. Изменение файла

Описание

В таблицу добавляется запись или объект представляет собой новую таблицу, которая скорее всего будет изменена другими программами или пакетами.

Реализация

Определите объект как файловый типа `e` и относящийся к классу `build`, `awk` или `sed`. Сценарий, выполняющий эту задачу, должен удалять себя так же эффективно, как добавляет.

Пример

Необходимо добавить запись в `/etc/vfstab` для обеспечения поддержки нового твердотельного жесткого диска.

Запись в файле `pkgmap` может быть такой:

```
1 e sed /etc/vfstab ? ? ?
```

Сценарий `request` спрашивает оператора, должен ли каталог `/etc/vfstab` быть изменен пакетом. Если оператор отвечает “no” (нет), то сценарий запроса выведет на печать инструкции о том, как выполнить это задание вручную, и выполнит

```
echo "CLASSES=none" >> $1
```

Если оператор отвечает “yes” (да), тогда сценарий выполнит

```
echo "CLASSES=none sed" >> $1
```

активируя сценарий действия над классом, который произведет необходимые изменения. Класс `sed` означает, что файл пакета `/etc/vfstab` представляет собой программу `sed`, содержащую операции по установке и удалению одноименного файла в целевой системе.

Пример. Создание нового файла

Описание

Объект представляет собой совершенно новый файл, который вряд ли будет изменяться в будущем, или он заменяет файл, принадлежащий другому пакету.

Реализация

Определите объект пакета как тип файла `f` и установите его с помощью сценария действия над классом, способного отменить изменения.

Пример

В каталоге `/etc` необходим совершенно новый файл для предоставления необходимой информации для поддержки твердотельного жесткого диска с именем `/etc/shdisk.conf`. Запись в файле `pkgmap` может выглядеть следующим образом:

```
.
.
.
1 f newetc /etc/shdisk.conf
.
.
.
```

Сценарий действия над классом `i.newetc` отвечает за установку этого и любых других файлов, которые должны быть размещены в каталоге `/etc`. Он проверяет наличие в этом каталоге другого файла. Если файла нет, сценарий просто скопирует новый файл на место. Если же там уже существует файл, сценарий произведет его резервное копирование перед установкой нового файла. Сценарий `r.newetc` удаляет эти файлы и, если требуется, восстанавливает оригиналы файлов. Ниже представлен ключевой фрагмент установочного сценария.

```
# i.newetc
while read src dst; do
      if [ -f $dst ]; then
        dstfile=‘basename $dst‘
        cp $dst $PKGSAV/$dstfile
      fi
      cp $src $dst
done

if [ "${1}" = "ENDOFCLASS" ]; then
      cd $PKGSAV
      tar cf SAVE.newetc .
      $INST_DATADIR/$PKG/install/squish SAVE.newetc
fi
```

Обратите внимание, что сценарий использует переменную среды `PKGSAV` для сохранения резервной копии заменяемого файла. Когда аргумент `ENDOFCLASS` передается в сценарий, команда `pkgadd` сообщает сценарию, что это последние записи в данном классе, после чего сценарий производит архивирование и сжатие файлов, сохраненных с помощью частной программы сжатия, хранящейся в установочном каталоге пакета.

Использование переменной среды PKGSAV во время обновления пакета представляется ненадежным; однако, если пакет не обновляется (посредством внесения изменений, например), то архивный файл остается в безопасности. Следующий сценарий удаления содержит программный код, который решает другую проблему - старые версии команды pkgrmне передают сценариям правильный путь к переменной среды PKGSAV.

Сценарий удаления может выглядеть так:

```
# r.newetc

# make sure we have the correct PKGSAV
if [ -d $PKG_INSTALL_ROOT$PKGSAV ]; then
      PKGSAV="$PKG_INSTALL_ROOT$PKGSAV"
fi

# find the unsquish program
UNSQUISH_CMD=‘dirname $0‘/unsquish

while read file; do
      rm $file
done

if [ "${1}" = ENDOFCLASS ]; then
      if [ -f $PKGSAV/SAVE.newetc.sq ]; then
         $UNSQUISH_CMD $PKGSAV/SAVE.newetc
      fi

      if [ -f $PKGSAV/SAVE.newetc ]; then
         targetdir=dirname $file    # get the right directory
         cd $targetdir
            tar xf $PKGSAV/SAVE.newetc
            rm $PKGSAV/SAVE.newetc
      fi
fi
```

Данный сценарий использует частный не установленный алгоритм (unsquish), который расположен в установочном каталоге базы данных пакета. Это происходит автоматически с помощью команды pkgadd во время установки. Все сценарии, не опознанные явно командой pkgadd как сценарии "только для установки", остаются в этом каталоге для использования командой pkgrm. Нельзя рассчитывать на местоположение каталога, но можно положиться на то, что каталог является неструктурированным и содержит все надлежащие информационные файлы и установочные сценарии, необходимые для пакета. Данный сценарий находит каталог благодаря тому, что сценарий действия над классом наверняка будет исполняться из каталога, содержащего программу unsquish.

Обратите также внимание на то, что в данном сценарии просто не предполагается, что целевым каталогом является каталог `/etc`. В действительности это может быть каталог `/export/root/client2/etc` . Правильный каталог может быть составлен одним из двух способов.

- Использовать конструкцию `${PKG_INSTALL_ROOT}/etc` или
- Путем принятия имени каталога файла, передаваемого командой `pkgadd` (что и делает этот сценарий).

Используя данный подход для каждого абсолютного объекта в пакете, можно быть уверенным, что существующее желаемое поведение не изменится или, по крайней мере, его можно восстановить.

Пример. Составной пакет

Ниже приведен пример файлов `pkginfo` и `pkgmap` для составного пакета.

Файл `pkginfo`

```
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none daemon
PSTAMP=hubert990707141632
```

Файл `pkgmap`

```
: 1 1758
1 d none SUNWstuf/EZstuf 0775 root bin
1 f none SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none SUNWstuf/HRDstuf 0775 root bin
1 f none SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
```

```
1 e daemon /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i i.daemon 509 39560 752978103
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
1 i r.daemon 320 24573 742152591
```

Хотя `S70dostuf`принадлежит к классу `daemon`, каталоги, которые ведут к нему (и которые уже находятся на месте во время установки), принадлежат к классу `none`. Даже если каталоги являются уникальными для этого пакета, все равно следует оставить их в классе `none`. Причина этого кроется в том, что каталоги необходимо создавать в первую, а удалять в последнюю очередь, что всегда истинно для класса `none`. Команда `pkgadd` создает каталоги; они не копируются из пакета и не передаются в сценарий действия над классом для создания. Вместо этого они создаются командой `pkgadd` перед тем, как она вызывает установочный сценарий действия над классом, а команда `pkgrm` удаляет каталоги после завершения удаления сценария действия над классом.

Это означает, что если каталог в особом классе содержит объекты в классе `none`, то при попытке команды `pkgrm` удалить этот каталог произойдет сбой, поскольку каталог не будет вовремя освобожден. Если объект класса `none` необходимо вставить в каталог какого-либо особого класса, этот каталог еще не будет существовать в это время для принятия такого объекта. Команда `pkgadd` создаст каталог динамически в ходе установки объекта и, возможно, не сможет синхронизировать атрибуты этого каталога на момент появления его определения в файле `pkgmap`.

Примечание – При назначении каталога классу всегда помните о порядке создания и удаления.

Создание пакетов с возможностью дистанционной установки

Все пакеты *должны* иметь возможность дистанционной установки. Дистанционная установка означает отсутствие предположений о том, что администратор, устанавливающий пакет, станет устанавливать его в корневую (`/`) файловую систему компьютера с помощью команды `pkgadd`. Если в одном из процедурных сценариев требуется добраться до файла `/etc/vfstab` на целевой системе, необходимо использовать переменную среды `PKG_INSTALL_ROOT`. Другими словами, имя пути `/etc/vfstab`приведет к файлу `/etc/vfstab` системы, в которой выполняется команда `pkgadd`, но администратор может осуществлять установку на клиентскую систему в `/export/root/client3`. Путь `${PKG_INSTALL_ROOT}/etc/vfstab` гарантированно приведет на целевую файловую систему.

Пример. Установка на клиентскую систему

В этом примере пакет `SUNWstuf` устанавливается на `client3`, который настроен на `/opt` в своей корневой (`/`) файловой системе. Другая версия этого пакета уже установлена на `client3`, а базовый каталог установлен как `basedir=/opt/$PKGINST` в файле администрирования `thisadmin`. (Для получения дополнительной информации о файлах администрирования см. раздел «Файл административных значений по умолчанию» на стр. 138.) Команда `pkgadd`, исполняемая на сервере:

```
# pkgadd -a thisadmin -R /export/root/client3 SUNWstuf
```

В таблице ниже перечислены переменные среды и их значения, которые передаются в процедурные сценарии.

ТАБЛИЦА 6–1 Значения, передаваемые в процедурные сценарии

Переменная среды	Значение
`PKGINST`	`SUNWstuf.2`
`PKG_INSTALL_ROOT`	`/export/root/client3`
`CLIENT_BASEDIR`	`/opt/SUNWstuf.2`
`BASEDIR`	`/export/root/client3/opt/SUNWstuf.2`

Пример. Установка на сервер или на автономную систему

Для установки на сервер или на автономную систему в тех же условиях, что и в предыдущем примере, команда будет выглядеть так:

```
# pkgadd -a thisadmin SUNWstuf
```

В таблице ниже перечислены переменные среды и их значения, которые передаются в процедурные сценарии.

ТАБЛИЦА 6–2 Значения, передаваемые в процедурные сценарии

Переменная среды	Значение
`PKGINST`	`SUNWstuf.2`
`PKG_INSTALL_ROOT`	Не определено.
`CLIENT_BASEDIR`	`/opt/SUNWstuf.2`

ТАБЛИЦА 6–2 Значения, передаваемые в процедурные сценарии *(Продолжение)*

Переменная среды	Значение
BASEDIR	/opt/SUNWstuf.2

Пример. Монтирование общих файловых систем

Предположим, что пакет SUNWstuf создает и совместно использует файловую систему на сервере в каталоге /export/SUNWstuf/share. Когда пакет устанавливается на клиентские системы, их файлы /etc/vfstab должны быть обновлены для монтирования этой общей файловой системы. В этой ситуации можно использовать переменную CLIENT_BASEDIR.

Запись в клиентской системе должна содержать точку монтирования со ссылкой на клиентскую файловую систему. Эта строка должна быть построена правильно независимо от того, производится ли установка с сервера или с клиента. Предположим, что системное имя сервера - $SERVER. Можно перейти в $PKG_INSTALL_ROOT/etc/vfstab и с помощью команд sed или awk создать следующую строку для файла /etc/vfstab клиентской системы:

```
$SERVER:/export/SUNWstuf/share - $CLIENT_BASEDIR/usr nfs - yes ro
```

Например, для сервера с именем universe и для клиентской системы client9 строка в файле клиентской системы /etc/vfstab будет выглядеть так:

```
universe:/export/SUNWstuf/share - /opt/SUNWstuf.2/usr nfs - yes ro
```

При правильном использовании этих параметров запись всегда монтирует клиентскую файловую систему независимо от того, была ли она создана локально или на сервере.

Внесение исправлений в пакеты

Исправление пакета - это просто разреженный пакет, созданный для того, чтобы заменить определенные файлы оригинала. Нет особых причин для поставки разреженного пакета, за исключением экономии места на распространяемом носителе. Можно также поставить весь исходный пакет с несколькими измененными файлами или предоставить доступ к измененному пакету через сеть. Поскольку различаются только новые файлы (другие файлы не были перекомпилированы), команда pkgadd устанавливает эти отличающиеся файлы. Рассмотрите следующие рекомендации по внесению исправлений в пакеты:

- Если система достаточно сложная, рекомендуется создать систему идентификации исправлений, которая будет следить за тем, чтобы два файла с исправлениями не заменили один и тот же файл в попытке скорректировать ошибочное поведение

системы, вызванное разными причинами. Например, базовые номера исправлений Sun назначаются взаимоисключающим наборам файлов, за которые они ответственны.

- Необходимо реализовать возможность отката исправления.

Очень важно, чтобы номер версии пакета исправлений совпадал с номером версии исходного пакета. Необходимо отслеживать состояние исправлений пакета с помощью отдельной записи в файле `pkginfo` в следующей форме:

`PATCH=`*patch_number*

Если версия пакета заменена на версию исправления, создается еще один экземпляр пакета, и управлять таким исправленным продуктом становится чрезвычайно сложно. Метод последовательного применения исправлений к экземплярам имел определенные преимущества в ранних версиях ОС Solaris, но при работе на более сложных системах он сильно затрудняет управление.

Все параметры зоны в исправлении должны соответствовать параметрам зоны в пакете.

Что касается пакетов, составляющих ОС Solaris, то в базе данных пакетов должна быть только одна копия пакета, хотя там может находиться большое количество исправленных экземпляров. Для удаления объекта из установленного пакета (с помощью команды `removef`) необходимо выяснить, к каким экземплярам принадлежит этот файл.

Однако если для пакета (который не является частью операционной среды Solaris) требуется определить уровень исправления какого-либо другого конкретного пакета, который *является* частью операционной среды Solaris, возникает проблема, требующая решения. Установочные сценарии могут быть достаточно большими, не нанося какого-либо значительного ущерба целевой файловой системе, поскольку они там не хранятся. С помощью сценариев действий над классами и различных процедурных сценариев можно сохранять измененные файлы в переменную среды `PKGSAV` (или какой-то другой, более постоянный каталог), чтобы иметь возможность отменить изменения, произведенные установленными исправлениями. Наблюдение за журналом установки исправлений можно также осуществить, установив соответствующие переменные среды посредством сценариев `request`. В сценариях, приведенных в следующих разделах, предполагается, что может существовать множество исправлений, схема нумерации которых имеет определенный смысл применительно к одному пакету. В этом случае отдельные номера исправлений представляют поднабор функционально связанных файлов в пакете. Два исправления с разными номерами не могут изменять один и тот же файл.

Чтобы превратить обычный разреженный пакет в пакет исправлений, сценарии, описанные в следующих разделах, можно просто вложить в пакет в сжатом виде. Все они распознаются как стандартные компоненты пакета, за исключением двух последних: `patch_checkinstall` и `patch_postinstall`. Эти два сценария могут быть внедрены в

пакет отмены исправлений, если есть необходимость предусмотреть такую возможность. Сценарии достаточно просты, а их различные задачи - прямолинейны.

Примечание – Этот метод внесения исправлений может использоваться на клиентских системах, однако корневые каталоги клиента на сервере должны иметь соответствующие права доступа, позволяющие чтение данных пользователем `install` или `nobody`.

Сценарий `checkinstall`

Сценарий `checkinstall` проверяет, подходит ли исправление к данному конкретному пакету. После подтверждения он создает *список исправлений* и *список информации об исправлении*, а затем вставляет их в файл ответа для внесения в базу данных пакета.

Список изменений - это список тех изменений, которые затронули текущий пакет. Список изменений записывается в файле `pkginfo` установленного пакета отдельной строкой, которая может выглядеть следующим образом:

`PATCHLIST=`*patch_id patch_id* ...

Список информации об исправлении - это список исправлений, от которых зависит текущее исправление. Этот список изменений также записывается в файле `pkginfo` отдельной строкой, которая может выглядеть следующим образом:

```
PATCH_INFO_103203-01=Installed... Obsoletes:103201-01 Requires: \ Incompatibles: 120134-01
```

Примечание – Эти строки (и их формат) объявляются общедоступным интерфейсом. Все компании, поставляющие исправления для пакетов Solaris, должны надлежащим образом обновлять этот список. При поставке исправления, каждый пакет в рамках исправлений содержит сценарий `checkinstall`, который выполняет эту задачу. Кроме того, сценарий `checkinstall` обновляет некоторые другие, относящиеся к исправлениям, параметры. Это новая архитектура исправлений, называемая прямым исправлением экземпляра.

В приводимом примере исходные пакеты и их исправления находятся в одном каталоге. Два исходных пакета называются `SUNWstuf.v1` и `SUNWstuf.v2`, а их исправления называются `SUNWstuf.p1` и `SUNWstuf.p2`. Это означает, что для процедурного сценария будет очень сложно обнаружить, из какого каталога пришли эти файлы, поскольку все символы в имени пакета, стоящие после точки (“.”), удаляются для параметра `PKG`, а переменная среды `PKGINST` ссылается на установленный экземпляр, а не на исходный экземпляр. Для того, чтобы процедурные сценарии смогли найти исходный каталог, сценарий `checkinstall` (который всегда выполняется из исходного каталога) делает запрос и передает расположение каталога дальше в виде переменной `SCRIPTS_DIR`. Если

бы в исходном каталоге существовал только один пакет с именем SUNWstuf, тогда процедурные сценарии могли бы его найти с помощью $INSTDIR/$PKG.

```
# checkinstall script to control a patch installation.
# directory format options.
#
#       @(#)checkinstall 1.6 96/09/27 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

PATH=/usr/sadm/bin:$PATH

INFO_DIR=‘dirname $0‘
INFO_DIR=‘dirname $INFO_DIR‘    # one level up

NOVERS_MSG="PaTcH_MsG 8 Version $VERSION of $PKG is not installed on this system."
ALRDY_MSG="PaTcH_MsG 2 Patch number $Patch_label is already applied."
TEMP_MSG="PaTcH_MsG 23 Patch number $Patch_label cannot be applied until all \
restricted patches are backed out."

# Read the provided environment from what may have been a request script
. $1

# Old systems can’t deal with checkinstall scripts anyway
if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        exit 0
fi

#
# Confirm that the intended version is installed on the system.
#
if [ "${UPDATE}" != "yes" ]; then
        echo "$NOVERS_MSG"
        exit 3
fi

#
# Confirm that this patch hasn’t already been applied and
# that no other mix-ups have occurred involving patch versions and
# the like.
#
Skip=0
active_base=‘echo $Patch_label | nawk ’
        { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
active_inst=‘echo $Patch_label | nawk ’
        { print substr($0, match($0, "Patchvers_pfx")+Patchvers_pfx_lnth) } ’‘
```

```
# Is this a restricted patch?
if echo $active_base | egrep -s "Patchstrict_str"; then
        is_restricted="true"
        # All restricted patches are backoutable
        echo "PATCH_NO_UNDO=" >> $1
else
        is_restricted="false"
fi

for patchappl in ${PATCHLIST}; do
        # Is this an ordinary patch applying over a restricted patch?
        if [ $is_restricted = "false" ]; then
                if echo $patchappl | egrep -s "Patchstrict_str"; then
                        echo "$TEMP_MSG"
                        exit 3;
                fi
        fi

        # Is there a newer version of this patch?
        appl_base=‘echo $patchappl | nawk ’
                { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
        if [ $appl_base = $active_base ]; then
                appl_inst=‘echo $patchappl | nawk ’
                        { print substr($0, match($0, "Patchvers_pfx")\
+Patchvers_pfx_lnth) } ’‘
                result=‘expr $appl_inst \> $active_inst‘
                if [ $result -eq 1 ]; then
                        echo "PaTcH_MsG 1 Patch number $Patch_label is \
superceded by the already applied $patchappl."
                        exit 3
                elif [ $appl_inst = $active_inst ]; then
                        # Not newer, it’s the same
                        if [ "$PATCH_UNCONDITIONAL" = "true" ]; then
                                if [ -d $PKGSAV/$Patch_label ]; then
                                        echo "PATCH_NO_UNDO=true" >> $1
                                fi
                        else
                                echo "$ALRDY_MSG"
                                exit 3;
                        fi
                fi
        fi
done

# Construct a list of applied patches in order
echo "PATCHLIST=${PATCHLIST} $Patch_label" >> $1
```

```
#
# Construct the complete list of patches this one obsoletes
#
ACTIVE_OBSOLETES=$Obsoletes_label

if [ -n "$Obsoletes_label" ]; then
        # Merge the two lists
        echo $Obsoletes_label | sed 'y/\ /\n/' | \
        nawk -v PatchObsList="$PATCH_OBSOLETES" '
        BEGIN {
                printf("PATCH_OBSOLETES=");
                PatchCount=split(PatchObsList, PatchObsComp, " ");

                for(PatchIndex in PatchObsComp) {
                        Atisat=match(PatchObsComp[PatchIndex], "@");
                        PatchObs[PatchIndex]=substr(PatchObsComp[PatchIndex], \
0, Atisat-1);
                        PatchObsCnt[PatchIndex]=substr(PatchObsComp\
[PatchIndex], Atisat+1);
                }
        }
        {
                Inserted=0;
                for(PatchIndex in PatchObs) {
                        if (PatchObs[PatchIndex] == $0) {
                                if (Inserted == 0) {
                                        PatchObsCnt[PatchIndex]=PatchObsCnt\
[PatchIndex]+1;
                                        Inserted=1;
                                } else {
                                        PatchObsCnt[PatchIndex]=0;
                                }
                        }
                }
                if (Inserted == 0) {
                        printf ("%s@1 ", $0);
                }
                next;
        }
        END {
                for(PatchIndex in PatchObs) {
                        if ( PatchObsCnt[PatchIndex] != 0) {
                                printf("%s@%d ", PatchObs[PatchIndex], \
PatchObsCnt[PatchIndex]);
                        }
                }
                printf("\n");
        } ' >> $1
```

```
        # Clear the parameter since it has already been used.
        echo "Obsoletes_label=" >> $1

        # Pass it's value on to the preinstall under another name
        echo "ACTIVE_OBSOLETES=$ACTIVE_OBSOLETES" >> $1
fi

#
# Construct PATCH_INFO line for this package.
#

tmpRequire=`nawk -F= ' $1 ~ /REQUIR/ { print $2 } ' $INFO_DIR/pkginfo `
tmpIncompat=`nawk -F= ' $1 ~ /INCOMPAT/ { print $2 } ' $INFO_DIR/pkginfo `

if [ -n "$tmpRequire" ] && [ -n "$tmpIncompat" ]
then
        echo "PATCH_INFO_$Patch_label=Installed: `date` From: `uname -n` \
          Obsoletes: $ACTIVE_OBSOLETES Requires: $tmpRequire \
          Incompatibles: $tmpIncompat" >> $1
elif [ -n "$tmpRequire" ]
then
        echo "PATCH_INFO_$Patch_label=Installed: `date` From: `uname -n` \
          Obsoletes: $ACTIVE_OBSOLETES Requires: $tmpRequire \
Incompatibles: " >> $1
elif [ -n "$tmpIncompat" ]
then
        echo "PATCH_INFO_$Patch_label=Installed: `date` From: `uname -n` \
          Obsoletes: $ACTIVE_OBSOLETES Requires: Incompatibles: \
$tmpIncompat" >> $1
else
        echo "PATCH_INFO_$Patch_label=Installed: `date` From: `uname -n` \
          Obsoletes: $ACTIVE_OBSOLETES Requires: Incompatibles: " >> $1
fi

#
# Since this script is called from the delivery medium and we may be using
# dot extensions to distinguish the different patch packages, this is the
# only place we can, with certainty, trace that source for our backout
# scripts. (Usually $INST_DATADIR would get us there).
#
echo "SCRIPTS_DIR=`dirname $0`" >> $1

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

exit 0
```

Сценарий `preinstall`

Сценарий `preinstall` инициализирует файл `prototype`, информационные файлы и установочные сценарии для создаваемого пакета отмены исправлений. Сценарий очень прост, и оставшиеся сценарии в данном примере позволяют пакету отмены исправлений только описывать обычные файлы.

Если требуется восстановить символьные ссылки, жесткие ссылки, устройства и именованные каналы в пакете отмены исправлений, необходимо изменить сценарий `preinstall` так, чтобы он использовал команду `pkgproto` для сравнения поставленного файла `pkgmap` с установленными файлами, а затем создать запись в файле `prototype` для каждого не являющегося файлом объекта, который будет изменен в пакете отмены исправлений. Метод, который необходимо использовать, похож на метод в сценарии действия над классом.

Сценарии `patch_checkinstall` и `patch_postinstall` вставляются в исходное дерево пакета из сценария `preinstall`. Эти два сценария отменяют выполненные исправления.

```
# This script initializes the backout data for a patch package
# directory format options.
#
#       @(#)preinstall 1.5 96/05/10 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

PATH=/usr/sadm/bin:$PATH
recovery="no"

if [ "$PKG_INSTALL_ROOT" = "/" ]; then
        PKG_INSTALL_ROOT=""
fi

# Check to see if this is a patch installation retry.
if [ "$INTERRUPTION" = "yes" ]; then
    if [ -d "$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST" ] || [ -d \
"$PATCH_BUILD_DIR/$Patch_label.$PKGINST" ]; then
        recovery="yes"
    fi
fi

if [ -n "$PATCH_BUILD_DIR" -a -d "$PATCH_BUILD_DIR" ]; then
        BUILD_DIR="$PATCH_BUILD_DIR/$Patch_label.$PKGINST"
else
        BUILD_DIR="$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST"
fi
```

```
FILE_DIR=$BUILD_DIR/files
RELOC_DIR=$BUILD_DIR/files/reloc
ROOT_DIR=$BUILD_DIR/files/root
PROTO_FILE=$BUILD_DIR/prototype
PKGINFO_FILE=$BUILD_DIR/pkginfo
THIS_DIR=‘dirname $0‘

if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        # If this is being used in an old-style patch, insert
        # the old-style script commands here.

        #XXXOld_CommandsXXX#

        exit 0
fi

#
# Unless specifically denied, initialize the backout patch data by
# creating the build directory and copying over the original pkginfo
# which pkgadd saved in case it had to be restored.
#
if [ "$PATCH_NO_UNDO" != "true" ] && [ "$recovery" = "no" ]; then
        if [ -d $BUILD_DIR ]; then
                rm -r $BUILD_DIR
        fi

        # If this is a retry of the same patch then recovery is set to
        # yes. Which means there is a build directory already in
        # place with the correct backout data.

        if [ "$recovery" = "no" ]; then
                mkdir $BUILD_DIR
                mkdir -p $RELOC_DIR
                mkdir $ROOT_DIR
        fi

        #
        # Here we initialize the backout pkginfo file by first
        # copying over the old pkginfo file and themn adding the
        # ACTIVE_PATCH parameter so the backout will know what patch
        # it’s backing out.
        #
        # NOTE : Within the installation, pkgparam returns the
        # original data.
        #
        pkgparam -v $PKGINST | nawk ’
                $1 ~ /PATCHLIST/        { next; }
```

```
        $1 ~ /PATCH_OBSOLETES/  { next; }
        $1 ~ /ACTIVE_OBSOLETES/ { next; }
        $1 ~ /Obsoletes_label/  { next; }
        $1 ~ /ACTIVE_PATCH/     { next; }
        $1 ~ /Patch_label/      { next; }
        $1 ~ /UPDATE/   { next; }
        $1 ~ /SCRIPTS_DIR/      { next; }
        $1 ~ /PATCH_NO_UNDO/    { next; }
        $1 ~ /INSTDATE/ { next; }
        $1 ~ /PKGINST/  { next; }
        $1 ~ /OAMBASE/  { next; }
        $1 ~ /PATH/     { next; }
        { print; } ' > $PKGINFO_FILE
echo "ACTIVE_PATCH=$Patch_label" >> $PKGINFO_FILE
echo "ACTIVE_OBSOLETES=$ACTIVE_OBSOLETES" >> $PKGINFO_FILE

# And now initialize the backout prototype file with the
# pkginfo file just formulated.
echo "i pkginfo" > $PROTO_FILE

# Copy over the backout scripts including the undo class
# action scripts
for script in $SCRIPTS_DIR/*; do
        srcscript=`basename $script`
        targscript=`echo $srcscript | nawk '
                { script=$0; }
                /u\./ {
                        sub("u.", "i.", script);
                        print script;
                        next;
                }
                /patch_/ {
                        sub("patch_", "", script);
                        print script;
                        next;
                }
                { print "dont_use" } '`
        if [ "$targscript" = "dont_use" ]; then
                continue
        fi

        echo "i $targscript=$FILE_DIR/$targscript" >> $PROTO_FILE
        cp $SCRIPTS_DIR/$srcscript $FILE_DIR/$targscript
done
 #
# Now add entries to the prototype file that won't be passed to
# class action scripts. If the entry is brand new, add it to the
# deletes file for the backout package.
```

```
        #
        Our_Pkgmap=`dirname $SCRIPTS_DIR`/pkgmap
        BO_Deletes=$FILE_DIR/deletes

        nawk -v basedir=${BASEDIR:-/} '
                BEGIN { count=0; }
                {
                        token = $2;
                        ftype = $1;
                }
                $1 ~ /[#\!:]/ { next; }
                $1 ~ /[0123456789]/ {
                        if ( NF >= 3) {
                                token = $3;
                                ftype = $2;
                        } else {
                                next;
                        }
                }
                { if (ftype == "i" || ftype == "e" || ftype == "f" || ftype == \
"v" || ftype == "d") { next; } }
                {
                        equals=match($4, "=")-1;
                        if ( equals == -1 ) { print $3, $4; }
                        else { print $3, substr($4, 0, equals); }
                }
                ' < $Our_Pkgmap | while read class path; do
                        #
                        # NOTE: If pkgproto is passed a file that is
                        # actually a hard link to another file, it
                        # will return ftype "f" because the first link
                        # in the list (consisting of only one file) is
                        # viewed by pkgproto as the source and always
                        # gets ftype "f".
                        #
                        # If this isn't replacing something, then it
                        # just goes to the deletes list.
                        #
                        if valpath -l $path; then
                                Chk_Path="$BASEDIR/$path"
                                Build_Path="$RELOC_DIR/$path"
                                Proto_From="$BASEDIR"
                        else    # It's an absolute path
                                Chk_Path="$PKG_INSTALL_ROOT$path"
                                Build_Path="$ROOT_DIR$path"
                                Proto_From="$PKG_INSTALL_ROOT"
                        fi
                         #
```

```
                        # Hard links have to be restored as regular files.
                        # Unlike the others in this group, an actual
                        # object will be required for the pkgmk.
                        #
                        if [ -f "$Chk_Path" ]; then
                                mkdir -p ‘dirname $Build_Path‘
                                cp $Chk_Path $Build_Path
                                cd $Proto_From
                                pkgproto -c $class "$Build_Path=$path" 1>> \
$PROTO_FILE 2> /dev/null
                                cd $THIS_DIR
                        elif [ -h "$Chk_Path" -o \
                             -c "$Chk_Path" -o \
                             -b "$Chk_Path" -o \
                             -p "$Chk_Path" ]; then
                                pkgproto -c $class "$Chk_Path=$path" 1>> \
$PROTO_FILE 2> /dev/null
                        else
                                echo $path >> $BO_Deletes
                        fi
                done
fi

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

exit 0
```

Сценарий действия над классом

Сценарий действия над классом создает копию каждого файла, который заменяет существующий файл, и добавляет соответствующую строку к файлу `prototype` для пакета отмены исправлений. Все это выполняется очень простыми сценариями `nawk` . Сценарий действия над классом получает список пар источник-адресат, состоящих из обычных файлов, не совпадающих с соответствующими установленными файлами. Символьные ссылки и другие не являющиеся файлами объекты должны обрабатываться в сценарии `preinstall`.

```
# This class action script copies the files being replaced
# into a package being constructed in $BUILD_DIR. This class
# action script is only appropriate for regular files that
# are installed by simply copying them into place.
#
# For special package objects such as editable files, the patch
```

```
# producer must supply appropriate class action scripts.
#
# directory format options.
#
#       @(#)i.script 1.6 96/05/10 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

PATH=/usr/sadm/bin:$PATH

ECHO="/usr/bin/echo"
SED="/usr/bin/sed"
PKGPROTO="/usr/bin/pkgproto"
EXPR="/usr/bin/expr"    # used by dirname
MKDIR="/usr/bin/mkdir"
CP="/usr/bin/cp"
RM="/usr/bin/rm"
MV="/usr/bin/mv"

recovery="no"
Pn=$$
procIdCtr=0

CMDS_USED="$ECHO $SED $PKGPROTO $EXPR $MKDIR $CP $RM $MV"
LIBS_USED=""

if [ "$PKG_INSTALL_ROOT" = "/" ]; then
        PKG_INSTALL_ROOT=""
fi

# Check to see if this is a patch installation retry.
if [ "$INTERRUPTION" = "yes" ]; then
        if [ -d "$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST" ] ||
\
[ -d "$PATCH_BUILD_DIR/$Patch_label.$PKGINST" ]; then
                recovery="yes"
        fi
fi

if [ -n "$PATCH_BUILD_DIR" -a -d "$PATCH_BUILD_DIR" ]; then
        BUILD_DIR="$PATCH_BUILD_DIR/$Patch_label.$PKGINST"
else
        BUILD_DIR="$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST"
fi

FILE_DIR=$BUILD_DIR/files
```

```
RELOC_DIR=$FILE_DIR/reloc
ROOT_DIR=$FILE_DIR/root
BO_Deletes=$FILE_DIR/deletes
PROGNAME=‘basename $0‘

if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        PATCH_NO_UNDO="true"
fi

# Since this is generic, figure out the class.
Class=‘echo $PROGNAME | nawk ’ { print substr($0, 3)  }’‘

# Since this is an update, $BASEDIR is guaranteed to be correct
BD=${BASEDIR:-/}

cd $BD

#
# First, figure out the dynamic libraries that can trip us up.
#
if [ -z "$PKG_INSTALL_ROOT" ]; then
        if [ -x /usr/bin/ldd ]; then
                LIB_LIST=‘/usr/bin/ldd $CMDS_USED | sort -u | nawk ’
                        $1 ~ /\// { continue; }
                        { printf "%s ", $3 } ’‘
        else
                LIB_LIST="/usr/lib/libc.so.1 /usr/lib/libdl.so.1
\
/usr/lib/libw.so.1 /usr/lib/libintl.so.1 /usr/lib/libadm.so.1 \
/usr/lib/libelf.so.1"
        fi
fi

#
# Now read the list of files in this class to be replaced. If the file
# is already in place, then this is a change and we need to copy it
# over to the build directory if undo is allowed. If it’s a new entry
# (No $dst), then it goes in the deletes file for the backout package.
#
procIdCtr=0
while read src dst; do
        if [ -z "$PKG_INSTALL_ROOT" ]; then
                Chk_Path=$dst
                for library in $LIB_LIST; do
                        if [ $Chk_Path = $library ]; then
                                $CP $dst $dst.$Pn
                                LIBS_USED="$LIBS_USED $dst.$Pn"
                                LD_PRELOAD="$LIBS_USED"
```

```
                        export LD_PRELOAD
                fi
        done
fi

if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        # If this is being used in an old-style patch, insert
        # the old-style script commands here.

        #XXXOld_CommandsXXX#
        echo >/dev/null # dummy
fi

if [ "${PATCH_NO_UNDO}" != "true" ]; then
        #
        # Here we construct the path to the appropriate source
        # tree for the build. First we try to strip BASEDIR. If
        # there's no BASEDIR in the path, we presume that it is
        # absolute and construct the target as an absolute path
        # by stripping PKG_INSTALL_ROOT. FS_Path is the path to
        # the file on the file system (for deletion purposes).
        # Build_Path is the path to the object in the build
        # environment.
        #
        if [ "$BD" = "/" ]; then
                FS_Path=`$ECHO $dst | $SED s@"$BD"@@`
        else
                FS_Path=`$ECHO $dst | $SED s@"$BD/"@@`
        fi

        # If it's an absolute path the attempt to strip the
        # BASEDIR will have failed.
        if [ $dst = $FS_Path ]; then
                if [ -z "$PKG_INSTALL_ROOT" ]; then
                        FS_Path=$dst
                        Build_Path="$ROOT_DIR$dst"
                else
                        Build_Path="$ROOT_DIR`echo $dst | \
                            sed s@"$PKG_INSTALL_ROOT"@@`"
                        FS_Path=`echo $dst | \
                            sed s@"$PKG_INSTALL_ROOT"@@`
                fi
        else
                Build_Path="$RELOC_DIR/$FS_Path"
        fi

        if [ -f $dst ]; then    # If this is replacing something
                cd $FILE_DIR
```

```
                        #
                        # Construct the prototype file entry. We replace
                        # the pointer to the filesystem object with the
                        # build directory object.
                        #
                        $PKGPROTO -c $Class $dst=$FS_Path | \
                            $SED -e s@=$dst@=$Build_Path@ >> \
                            $BUILD_DIR/prototype

                        # Now copy over the file
                        if [ "$recovery" = "no" ]; then
                                DirName=‘dirname $Build_Path‘
                                $MKDIR -p $DirName
                                $CP -p $dst $Build_Path
                        else
                                # If this file is already in the build area skip it
                                if [ -f "$Build_Path" ]; then
                                        cd $BD
                                        continue
                                else
                                        DirName=‘dirname $Build_Path‘
                                        if [ ! -d "$DirName" ]; then
                                                $MKDIR -p $DirName
                                        fi
                                        $CP -p $dst $Build_Path
                                fi
                        fi

                        cd $BD
                else    # It’s brand new
                        $ECHO $FS_Path >> $BO_Deletes
                fi
        fi

        # If special processing is required for each src/dst pair,
        # add that here.
        #
        #XXXSpecial_CommandsXXX#
        #

        $CP $src $dst.$$$procIdCtr
        if [ $? -ne 0 ]; then
                $RM $dst.$$$procIdCtr 1>/dev/null 2>&1
        else
                $MV -f $dst.$$$procIdCtr $dst
                for library in $LIB_LIST; do
                        if [ "$library" = "$dst" ]; then
                                LD_PRELOAD="$dst"
```

```
                                export LD_PRELOAD
                        fi
                done
        fi
        procIdCtr=‘expr $procIdCtr + 1‘
done

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

#
# Release the dynamic libraries
#
for library in $LIBS_USED; do
        $RM -f $library
done

exit 0
```

Сценарий `postinstall`

Сценарий `postinstall` создает пакет отмены исправлений с использованием информации, предоставленной другими сценариями. Поскольку командам `pkgmk` и `pkgtrans` не требуется база данных пакета, их можно выполнять в установке пакета.

В приводимом примере отмена исправлений допустима путем создания пакета в формате потока в каталоге сохранения (с помощью переменной среды `PKGSAV`). Это совсем не очевидно, однако этот пакет должен быть в формате потока, потому что каталог сохранения перемещается в ходе выполнения команды `pkgadd`. Если команда `pkgadd` применена к пакету в его собственном каталоге сохранения, любые предположения о том, где находится источник пакета в любой момент времени, становятся очень ненадежными. Пакет в формате потока распаковывается во временный каталог и устанавливается из него. (Пакет в формате каталога начал бы установку из каталога сохранения и внезапно оказался бы перемещенным в ходе выполнения отказоустойчивой команды `pkgadd`.)

Чтобы определить, какие исправления применяются к пакету, используйте следующую команду:

```
$ pkgparam SUNWstuf PATCHLIST
```

За исключением параметра `PATCHLIST`, который является общедоступным интерфейсом Sun, в данном примере нет ничего особенного в именах параметров. Вместо `PATCH` можно использовать традиционный `SUNW_PATCHID`, а различные другие списки, например `PATCH_EXCL` и `PATCH_REQD`, можно соответствующим образом переименовать.

Если определенные пакеты исправлений зависят от других пакетов исправлений, доступных на том же носителе, то сценарий `checkinstall` может обнаружить это и создать сценарий, который будет выполняться сценарием `postinstall` так же, как и в примере с обновлением (см. раздел «Обновление пакетов» на стр. 188).

```
# This script creates the backout package for a patch package
#
# directory format options.
#
# @(#) postinstall 1.6 96/01/29 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

# Description:
#       Set the TYPE parameter for the remote file
#
# Parameters:
#       none
#
# Globals set:
#       TYPE

set_TYPE_parameter () {
        if [ ${PATCH_UNDO_ARCHIVE:?????} = "/dev" ]; then
                # handle device specific stuff
                TYPE="removable"
        else
                TYPE="filesystem"
        fi
}

#
# Description:
#       Build the remote file that points to the backout data
#
# Parameters:
#       $1:     the un/compressed undo archive
#
# Globals set:
#       UNDO, STATE

build_remote_file () {
        remote_path=$PKGSAV/$Patch_label/remote
        set_TYPE_parameter
        STATE="active"
```

```
        if [ $1 = "undo" ]; then
                UNDO="undo"
        else
                UNDO="undo.Z"
        fi

        cat > $remote_path << EOF
# Backout data stored remotely
TYPE=$TYPE
FIND_AT=$ARCHIVE_DIR/$UNDO
STATE=$STATE
EOF
}

PATH=/usr/sadm/bin:$PATH

if [ "$PKG_INSTALL_ROOT" = "/" ]; then
        PKG_INSTALL_ROOT=""
fi

if [ -n "$PATCH_BUILD_DIR" -a -d "$PATCH_BUILD_DIR" ]; then
        BUILD_DIR="$PATCH_BUILD_DIR/$Patch_label.$PKGINST"
else
        BUILD_DIR="$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST"
fi

if [ ! -n "$PATCH_UNDO_ARCHIVE" ]; then
        PATCH_UNDO_ARCHIVE="none"
fi

FILE_DIR=$BUILD_DIR/files
RELOC_DIR=$FILE_DIR/reloc
ROOT_DIR=$FILE_DIR/root
BO_Deletes=$FILE_DIR/deletes
THIS_DIR=‘dirname $0‘
PROTO_FILE=$BUILD_DIR/prototype
TEMP_REMOTE=$PKGSAV/$Patch_label/temp

if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        # remove the scripts that are left behind
        install_scripts=‘dirname $0‘
        rm $install_scripts/checkinstall \
$install_scripts/patch_checkinstall $install_scripts/patch_postinstall

        # If this is being used in an old-style patch, insert
        # the old-style script commands here.

        #XXXOld_CommandsXXX#
```

```
        exit 0
fi
#
# At this point we either have a deletes file or we don't. If we do,
# we create a prototype entry.
#
if [ -f $BO_Deletes ]; then
        echo "i deletes=$BO_Deletes" >> $BUILD_DIR/prototype
fi

#
# Now delete everything in the deletes list after transferring
# the file to the backout package and the entry to the prototype
# file. Remember that the pkgmap will get the CLIENT_BASEDIR path
# but we have to actually get at it using the BASEDIR path. Also
# remember that removef will import our PKG_INSTALL_ROOT
#
Our_Deletes=$THIS_DIR/deletes
if [ -f $Our_Deletes ]; then
        cd $BASEDIR

        cat $Our_Deletes | while read path; do
                Reg_File=0

                if valpath -l $path; then
                        Client_Path="$CLIENT_BASEDIR/$path"
                        Build_Path="$RELOC_DIR/$path"
                        Proto_Path=$BASEDIR/$path
                else    # It's an absolute path
                        Client_Path=$path
                        Build_Path="$ROOT_DIR$path"
                        Proto_Path=$PKG_INSTALL_ROOT$path
                fi

                # Note: If the file isn't really there, pkgproto
                # doesn't write anything.
                LINE=`pkgproto $Proto_Path=$path`
                ftype=`echo $LINE | nawk '{ print $1 }'`
                if [ $ftype = "f" ]; then
                        Reg_File=1
                fi

                if [ $Reg_File = 1 ]; then
                        # Add source file to the prototype entry
                        if [ "$Proto_Path" = "$path" ]; then
                                LINE=`echo $LINE | sed -e s@$Proto_Path@$Build_Path@2`
                        else
```

```
                                LINE=‘echo $LINE | sed -e s@$Proto_Path@$Build_Path@‘
                        fi

                        DirName=‘dirname $Build_Path‘
                        # make room in the build tree
                        mkdir -p $DirName
                        cp -p $Proto_Path $Build_Path
                fi

                # Insert it into the prototype file
                echo $LINE 1>>$PROTO_FILE 2>/dev/null

                # Remove the file only if it’s OK’d by removef
                rm ‘removef $PKGINST $Client_Path‘ 1>/dev/null 2>&1
        done
        removef -f $PKGINST

        rm $Our_Deletes
fi

#
# Unless specifically denied, make the backout package.
#
if [ "$PATCH_NO_UNDO" != "true" ]; then
        cd $BUILD_DIR   # We have to build from here.

        if [ "$PATCH_UNDO_ARCHIVE" != "none" ]; then
                STAGE_DIR="$PATCH_UNDO_ARCHIVE"
                ARCHIVE_DIR="$PATCH_UNDO_ARCHIVE/$Patch_label/$PKGINST"
                mkdir -p $ARCHIVE_DIR
                mkdir -p $PKGSAV/$Patch_label
        else
                if [ -d $PKGSAV/$Patch_label ]; then
                        rm -r $PKGSAV/$Patch_label
                fi
                STAGE_DIR=$PKGSAV
                ARCHIVE_DIR=$PKGSAV/$Patch_label
                mkdir $ARCHIVE_DIR
        fi

        pkgmk -o -d $STAGE_DIR 1>/dev/null 2>&1
        pkgtrans -s $STAGE_DIR $ARCHIVE_DIR/undo $PKG 1>/dev/null 2>&1
        compress $ARCHIVE_DIR/undo
        retcode=$?
        if [ "$PATCH_UNDO_ARCHIVE" != "none" ]; then
                if [ $retcode != 0 ]; then
                        build_remote_file "undo"
                else
```

```
                        build_remote_file "undo.Z"
                fi
        fi
        rm -r $STAGE_DIR/$PKG

        cd ..
        rm -r $BUILD_DIR
        # remove the scripts that are left behind
        install_scripts=`dirname $0`
        rm $install_scripts/checkinstall $install_scripts/patch_\
checkinstall $install_scripts/patch_postinstall
fi

#
# Since this apparently worked, we'll mark as obsoleted the prior
# versions of this patch - installpatch deals with explicit obsoletions.
#
cd ${PKG_INSTALL_ROOT:-/}
cd var/sadm/pkg

active_base=`echo $Patch_label | nawk '
        { print substr($0, 1, match($0, "Patchvers_pfx")-1) } '`

List=`ls -d $PKGINST/save/${active_base}*`
if [ $? -ne 0 ]; then
        List=""
fi

for savedir in $List; do
        patch=`basename $savedir`
        if [ $patch = $Patch_label ]; then
                break
        fi

        # If we get here then the previous patch gets deleted
        if [ -f $savedir/undo ]; then
                mv $savedir/undo $savedir/obsolete
                echo $Patch_label >> $savedir/obsoleted_by
        elif [ -f $savedir/undo.Z ]; then
                mv $savedir/undo.Z $savedir/obsolete.Z
                echo $Patch_label >> $savedir/obsoleted_by
        elif  [ -f $savedir/remote ]; then
                `grep . $PKGSAV/$patch/remote | sed 's/STATE=.*/STATE=obsolete/
' > $TEMP_REMOTE`
                rm -f $PKGSAV/$patch/remote
                mv $TEMP_REMOTE $PKGSAV/$patch/remote
                rm -f $TEMP_REMOTE
                echo $Patch_label >> $savedir/obsoleted_by
```

```
        elif  [ -f $savedir/obsolete -o -f $savedir/obsolete.Z ]; then
                echo $Patch_label >> $savedir/obsoleted_by
        fi
done

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

exit 0
```

Сценарий patch_checkinstall

```
# checkinstall script to validate backing out a patch.
# directory format option.
#
#       @(#)patch_checkinstall 1.2 95/10/10 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

PATH=/usr/sadm/bin:$PATH

LATER_MSG="PaTcH_MsG 6 ERROR: A later version of this patch is applied."
NOPATCH_MSG="PaTcH_MsG 2 ERROR: Patch number $ACTIVE_PATCH is not installed"
NEW_LIST=""

# Get OLDLIST
. $1

#
# Confirm that the patch that got us here is the latest one installed on
# the system and remove it from PATCHLIST.
#
Is_Inst=0
Skip=0
active_base=‘echo $ACTIVE_PATCH | nawk ’
        { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
active_inst=‘echo $ACTIVE_PATCH | nawk ’
        { print substr($0, match($0, "Patchvers_pfx")+1) } ’‘
for patchappl in ${OLDLIST}; do
        appl_base=‘echo $patchappl | nawk ’
                { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
        if [ $appl_base = $active_base ]; then
                appl_inst=‘echo $patchappl | nawk ’
```

```
                        { print substr($0, match($0, "Patchvers_pfx")+1) } ’‘
                result=‘expr $appl_inst \> $active_inst‘
                if [ $result -eq 1 ]; then
                        puttext "$LATER_MSG"
                        exit 3
                elif [ $appl_inst = $active_inst ]; then
                        Is_Inst=1
                        Skip=1
                fi
        fi
        if [ $Skip = 1 ]; then
                Skip=0
        else
                NEW_LIST="${NEW_LIST} $patchappl"
        fi
done

if [ $Is_Inst = 0 ]; then
        puttext "$NOPATCH_MSG"
        exit 3
fi

#
# OK, all’s well. Now condition the key variables.
#
echo "PATCHLIST=${NEW_LIST}" >> $1
echo "Patch_label=" >> $1
echo "PATCH_INFO_$ACTIVE_PATCH=backed out" >> $1

# Get the current PATCH_OBSOLETES and condition it
Old_Obsoletes=$PATCH_OBSOLETES

echo $ACTIVE_OBSOLETES | sed ’y/\ /\n/’ | \
nawk -v PatchObsList="$Old_Obsoletes" ’
        BEGIN {
                printf("PATCH_OBSOLETES=");
                PatchCount=split(PatchObsList, PatchObsComp, " ");

                for(PatchIndex in PatchObsComp) {
                        Atisat=match(PatchObsComp[PatchIndex], "@");
                        PatchObs[PatchIndex]=substr(PatchObsComp[PatchIndex], \
0, Atisat-1);
                        PatchObsCnt[PatchIndex]=substr(PatchObsComp\
[PatchIndex], Atisat+1);
                }
        }
        {
                for(PatchIndex in PatchObs) {
```

```
                        if (PatchObs[PatchIndex] == $0) {
                                PatchObsCnt[PatchIndex]=PatchObsCnt[PatchIndex]-1;
                        }
                }
                next;
        }
        END {
                for(PatchIndex in PatchObs) {
                        if ( PatchObsCnt[PatchIndex] > 0 ) {
                                printf("%s@%d ", PatchObs[PatchIndex], PatchObsCnt\
[PatchIndex]);
                        }
                }
                printf("\n");
        } ' >> $1

        # remove the used parameters
        echo "ACTIVE_OBSOLETES=" >> $1
        echo "Obsoletes_label=" >> $1

exit 0
```

Сценарий patch_postinstall

```
# This script deletes the used backout data for a patch package
# and removes the deletes file entries.
#
# directory format options.
#
#       @(#)patch_postinstall 1.2 96/01/29 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#
PATH=/usr/sadm/bin:$PATH
THIS_DIR=‘dirname $0‘

Our_Deletes=$THIS_DIR/deletes

#
# Delete the used backout data
#
if [ -f $Our_Deletes ]; then
        cat $Our_Deletes | while read path; do
                if valpath -l $path; then
                        Client_Path=‘echo "$CLIENT_BASEDIR/$path" | sed s@//@/@‘
                else    # It’s an absolute path
```

```
                        Client_Path=$path
                fi
                rm `removef $PKGINST $Client_Path`
        done
        removef -f $PKGINST

        rm $Our_Deletes
fi

#
# Remove the deletes file, checkinstall and the postinstall
#
rm -r $PKGSAV/$ACTIVE_PATCH
rm -f $THIS_DIR/checkinstall $THIS_DIR/postinstall

exit 0
```

Обновление пакетов

Процесс обновления пакета сильно отличается от процесса перезаписи пакета. Несмотря на то, что имеются специальные средства, позволяющие обновлять стандартные пакеты, поставляемые как часть ОС Solaris, несобранный пакет может быть разработан с поддержкой собственного обновления – в некоторых предыдущих примерах описывались пакеты, подготовленные к будущим обновлениям и контролирующие точный метод установки под руководством администратора. Кроме того, можно создать сценарий `request`, который будет поддерживать прямое обновление пакета. Если администратор решит установить один пакет так, чтобы полностью удалить другой, не оставляя лишних устаревших файлов, сценарии пакета могут справиться с этой задачей.

Сценарии `request` и `postinstall` в приводимом примере предоставляют простой обновляемый пакет. Сценарий `request` открывает диалог с администратором, а затем создает простой файл в каталоге `/tmp`, служащий для удаления экземпляра старого пакета. (Хотя сценарий `request` и создает файл (что запрещено), в нашем случае это неважно, поскольку у всех есть доступ к каталогу `/tmp`).

Затем сценарий `postinstall` запускает сценарий интерпретатора команд в каталоге `/tmp`, который выполняет необходимую команду `pkgrm` на старом пакете и после этого удаляет себя.

Данный пример иллюстрирует базовое обновление. В нем менее пятидесяти строк кода, включая несколько довольно длинных сообщений. Его функциональность можно расширить, например, чтобы отменить обновление или произвести другие важные изменения пакета по требованию разработчика.

Разработчик пользовательского интерфейса для обновления должен быть абсолютно уверен, что администратор полностью осознает процесс и действительно запросил

обновление, а не параллельную установку экземпляра. Нет ничего страшного в выполнении хорошо понятой сложной операции, такой как обновление, при условии, что пользовательский интерфейс делает эту операцию понятной.

Сценарий `request`

```
# request script
control an upgrade installation

PATH=/usr/sadm/bin:$PATH
UPGR_SCRIPT=/tmp/upgr.$PKGINST

UPGRADE_MSG="Do you want to upgrade the installed version ?"

UPGRADE_HLP="If upgrade is desired, the existing version of the \
    package will be replaced by this version. If it is not \
    desired, this new version will be installed into a different \
    base directory and both versions will be usable."

UPGRADE_NOTICE="Conflict approval questions may be displayed. The \
    listed files are the ones that will be upgraded. Please \
    answer \"y\" to these questions if they are presented."

pkginfo -v 1.0 -q SUNWstuf.\*

if [ $? -eq 0 ]; then
      # See if upgrade is desired here
      response=‘ckyorn -p "$UPGRADE_MSG" -h "$UPGRADE_HLP"‘
      if [ $response = "y" ]; then
            OldPkg=‘pkginfo -v 1.0 -x SUNWstuf.\* | nawk ’ \
            /SUNW/{print $1} ’‘
            # Initiate upgrade
            echo "PATH=/usr/sadm/bin:$PATH" > $UPGR_SCRIPT
            echo "sleep 3" >> $UPGR_SCRIPT
            echo "echo Now removing old instance of $PKG" >> \
            $UPGR_SCRIPT
            if [ ${PKG_INSTALL_ROOT} ]; then
                  echo "pkgrm -n -R $PKG_INSTALL_ROOT $OldPkg" >> \
                  $UPGR_SCRIPT
            else
                  echo "pkgrm -n $OldPkg" >> $UPGR_SCRIPT
            fi
            echo "rm $UPGR_SCRIPT" >> $UPGR_SCRIPT
            echo "exit $?" >> $UPGR_SCRIPT

            # Get the original package’s base directory
            OldBD=‘pkgparam $OldPkg BASEDIR‘
```

```
            echo "BASEDIR=$OldBD" > $1
            puttext -l 5 "$UPGRADE_NOTICE"
      else
            if [ -f $UPGR_SCRIPT ]; then
                  rm -r $UPGR_SCRIPT
            fi
      fi
fi

exit 0
```

Сценарий `postinstall`

```
# postinstall
to execute a simple upgrade

PATH=/usr/sadm/bin:$PATH
UPGR_SCRIPT=/tmp/upgr.$PKGINST

if [ -f $UPGR_SCRIPT ]; then
      sh $UPGR_SCRIPT &
fi

exit 0
```

Создание пакетов с архивом класса

Пакетом с архивом класса, который является дополнением к двоичному интерфейсу приложений (ABI), называется пакет, в котором определенные наборы файлов были объединены в отдельные файлы или архивы и при необходимости сжаты или зашифрованы. Форматы архива класса увеличивают исходную скорость установки на величину до 30% и улучшают надежность в ходе установки пакетов и исправлений на потенциально активные файловые системы.

В следующих разделах приведена информация о структуре каталогов архивного пакета, ключевых словах и о служебной программе `faspac`.

Структура каталога архивного пакета

Элементы пакета, показанные на рисунке ниже, представляет собой каталог, содержащий файлы пакета. Этот каталог должен называться так же, как и пакет.

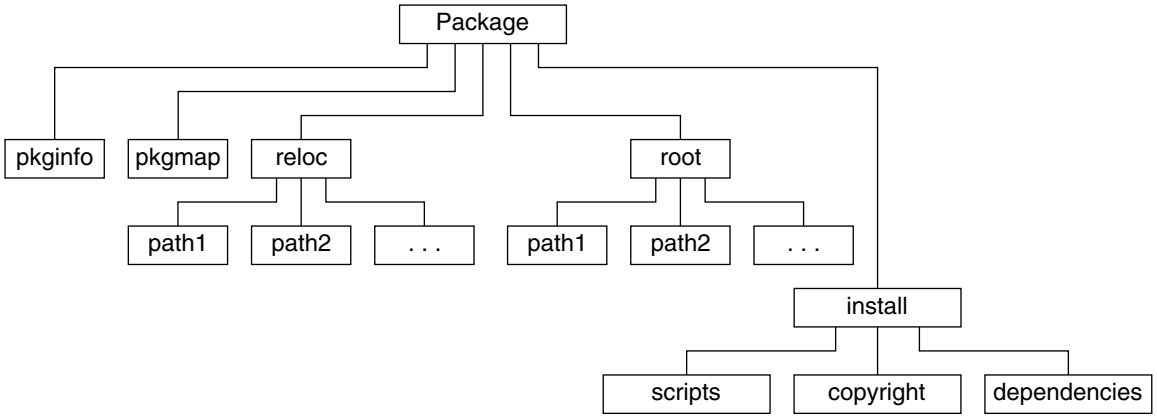


РИСУНОК 6–1 Структура каталога пакета

В следующей таблице перечислены функции файлов и каталогов, содержащихся в каталоге пакета.

Элемент	Описание
`pkginfo`	Файл, описывающий пакет в целом, включая особые переменные среды и установочные команды
`pkgmap`	Файл с описанием каждого устанавливаемого объекта, например, файла, каталога или канала
`reloc`	Необязательный каталог, содержащий файлы, которые должны быть установлены относительно базового каталога (перемещаемые объекты)
`root`	Необязательный каталог, содержащий файлы, которые будут установлены относительно каталога `root` (корневые объекты)
`install`	Необязательный каталог, содержащий сценарии и другие вспомогательные файлы (за исключением файлов `pkginfo` и `pkgmap`, все файлы типа `ftype i` находятся здесь)

Формат архива класса позволяет разработчику пакета объединить файлы из каталогов `reloc` и `root` в архивы, которые могут быть сжаты, зашифрованы или иным образом обработаны с тем, чтобы увеличить скорость установки, уменьшить размер пакета или увеличить его безопасность.

Интерфейс ABI позволяет назначить любой файл пакета какому-либо классу. Все файлы определенного класса могут быть установлены на диск с помощью особого метода, определенного сценарием действия над классом. Этот особый метод может использовать программы, доступные на целевой системе, или программы, поставляемые с пакетом. Результирующий формат похож на стандартный формат интерфейса ABI. На следующем

рисунке видно, что добавлен еще один каталог. Любые классы файлов, предназначенных для архивирования, просто объединяется в один файл и помещаются в каталог `archive`. Все архивированные файлы удаляются из каталогов `reloc` и `root`, а в каталог `install` помещается установочный сценарий действия над классом.

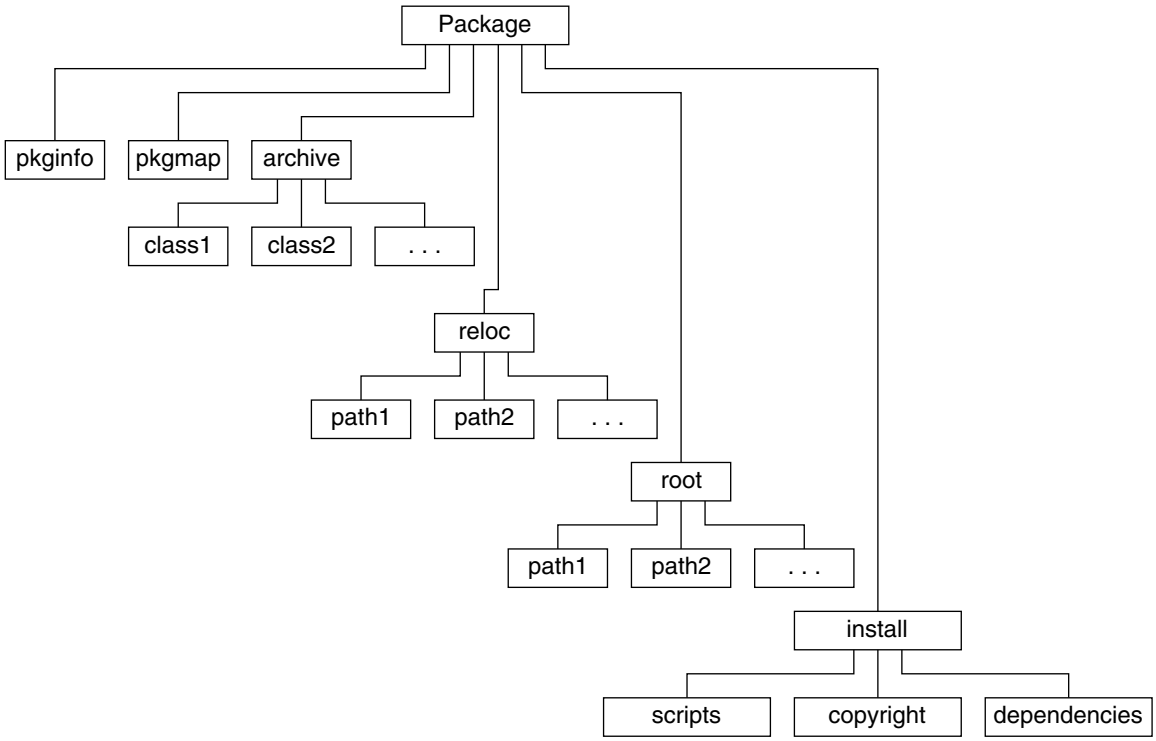


РИСУНОК 6–2 Структура каталогов архивного пакета

Ключевые слова, используемые в пакетах с архивом классов

Для поддержки этого нового формата архива класса, трем новым интерфейсам в виде ключевых слов присвоено особое значение в файле `pkginfo`. Эти ключевые слова используются для обозначения классов, требующих специального обращения. Формат каждой строки с ключевым словом таков: `keyword=class1[class2 class3 ...]`. Значения каждого ключевого слова определены в следующей таблице.

Ключевое слово	Описание
PKG_SRC_NOVERIFY	Сообщает команде pkgadd что не следует проверять существование и свойства файлов в поставляемых каталогах reloc или root пакета, если они принадлежат к именованному классу. Это требуется для всех заархивированных классов, поскольку эти файлы больше не содержатся в каталогах reloc или root. Они превратились в файл частного формата в каталоге archive.
PKG_DST_QKVERIFY	Файлы этого класса проверяются после установки с помощью быстрого алгоритма и почти не имеют текстового вывода. Функция быстрой проверки сначала правильно устанавливает атрибуты каждого файла, а затем проверяет, удачно ли прошла операция. Кроме того, существует тест на размер файла и время его изменения, результаты которого сравниваются с величинами, указанными в файле pkgmap. Проверка контрольной суммы не производится, и система восстановления после ошибки хуже, чем система, предлагаемая стандартным механизмом проверки. В случае отказа питания или сбоя диска в ходе установки содержимое может не соответствовать установленным файлам. Данное несоответствие всегда можно разрешить с помощью команды pkgrm.
PKG_CAS_PASSRELATIVE	Обычно установочный сценарий действия над классом получает из stdin (стандартного входного потока) список пар источник-адресат, который сообщает ему, какие файлы следует устанавливать. Классы, назначенные PKG_CAS_PASSRELATIVE, не получают пар источник-адресат. Вместо этого они получают список, первой записью в котором является расположение исходного пакета, а остальная часть списка - пути конечного каталога. Это сделано специально для упрощения процесса извлечения из архива. По положению исходного пакета можно найти архив в каталоге archive. После этого пути конечного каталога передаются в функцию, которая отвечает за извлечение содержимого архива. Каждый представленный путь конечного каталога является либо абсолютным, либо относительным по отношению к базовому каталогу в зависимости от того, располагался ли первоначально этот путь в каталоге root или reloc. При выборе этого варианта могут возникнуть сложности при объединении относительных и абсолютных путей в один класс.

Для каждого заархивированного класса требуется сценарий действия над классом. Это файл, содержащий команды интерпретатора sh, который выполняется командой pkgadd для фактической установки файлов из архива. Если сценарий действия над классом находится в каталоге install пакета, команда pkgadd передает всю ответственность за установку этому сценарию. Сценарий действия над классом выполняется с

полномочиями администратора (пользователя root) и может помещать свои файлы практически в любое место целевой системы.

Примечание – Единственное ключевое слово, являющееся абсолютно обязательным для создания пакета с архивом класса, - это `PKG_SRC_NOVERIFY`. Оставшиеся ключевые слова могут использоваться для увеличения скорости установки или для сохранения программного кода.

Утилита `faspac`

Утилита `faspac` преобразует стандартный пакет интерфейса ABI в формат архива класса, используемый в связанных пакетах. Эта утилита осуществляет архивирование с помощью cpio и сжатие с помощью compress. В результирующем пакете появляется дополнительный каталог верхнего уровня с названием `archive`. В этом каталоге будут находиться все архивы, названные по имени класса. В каталоге `install` будут находиться сценарии действий над классами, необходимые для распаковки каждого архива. Абсолютные пути не архивируются.

Вызов служебной программы `faspac` имеет следующий формат:

`faspac [-m` *Archive Method*`] -a -s -q [-d` *Base Directory*`] /`
`[-x` *Exclude List*`] [`*List of Packages*`]`

Каждый параметр команды `faspac` описан в следующей таблице.

Параметр	Описание
`-m` *Способ архивирования*	Указывает на способ архивирования или сжатия. По умолчанию для сжатия используется утилита `bzip2`. Для переключения на метод zip или unzip используйте параметр `-m zip`, либо (для cpio и compress) параметр `-m cpio`.
`-a`	Исправляет атрибуты (необходимы полномочия администратора (пользователя root)).
`-s`	Означает перевод пакета в стандартный тип ABI. Данный параметр берет пакеты cpio или compresssed и переводит их в стандартный формат пакета интерфейса ABI.
`-q`	Означает режим без выдачи сообщений.

Параметр	Описание
-d *Базовый каталог*	Указывает каталог, в котором все имеющиеся пакеты будут обрабатываться в соответствии с требованиями, указанными в командной строке. Этот параметр является взаимоисключающим с параметром *List of Packages*.
-x *Список исключений*	Выдает разделенный запятыми, кавычками или пробелами список пакетов, которые следует исключить из процесса обработки.
Список пакетов	Выдает список пакетов, которые необходимо обработать.

Глоссарий

ABI См. двоичный интерфейс приложений (ABI).

ASN.1 См. абстрактная синтаксическая нотация версии 1 (ASN.1)

DER См. правила различенной кодировки.

PEM См. конфиденциальное почтовое сообщение PEM.

PKCS12 См. стандарт криптографии открытых ключей #12.

PKCS7 См. стандарт криптографии открытых ключей #7.

`tar` Извлечение из ленточного архива (Tape archive retrieval). Команда Solaris для добавления или извлечения файлов с носителя.

X.509 См. Рекомендации X.509 ITU-T.

Рекомендация X.509 ITU-T Протокол, описывающий принятый синтаксис сертификата открытого ключа X.509.

аббревиатура пакета Краткое название пакета, определяемое параметром `PKG` в файле `pkginfo`.

абстрактная синтаксическая нотация версии 1 Способ описания абстрактных объектов. Например, ASN.1 определяет сертификат открытого ключа, все объекты, составляющие сертификат и порядок, в котором эти объекты установлены. Однако ASN.1 не указывает, каким образом эти объекты сериализуются для хранения или передачи.

авторское право Право владения и продажи интеллектуальной собственности, например программного обеспечения, исходного кода или документации. Право собственности должно быть указано на компакт-диске и в тексте на вкладыше. Следует указать, является ли владельцем авторских прав компания SunSoft или другая компания. Сведения об обладании авторскими правами указывается также и в документации компании SunSoft.

базовый каталог Место, куда будут устанавливаться перемещаемые объекты. Оно определяется в файле `pkginfo` с помощью параметра `BASEDIR`.

время сборки Время, в течение которого производится сборка пакета с помощью команды `pkgmk`.

время установки Время, в течение которого пакет устанавливается с помощью команды `pkgadd`.

двоичный интерфейс приложений Определение двоичного интерфейса между скомпилированными приложениями и операционной системой, на которой они выполняются.

доверенный сертификат Сертификат, содержащий один сертификат открытого ключа, принадлежащий другой организации. Доверенные сертификаты используются для проверки цифровых подписей и при инициировании подключения к защищенному серверу (SSL).

зависимый пакет Пакет, зависящий от существования другого пакета. См. также файл `depend`.

закрытый ключ Ключ шифрования/дешифрования, известный только стороне или сторонам, обменивающимся секретными сообщениями. Закрытый ключ используется совместно с открытыми ключами для создания подписанных пакетов.

идентификатор пакета Цифровой суффикс, добавляемый к аббревиатуре пакета командой `pkgadd`.

индивидуально перемещаемый объект Объект пакета, который не ограничен размещением в том же самом каталоге, что и коллективно перемещаемый объект. Он определяется с помощью установочной переменной в поле *path* (путь) в файле `prototype`, а место установки определяется сценариями `request` или `checkinstall`.

информационный файл Файл, который может определять зависимости пакета, содержать сообщение об авторских правах или резервировать дисковое пространство на целевой системе.

класс Имя, используемое для группировки объектов пакета. См. также сценарий действия над классом.

ключ пользователя Ключ, содержащий секретную информацию криптографического ключа. Эта информация хранится в защищенном формате для предотвращения несанкционированного использования. Ключи пользователя используются при создании подписанного пакета.

коллективно перемещаемый объект Объект пакета, который расположен в месте, связанном с общим установочным каталогом. См. также базовый каталог.

контрольный файл Файл, который осуществляет контроль над тем, как и куда устанавливается пакет и устанавливается ли он вообще. См. информационный файл и сценарий установки.

конфиденциальное почтовое сообщение Способ кодирования файла с помощью кодировки Base64 и нескольких дополнительных заголовков. Широко используется для кодирования сертификатов и открытых ключей в файле, существующем в файловой системе или в сообщении электронной почты.

неподписанный пакет Обычный пакет интерфейса ABI без какого-либо шифрования и без цифровых подписей.

несовместимый пакет Пакет, который несовместим с указанным пакетом. См. также файл `depend`.

обратная зависимость Состояние, при котором другой пакет зависит от наличия вашего пакета. См. также файл `depend`.

общее имя Псевдоним, записанный в хранилище ключей пакета у подписанных пакетов.

объект пакета Другое название файла приложения, содержащегося в пакете и подлежащего установке на целевую систему.

открытый ключ Значение, генерируемое в виде ключа шифрования, которое, совместно с закрытым ключом, полученным из открытого ключа, может использоваться для надежного шифрования сообщений и цифровых подписей.

пакет Набор файлов и каталогов, необходимых для программного приложения.

параметрическое название пути Имя пути, включающее спецификацию переменной.

переменная сборки Переменная начинается со строчной буквы и вычисляется в период сборки.

переменная установки Переменная начинается с заглавной буквы и вычисляется в период установки.

перемещаемый Объект пакета, определенный в файле `prototype`, имеющий относительное имя пути.

перемещаемый объект Объект пакета, не требующий абсолютного пути в целевой системе. Вместо этого его местоположение определяется в ходе процесса установки. См. также коллективно перемещаемый объект и индивидуально перемещаемый объект.

подписанные пакеты Обычный пакет формата потока с цифровой подписью, которая проверяет следующее: что пакет пришел от отправителя, его подписавшего, отправитель действительно подписал его, пакет не изменялся со времени его подписания отправителем и что отправитель, подписавший пакет, является доверенным отправителем.

правила различенной кодировки Двоичное представление объекта ASN.1. Определяет, каким образом объект ASN.1 сериализуется для хранения или передачи в вычислительной среде. Используется с подписанными пакетами.

процедурный сценарий Сценарий, определяющий действия, совершаемые на определенном этапе установки и удаления пакета.

сегментированный Пакет, не умещающийся на один том, например на гибкий диск.

составной пакет Пакет, содержащий как перемещаемые, так и абсолютные имена путей.

список исправлений Перечень исправлений, влияющих на используемый пакет. Этот список записан в установочном пакете в файле `pkginfo`.

стандарт криптографии открытого ключа #12 Стандарт, описывающий синтаксис для сохранения криптографических объектов на диск. В этом формате поддерживается хранилище ключей.

стандарт криптографии открытого ключа #7 Стандарт, описывающий общий синтаксис данных, к которым может быть применена криптография, например к цифровым подписям и цифровым конвертам. Подписанный пакет содержит встроенную подпись PKCS7.

сценарий действия над классом Файл, определяющий набор действий, которые будут произведены над группой объектов пакета.

сценарий установки Сценарий, позволяющий использовать настраиваемые процедуры установки пакета.

файл `compver` Способ указания обратной совместимости пакета.

файл `depend` Способ разрешения основных зависимостей пакета. См. также файл `compver`.

хранилище ключей пакета Хранилище сертификатов и ключей, которые можно запрашивать с помощью средств работы с пакетами.

центр сертификации Агентство, например Verisign, которое выдает сертификаты, используемые для подписывания пакетов.

цифровая подпись Кодированное сообщение, используемое для проверки целостности и безопасности пакета.

экземпляр пакета Вариация пакета, обусловленная комбинацией определений параметров `PKG`, `ARCH` и `VERSION` в файле `pkginfo`.

Указатель

хр

эк