PackageMaker How-to

This document is targeted towards Mac OS X Developers with a minimum background on Mac OS X and notions like file paths, bundles, Developer tools.

A few words regarding this document

This document was written with `PackageMaker` 1.x in mind.

`PackageMaker` is available in version 3.0.4 at the time of the writing of this note. All the pieces of information in this document still apply when you select the target to be Mac OS X 10.3 in `PackageMaker`.

What is PackageMaker?

`PackageMaker` is the installing solution provided by Apple with its developer tools.

Contrary to other installers' solution, `PackageMaker` does not create self-running installer (i.e. it does not build an application). Instead, `PackageMaker` is creating a package:

A package is a bundle (with a .pkg extension) which contains an archive of files to install and information on where to install them. It can also includes Read Me, License documents and scripts to run before or after the installation.

To install a package, all you have to do is to double-click on its icon in the `Finder` and the `Installer.app` application will be launched and will guide your through the necessary steps of the installation.

Packages are the solution used by Apple to install Mac OS X and to provide System or Applications upgrades via the Software Update solution.

In which cases may I use an installer?

The first question you must ask yourself is:

"*Do I need an installer to install my applications or documents?*"

For instance, if you're distributing an application without any other files to install (Frameworks, StartupItems, Drivers, etc.), you don't need to use an installer in 99.99% of cases.

The 0.01% left is made of unusual cases like for instance if you want your application to be installed in the `Applications` folder of the root home directory (but this is just an example). For the other 99.99%, just distribute your application inside a Disk Image and inform the user in the Read Me file he/she just needs to move the file into the `Applications` folder to install it. You can also use the background image of the Disk Image to provide instructions to the user.

If you're distributing a solution that needs to put a driver in `/SystemLlibrary/Extensions`, an application in `/Applications`, a framework in `/Library/Frameworks`, then you seriously need an installer.

What are the pro and cons of using PackageMaker as my installing solution?

Pro

- Apple uses installation packages in the format that PackageMaker can produce.
- It's free.
- This allows you to make remote installation using `Apple Remote Desktop` (1.1 or later).
- Packages can be installed via the `Terminal` (Mac OS X Server and Mac OS X client)
- It's a quite complete solution.

Cons

- If you're screwing something when building your installer this may lead to unwanted results on the end-user's machine - *do you hear the sound of the iTunes updater?* But you can ruin a system using other installers too (if you don't know how, just ask).
- There is no uninstall solution provided by Apple.
- It's not available on another platform.

There used to be two interesting articles on this topic on the Stepwise web site but this website has disappeared.

What are the other available installing solutions?

The following list is not exhaustive, it's just including the 3 major commercial solutions.

- **Installer VISE** by MindVision

Installer VISE has been one of the 2 legacy installing solutions on Mac OS. If you're a Shareware/Freeware developer, you may get qualified for a "free" license.

- **Install Anywhere** by ZeroG Software

The(?) Cross-Platform Installation solution. Available on Mac OS X. If you like Java, it might be your cup of te... coffee.

- **StuffIt InstallerMaker** by AladdinSys

The other legacy installing solution on Mac OS. I've never used it so I can't say a lot on it. If you're a Shareware/Freeware developer, you may get qualified for a "free" license.

Name	Developer	Version	Price	Supports Uninstall	Ease of use	Scriptable
Install Anywhere	ZeroG Software	5.0.7	$1,995	YES	?	?
Installer VISE	MindVision	8.2.1	Free to $1,500 a year	YES	?	AppleScript
PackageMaker	Apple	1.1a10	Free	NO	2/5	Command line
StuffIt InstallerMaker	AladdinSys	7.1.2	Free to $1,000+ a year	YES	?	?

Where can I find PackageMaker?

`PackageMaker` is installed with the Developer Tools. So if you don't have installed the Developer Tools, you won't find it.

If you have installed the Developer Tools, you can find `PackageMaker` inside the `/Developer/Applications` folder.

What version of PackageMaker should I use?

This may sound like a strange question but it's not that strange.

When Apple released Mac OS X 10.2 (Jaguar), a new `PackageMaker` application was released with the Jaguar Developer Tools.

This version of `PackageMaker` introduced new features but also a backward incompatibility.

The backward incompatibility is that packages built with the Jaguar version of `PackageMaker` can't be installed on Mac OS X system prior to 10.2 (i.e. 10.0.x and 10.1.x). I don't know why the "official" documentation is stated the contrary.

So if you want to allow users to install your package on Mac OS X 10.1 or prior, you will have to use the version of `PackageMaker` provided with the Mac OS X 10.1 Developer Tools.

This document will mainly deal with the 10.2 version of `PackageMaker`.

How do I create a package with PackageMaker?

Pre-requisite:

- `PackageMaker` needs to be installed on your machine (see Where I can find PackageMaker?)
- You need to have access to an admin account on your machine and probably to the root account.

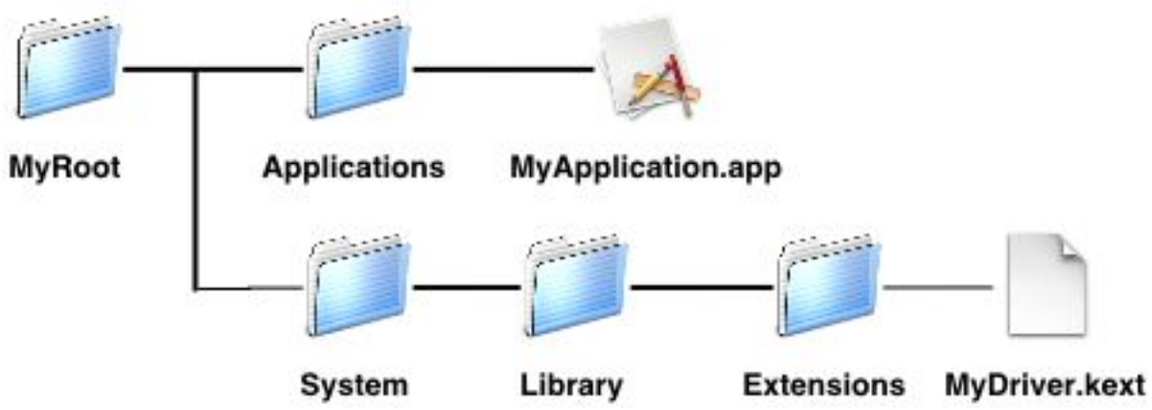
1st step:

This is the most important step. If you make mistake during this step, the consequences might be disastrous. So don't hesitate to check what you're doing.

To indicate to `PackageMaker` where the files of a package should be installed, you need to create a relative file hierarchy describing where each file should be installed. For instance, if a package needs to contain an application which has to be installed in the `Applications` folder, you need to create an `Applications` folder and put the application in this folder. If you need to install a driver in the `Extensions` folder, you need to create a `System` folder, then create a `Library` folder within the `System` folder, an `Extensions` folder within the `Library` folder and finally put your driver in the `Extensions` folder.

As stated, the file hierarchy is a relative one. This means that you first need to create a folder which will be the starting point of the file hierarchy. The final location of this starting point will be selected during the installation process. Usually, the final starting point is the root of a volume. If you need to install a driver or an application shared by every user, then the final starting point will be the root of a volume (the boot one or another).

To illustrate what you need to do, we will take the example of a package which needs to contain an Application and a driver used by this application. In this case, this is what the file hierarchy should look like:



The MyRoot folder can be located in your home folder for instance.

When this is done, you **ABSOLUTELY MUST** set the right owners and permissions of the files. If you don't set the appropriate permissions, you may end up creating a security hole or make the OS work incorrectly. A legend is saying that this may slow down the whole OS and this can only be solved later by using `Disk Utility.app`.

In our case, we need to set the following owners and permissions:

	Owner	Group	Permissions
`Applications`	`root`	`admin`	`rwxrwxr-x`
`System`	`root`	`admin`	`rwxrwxr-x`
`Library`	`root`	`admin`	`rwxrwxr-x`
`Extensions`	`root`	`admin`	`rwxrwxr-x`

As for our own files (bundles), we need to set all the files composing the bundles have the following attributes:

	Owner	Group	Permissions
`MyApplication.app`	`root`	`admin`	`rwxrwxr-x`
`MyDriver.kext`	`root`	`wheel`	`rwxr-xr-x`

You can of course set the permissions to be more restrictive but not less.

Once this is done, you then need to check that you don't have any useless files in this hierarchy.

One example of useless file is the `.DS_Store` file created by the `Finder`. While including this file in your package won't probably cause a security flaw on the system where the package will be installed, this may lead to some unwanted behavior.

For instance, if you have a `.DS_Store` file within the `Applications` folder, this file will overwrite the `.DS_Store` file in the `Applications` folder of the system where the package is installed. Since the `.DS_Store` file contains information on the way the `Finder` should display the content of a folder, when the user will display the content of the `/Applications` folder in the `Finder`, he/she may get a view with icons not located as they were before the package was installed.

> **Note:**
>
> You can easily remove all the `.DS_Store` in the relative file hierarchy using the Terminal and a command like:

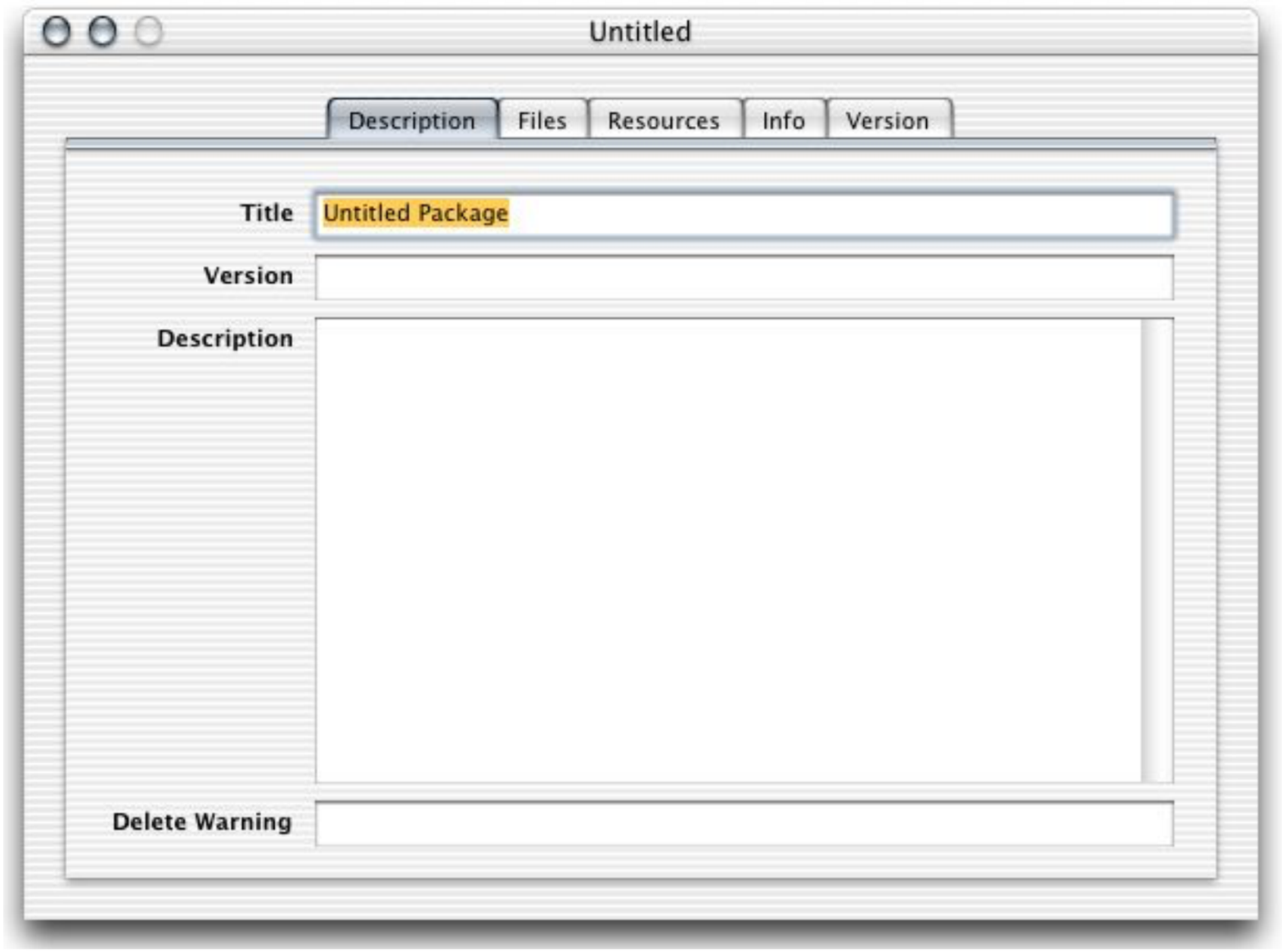
```
find $HOME/MyRoot/ -name ".DS_Store" -exec rm -f {} \;
```

2nd step:

Once, the file hierarchy is done, you can now use `PackageMaker` to create the package. To do this, you need to logout and login under the root account (maybe it can be done directly from an admin account, but I haven't tested that yet).

Depending on the version of `PackageMaker` you're using, the User Interface might be different.

You can now enter the information for your package:



Title: `MyApplication`

This title will be used during the Installation process for the Window title, the title of some steps, etc. If this package is to be a part of a metapackage, this will also be the name displayed in the Custom section of the metapackage installer.

Version: `1.0`

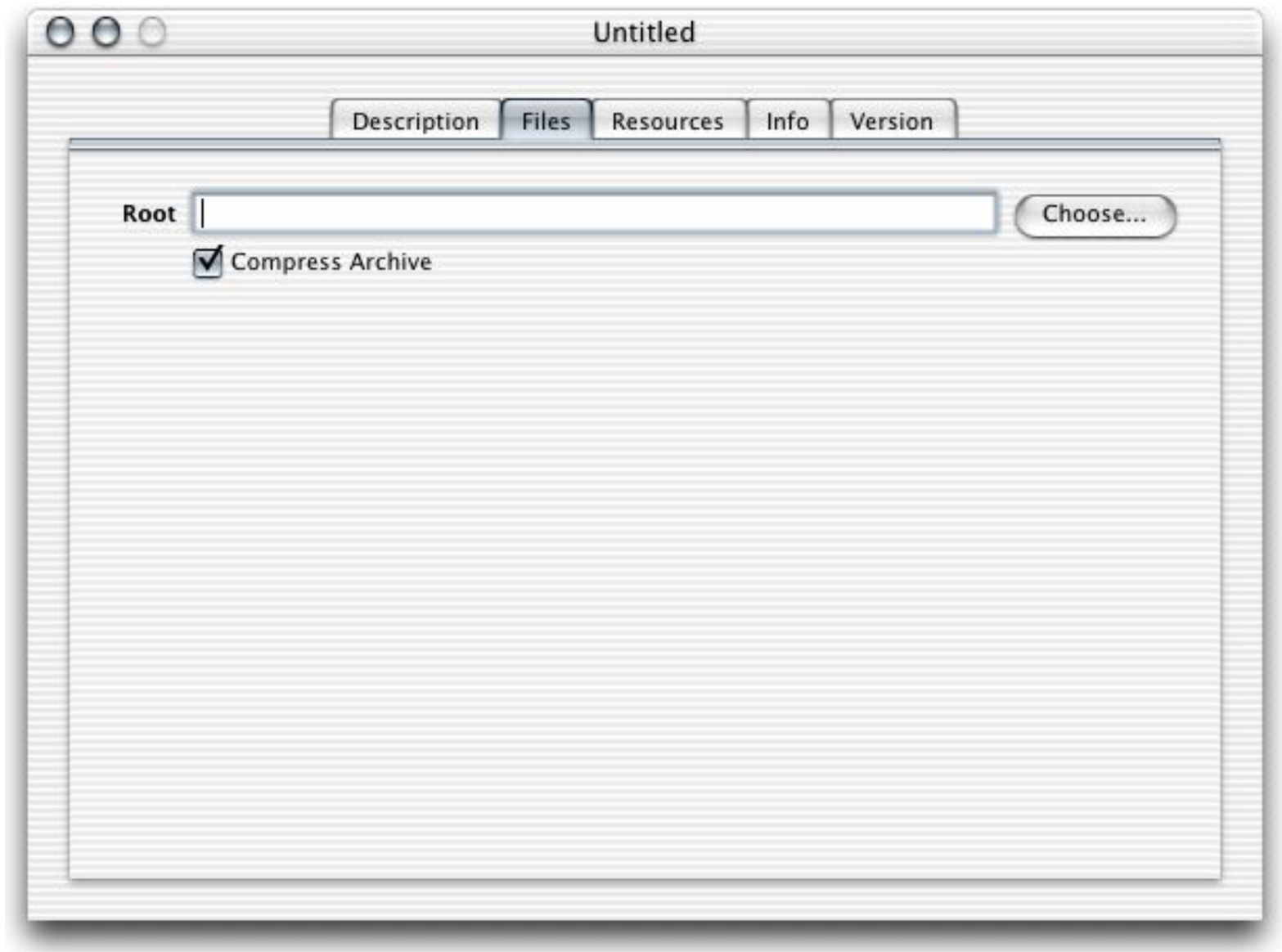
This piece of information is not used.

Description: `MyApplication is doing this and that.`

Contrary to what the `PackageMaker` documentation states, this description is used. This will be the description displayed for the package in the custom step of a metapackage installer.

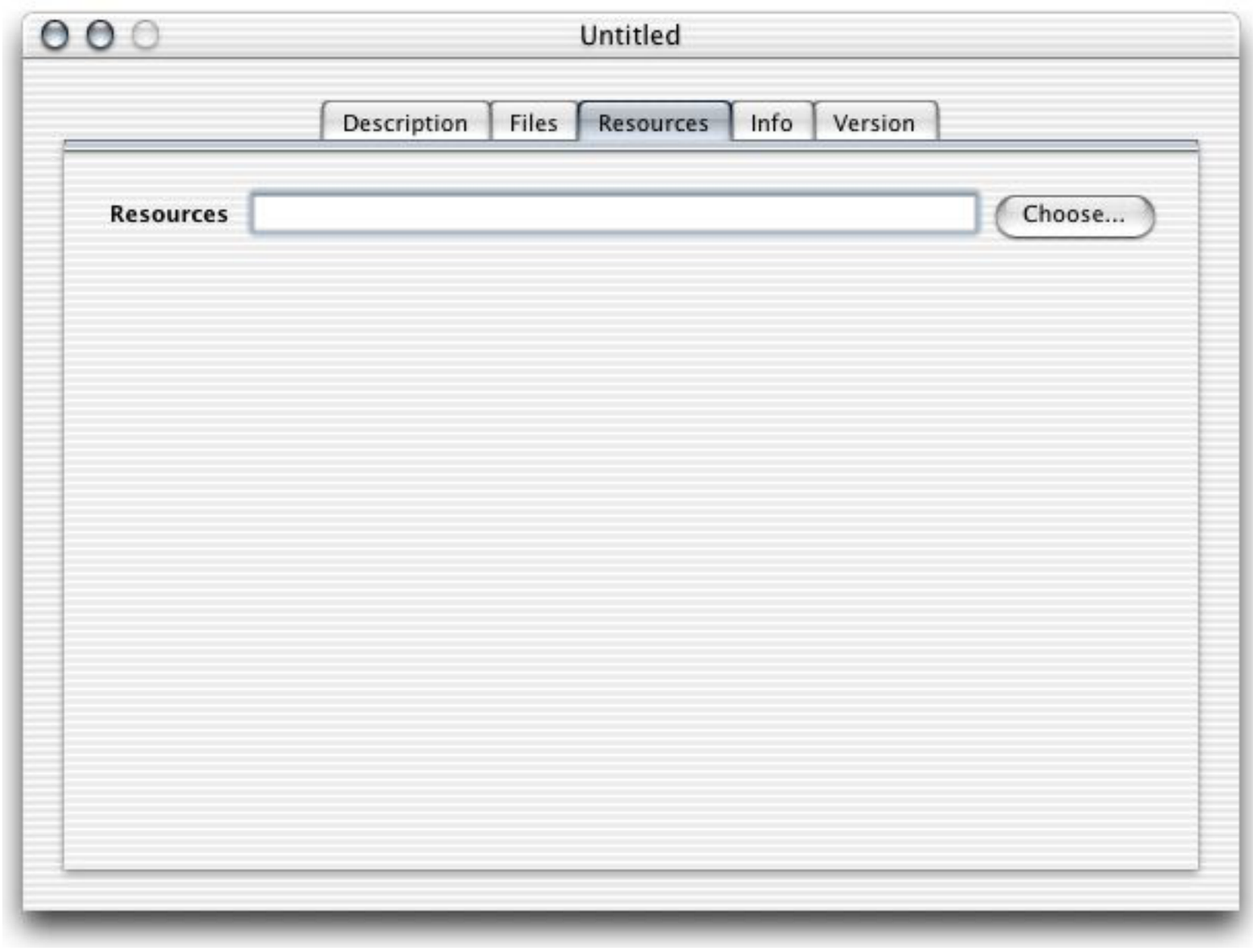
Delete Warning: `N/A`

This warning is not used currently since there is no Delete feature in the Apple's Installer solution.



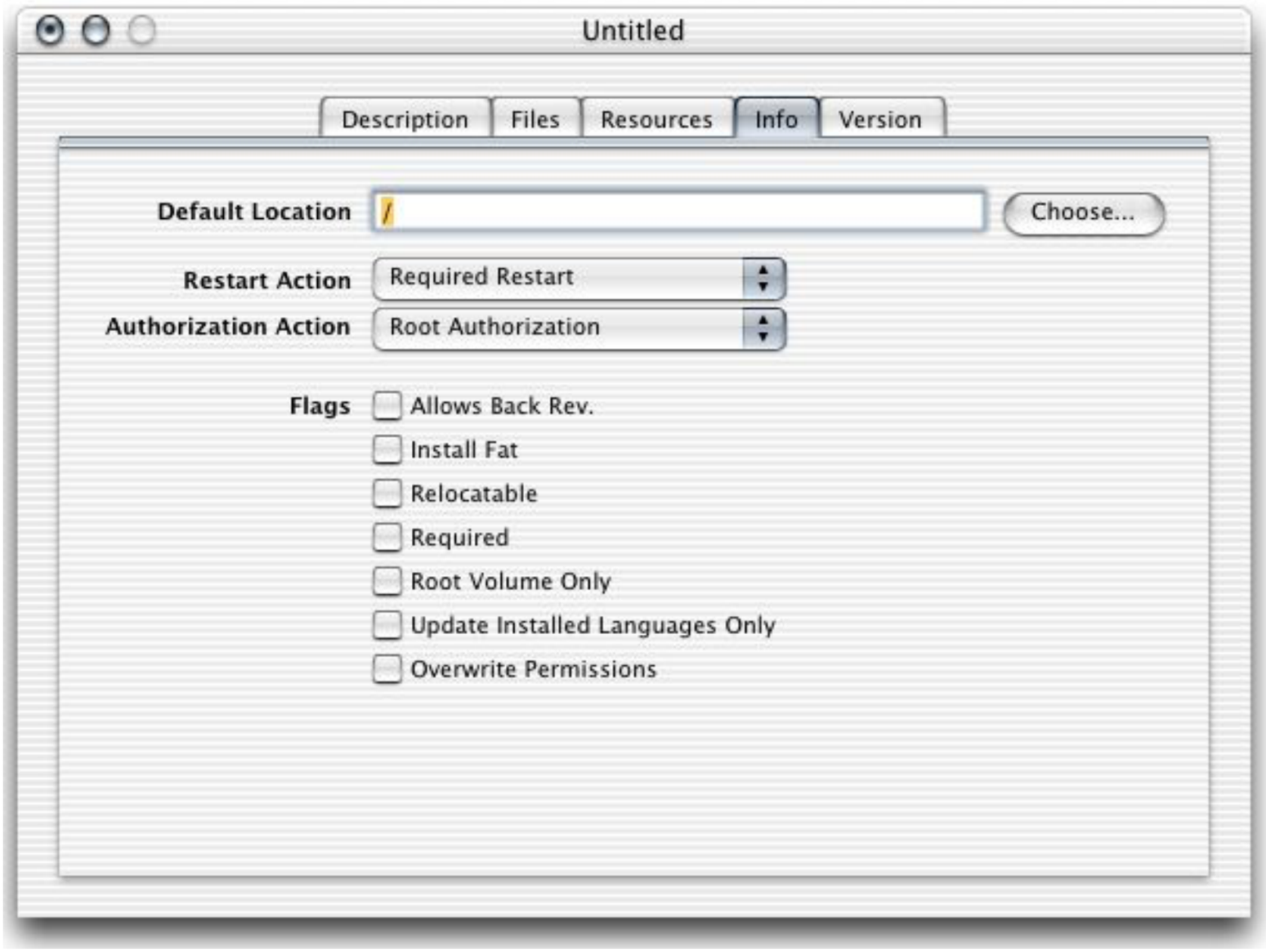
Root: `/Users/MyAccount/MyRoot`

This is the location of the Starting Point we built. It's usually a good idea to leave "Compress Archive" checked.



Resources:

We will discuss this part later. Shortly, this is the location of the Welcome, Read Me, License, background image, and scripts files.



Default location: `/`

This is from where the Starting Point should be added. If we were just installing an application, we could have just put MyApplication.app inside the MyRoot folder and then selected `/Applications` as the Default location.

Restart Action : `Required restart`

This depends on whether you need a restart so that your application, drivers, daemons, etc. are properly launched or not. In this case, let's just say that the Driver can't be launched properly outside of the startup process.

Authorization Action: `Root Authorization`

Since we're installing a Kernel Extensions and items inside `/System`, we need Root authorization.

Flags:

We will discuss these flags later.

Now, we can build our package using the "`Create Package`" in the "`File`" menu. Let's give it the following name: "MyPackage.pkg".

Before quitting `PackageMaker`, don't forget to save your package settings via the "`Save`" command in the "`File`" menu. This will avoid re-entering the same data later if the package information is changed or updated with new versions of the files to install.

To be able to find this package quickly, copy it into the `/Users/Shared` folder and logout. Login under your account and go to the `/Users/Shared` folder.

Double-click on the MyPackage.pkg file to test the package installation.

Additional steps

Now that you know how to build a basic package, we can see how to add additional information to the package.

As we said before, a package can contain Read Me and License documents. In fact, it can contains 3 types of documents:

Welcome:

This is the document that is displayed in the "Introduction" step (after the authentication step on Mac OS X 10.1 and 10.2 if any).

Tip: the real-estate for the Welcome document is limited to the size of the pane. It can't be scrolled.

Read Me:

This document is displayed after the Welcome document. It can be scrolled. The user can print the Read Me.

License:

This document is displayed after the Read Me document. It can be scrolled and to go to the next step, the user will have to click on a button to confirm he agrees to the terms of the license. The user can print the License.

Each of the 3 documents can be in one the following formats:

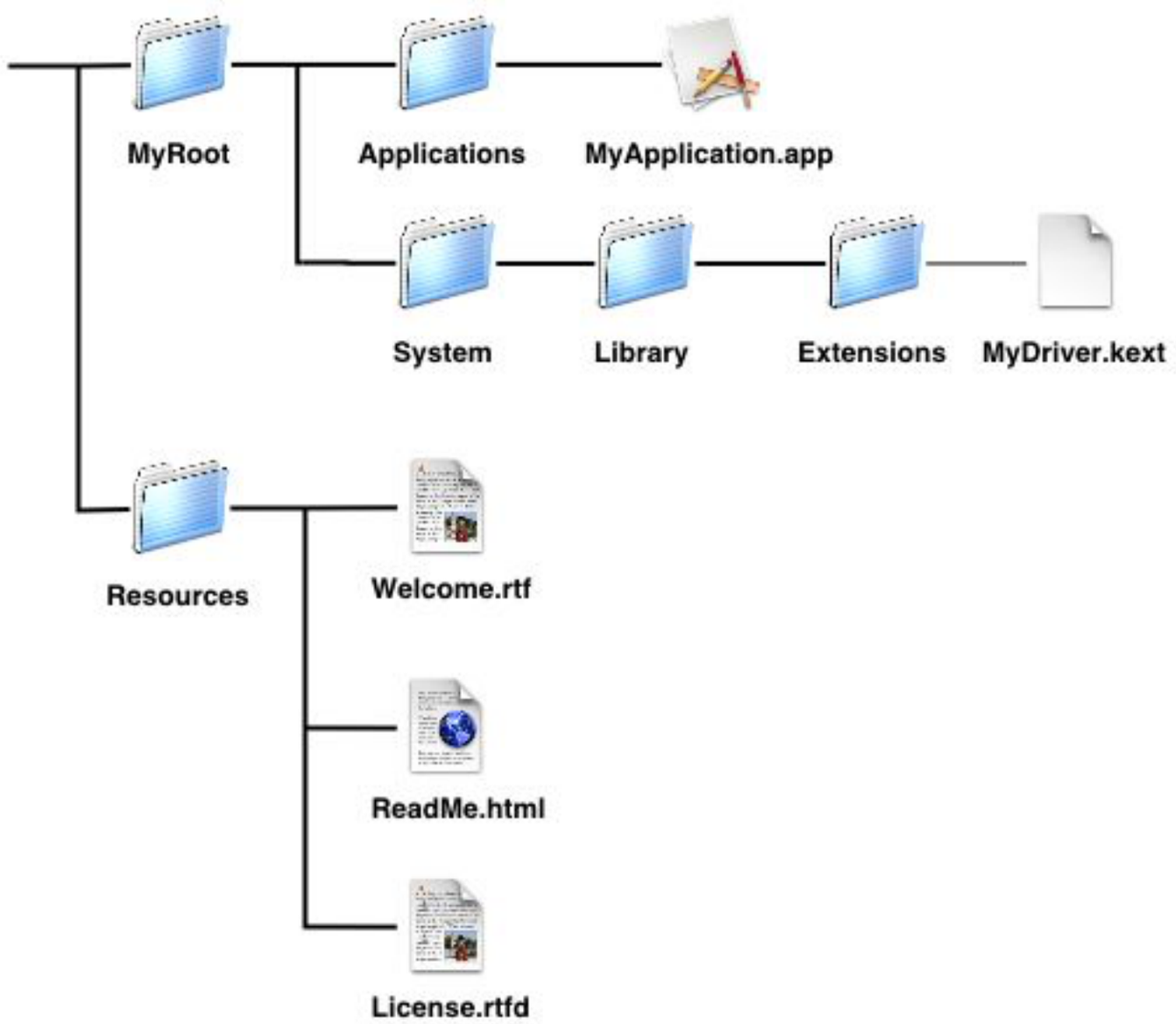
- RTF
- RTFD
- Text
- HTML

They should be named as follow:

Welcome: `Welcome`**`.rtf`**`, Welcome`**`.html`**`, Welcome`**`.rtfd`**`, Welcome`**`.txt`**.
Read Me: `ReadMe`**`.rtf`**`, ReadMe`**`.html`**`, ReadMe`**`.rtfd`**`, ReadMe`**`.txt`**.
License: `License`**`.rtf`**`, License`**`.html`**`, License`**`.rtfd`**`, License`**`.txt`**.

To add these documents to the package, you need to create a folder and put the documents inside it. It's a good idea to create this folder at the same level that your "MyRoot" folder.

This will look something like this:



In `PackageMaker`, you can now set the location for the `Resources` folder and when you're building your package, the documents will be added into the package and they will be displayed when the package is being installed.

What about scripts?

When installing your package, you may want or have to do some additional things.

For instance, if you're installing a Kernel Extension, you have to do the following additional operation: set the correct file owner and group. Starting with Mac OS X 10.2, Kernel Extension must be owned by the pair root/wheel. Yet even if you set the files to be owned by root/wheel in your Starting Point file hierarchy, they won't be

installed with root/wheel but with root/admin. This sucks, yes. One solution to fix this is to set the correct owner/group just after the package has been installed. This can be done via a script embedded in the package that will be launched by the Installer.

Another example: your package needs to remove files from a previous version before being installed. Again this can be done via a script embedded in the package that will be launched by the Installer.

Finally, another possible case where a script can be used is when you want to prevent the user from installing your package on a Mac OS X system prior to Mac OS 10.1.3. This can be done via a script embedded in the package that will be launched by the Installer.

There are 7 types of scripts than can be launched during the Installation Process:

InstallationCheck:

This script is launched at the beginning of the installation process (even before the Authentication step). It can be used to check that the package can be installed on this computer.

VolumeCheck:

This script is launched in the "Select Destination" step. It is used to determine on which volume the package can be installed.

preflight:

This script is launched just after the user clicked on the "Install" button.

preinstall / preupgrade:

This script is launched after the preflight script if there's one (in the case of a single package installation); otherwise just after the user clicked on the "Install" button.

As you see, there are 2 types of scripts here. The preinstall one is launched when the package has never been installed (from an `Installer.app` point of view), the preupgrade one is launched when the package has already been installed. To determine whether a package has already been installed or not, `Installer.app` is having a look at the content of the following directory: `/Library/Receipts`. If there's a file named PackageName.pkg within it, then the package has already been installed, otherwise it's the first install.

postinstall / postupgrade:

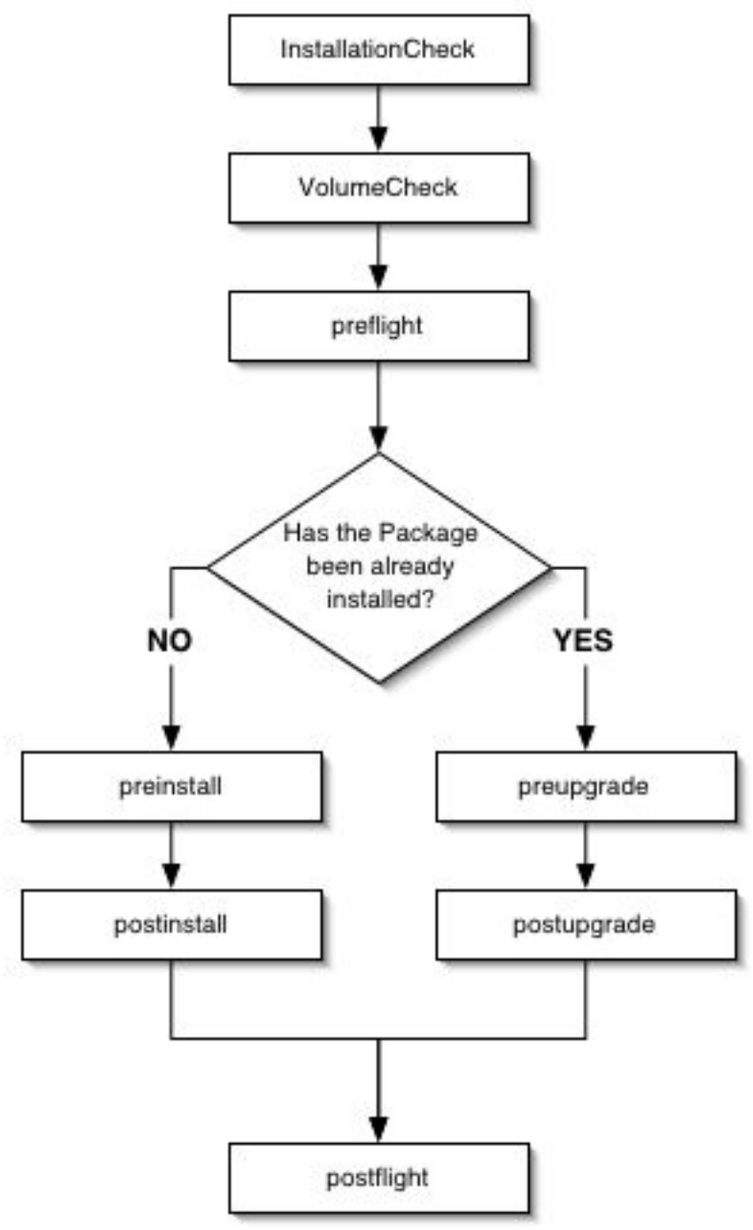
This script is launched after the files in the package have been installed. See (preinstall / preupgrade) to see which one is launched.

postflight:

This script is launched after the postinstall / postupgrade script or when the package has been installed.

Note: If you're building a package with the 10.1 version of `PackageMaker`, the names of the scripts are a bit different.

The scripts are launched in the following order:



These scripts can be written in different languages. The common ones are:

- Perl
- Shell

Some information is passed by the installer to these scripts when they are launched:

Language	Parameter	InstallationCheck	VolumeCheck	preflight	preinstall/upgrade	postinstall/upgrade	postflight

Shell	$0	Script path	Script path	Script path	Script path	Script path	Script path
	$1	Package path	Volume path [1]	Package path	Package path	Package path	Package path
	$2	Default location		Target location	Target location	Target location	Target location
	$3	Target volume		Target volume	Target volume	Target volume	Target volume
Perl	$0	Script path	Script path	Script path	Script path	Script path	Script path
	$ARGV[0]	Package path	Volume path [1]	Package path	Package path	Package path	Package path
	$ARGV[1]	Default location		Target location	Target location	Target location	Target location
	$ARGV[2]	Target volume		Target volume	Target volume	Target volume	Target volume
Returned value		**0:** Success **32:** Warning **48-63:** Warning: Detailed reason [2] **64:** Stop Install **112-127:** Stop Install: Detailed reason [3]	**0:** Success **32:** Failure: Unknow reason **48-63:** Failure: Detailed reason [4]	**0:** Success **!=0:**Failure	**0:** Success **!=0:**Failure	**0:** Success **!=0:**Failure	**0:** Success **!=0:**Failure

[1]
The `VolumeCheck` script is called for every volume.

[2]
The detailed reason is an index which is computed from the Returned value like this: `index = ReturnedValue - 32`
`Installer.app` then looks for a file named `InstallationCheck.strings` which associated a message to the index.
The `InstallationCheck.strings` will be used if you're building a localized package.

[3]
The detailed reason is an index which is computed from the Returned value like this: `index = ReturnedValue - 96`
`Installer.app` then looks for a file named `InstallationCheck.strings` which associated a message to the index.
The `InstallationCheck.strings` will be used if you're building a localized package.

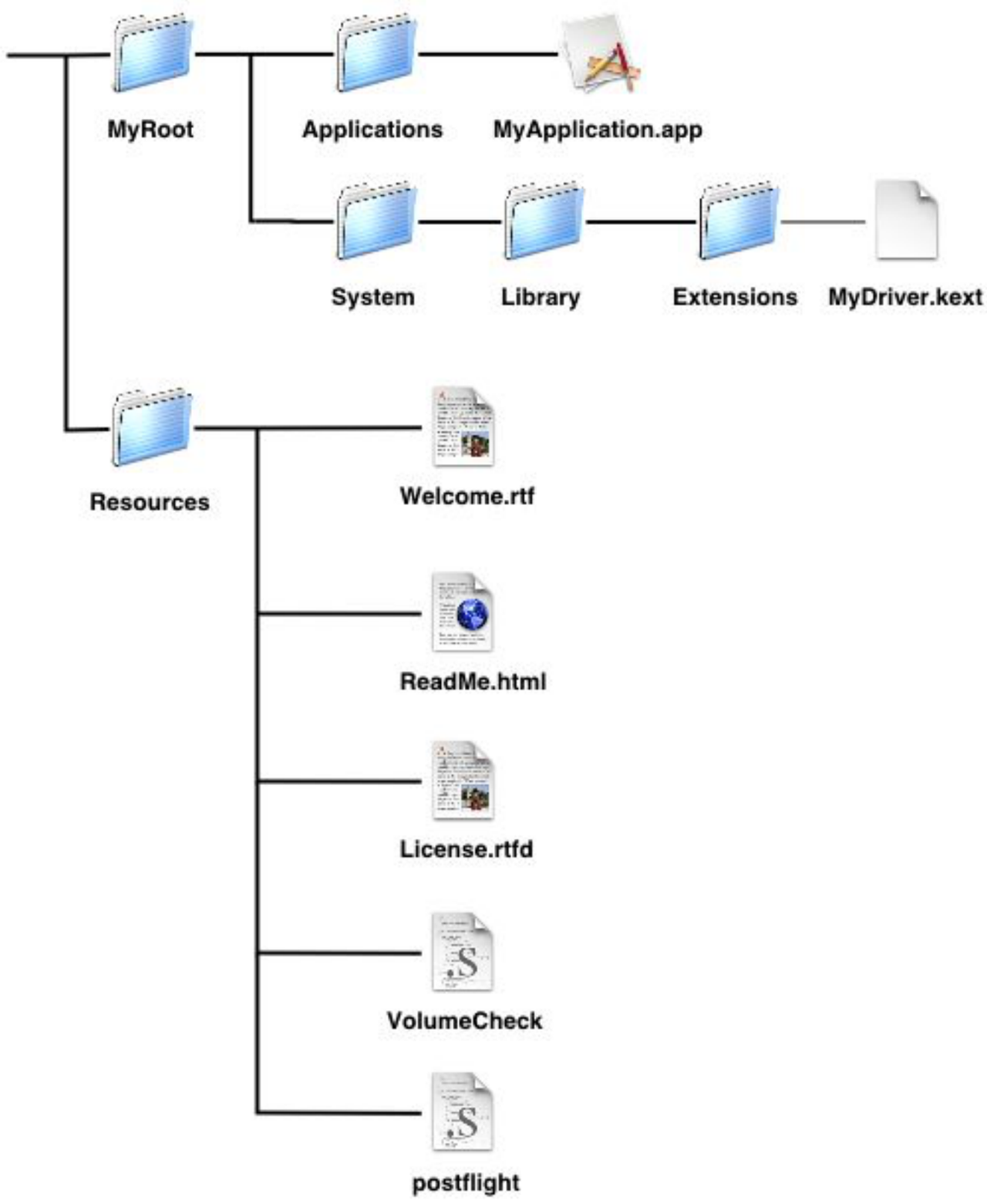
[4]
The detailed reason is an index which is computed from the Returned value like this: `index = ReturnedValue - 32`
`Installer.app` then looks for a file named `VolumeCheck.strings` which associated a message to the index.
The `VolumeCheck.strings` will be used if you're building a localized package.

Tip: The scripts must have an Unix line ending. If they don't, this may crash the installation.

The scripts are run under the following user account:

Authorization Action	InstallationCheck	VolumeCheck	preflight	preinstall/upgrade	postinstall/upgrade	postflight
No Authorization required	Login user	Login user	Login user	Login user	Login user	Login user
Admin Authorization	Login user	Login user	root	root	root	root
Root Authorization	Login user	Login user	root	root	root	root

Once you have written your scripts, add them into the `Resources` folder. In our virtual example, we're adding a `VolumeCheck` script to prevent the user from installing the application on a system prior to Mac OS X 10.2 and we're adding a `postFlight` script to fix the file permissions of our Driver.



You can then re-build your package: the scripts will be embedded in it and run during the Installation.

Finally, what about the flags?

Some of these flags are useful, some are useless.

Allows Back Rev. :

The installation allows to later install an earlier version of the package.

Install Fat:

The installation install all the binaries for multiple platforms.

Relocatable:

The user can choose the installation destination.

Required:

The package is a required component for the installation. This flag is only used when the package is a component of a metapackage.

Root Volume Only:

This option will only allow the root volume (/) to be the target volume of the installation. This will prevent the user to select another volume in the "Select Destination" step.

Update Installed Languages Only:

This is an interesting option that will prevent the installation of localized Resources in case the localization was not already installed.

For instance, let's say the user has not installed the German localization of your application. Your updater is including updated German resources but not all the resources. This means that if the user is installing the updater without this flag on, the German localization will only be partial and potentially crashy. With this flag on, the German files won't be installed and not cause any potential crash.

Overwrite Permissions:

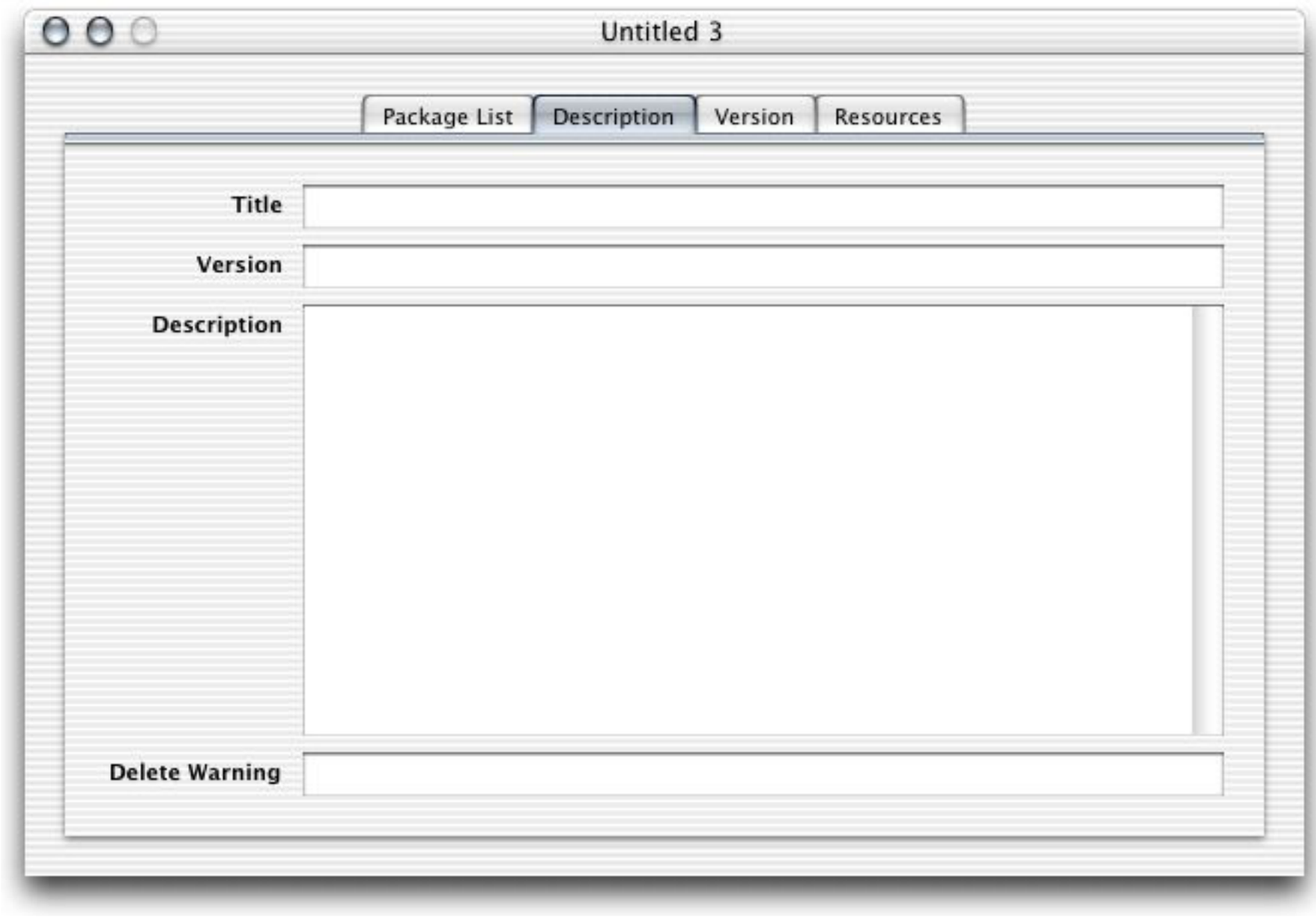
The permissions of directories in the installation overwrites the ones of the corresponding directories if they already exist in the destination volume.

How can I create a package with optional installs? (also How can I build a metapackage?)

For some installers, you may want to allow the user to select where he/she wants to install some optional files or not. For instance, the Mac OS X installation lets you decide whether you wan to install some localizations of the System or not.

This can be done via a metapackage. A metapackage is basically a package which does not contain any files but just points to a list of packages. A metapackage can contain Welcome, Read Me, and License documents. It can also contain `InstallationCheck` and `VolumeCheck` scripts. T he others scripts are not supported.

Building a metapackage with `PackageMaker` is easy as long as you know some information.



Title:

Description forthcoming

Version:

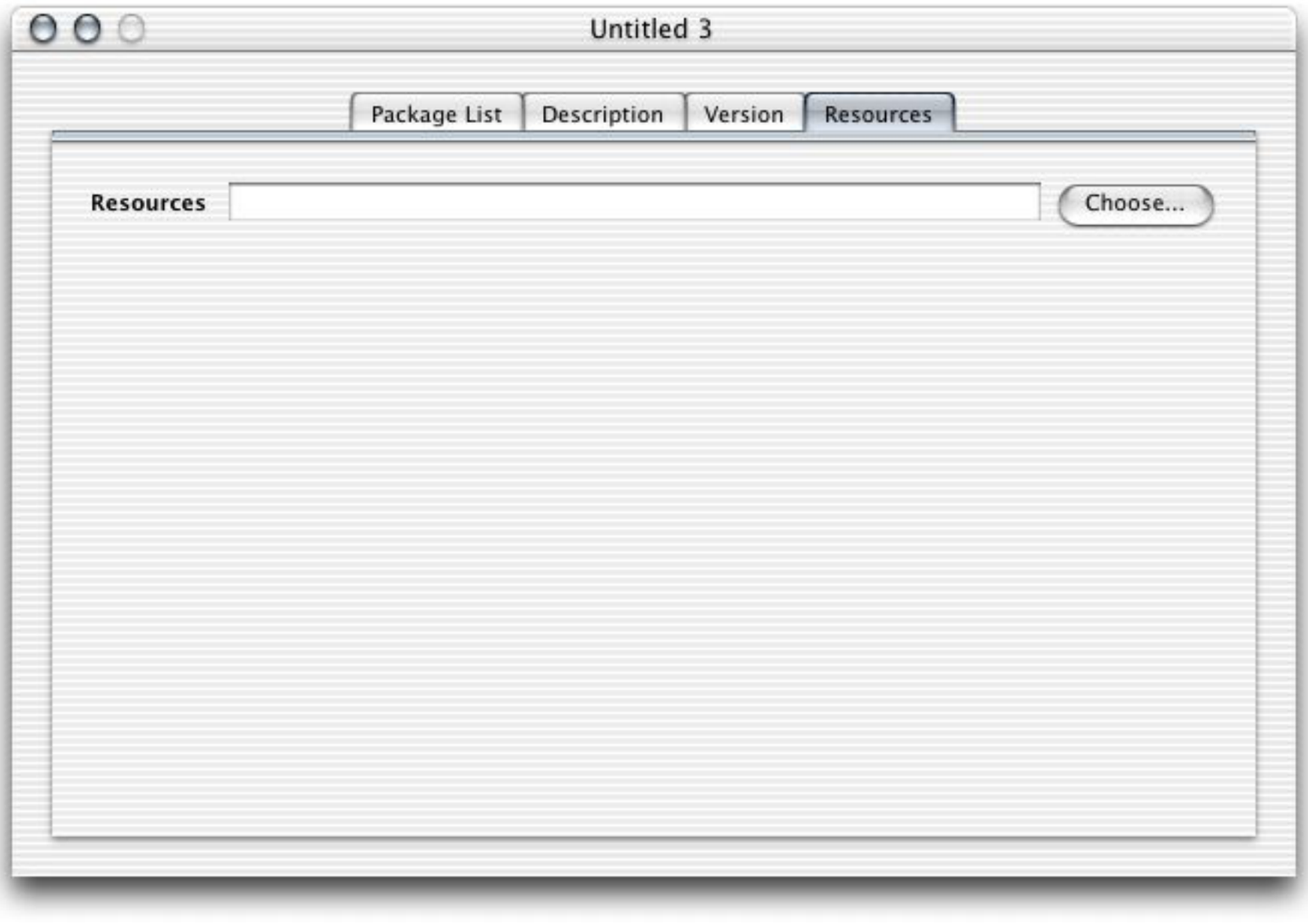
Description forthcoming

Description:

Description forthcoming

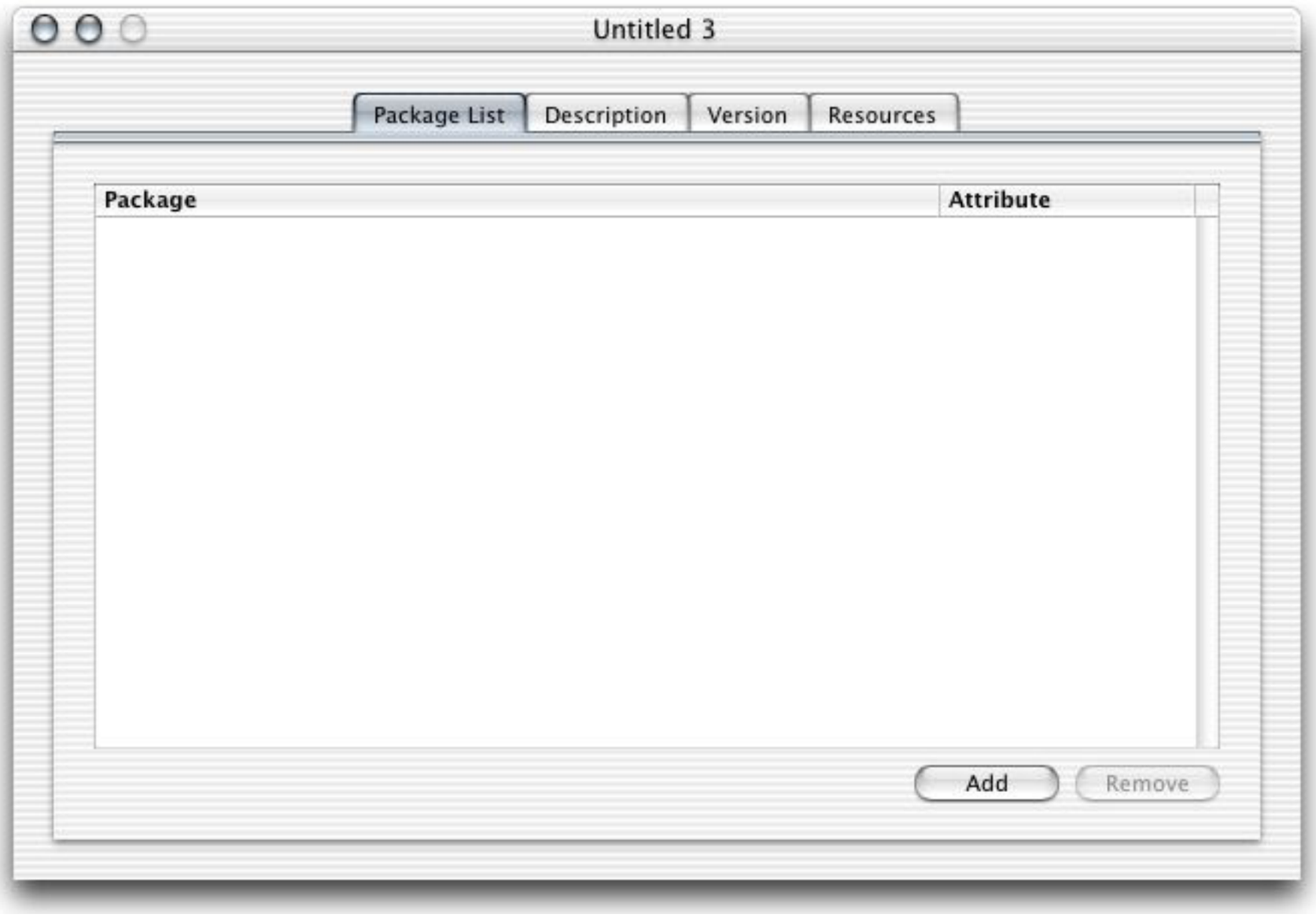
Delete Warning:

This warning is not used currently since there is no Delete feature in the Apple's Installer solution.



Resources:

Same thing as for packages. pre/postflight, pre/postinstall, pre/postupgrade scripts not supported.



Package List:

To be completed.

Attribute:

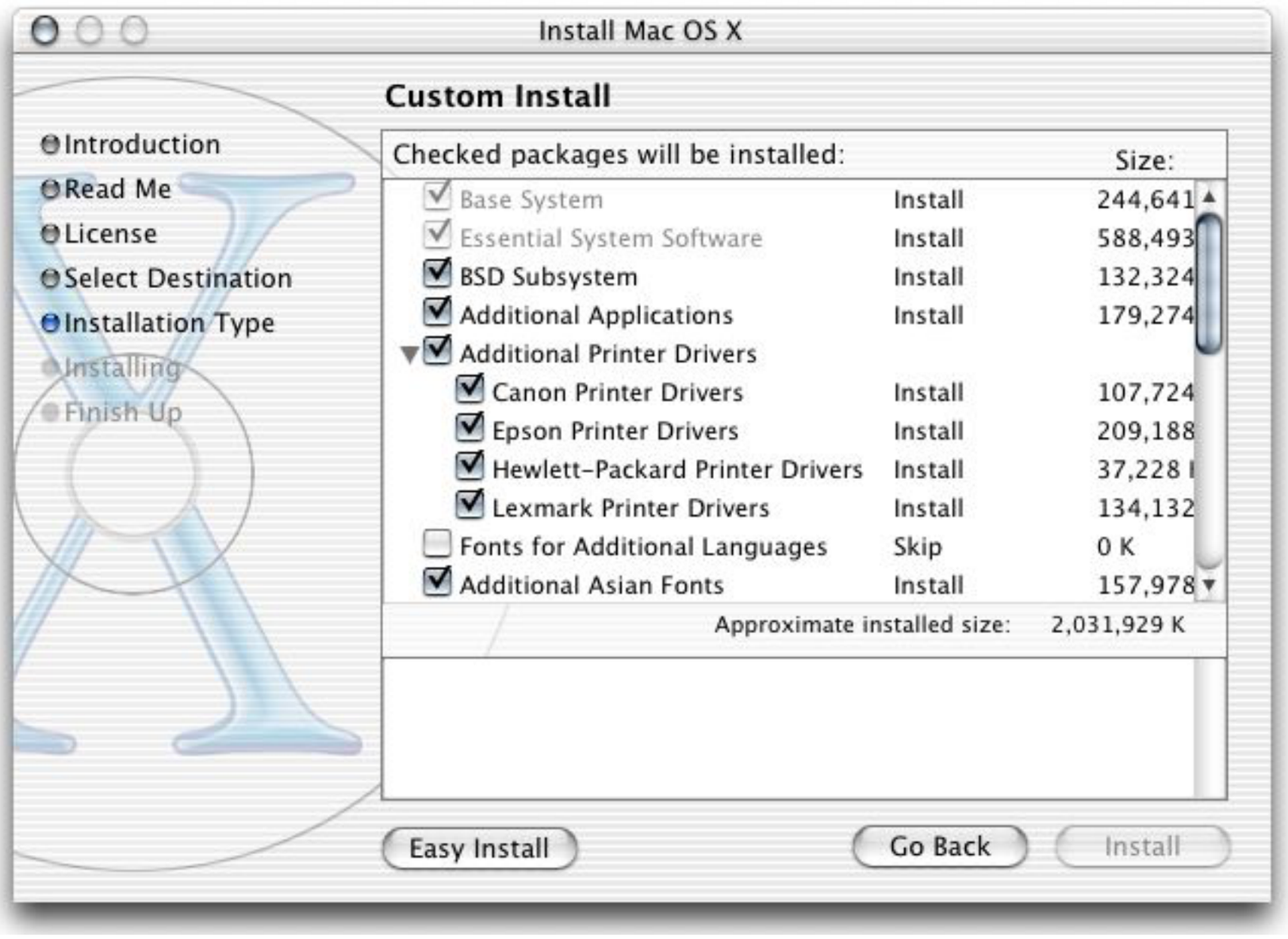
- **Required**: the package will always be installed/upgraded.
- **Selected**: the package will be installed unless the user switched to Custom mode during the installation process and unselect it.
- **Unselected**: the package will not be installed unless the user switched to Custom mode during the installation process and select it.

Tip: When building a metapackage with `PackageMaker` 10.1, you must know that the `Selected`, `Required`, `Unselected` option will not be taken into account. This has been fixed in the 10.2 version. So you can just specify whether a package is required or not. To do this, you have to check or not the `Required` flag when building the package with `PackageMaker` 10.1.

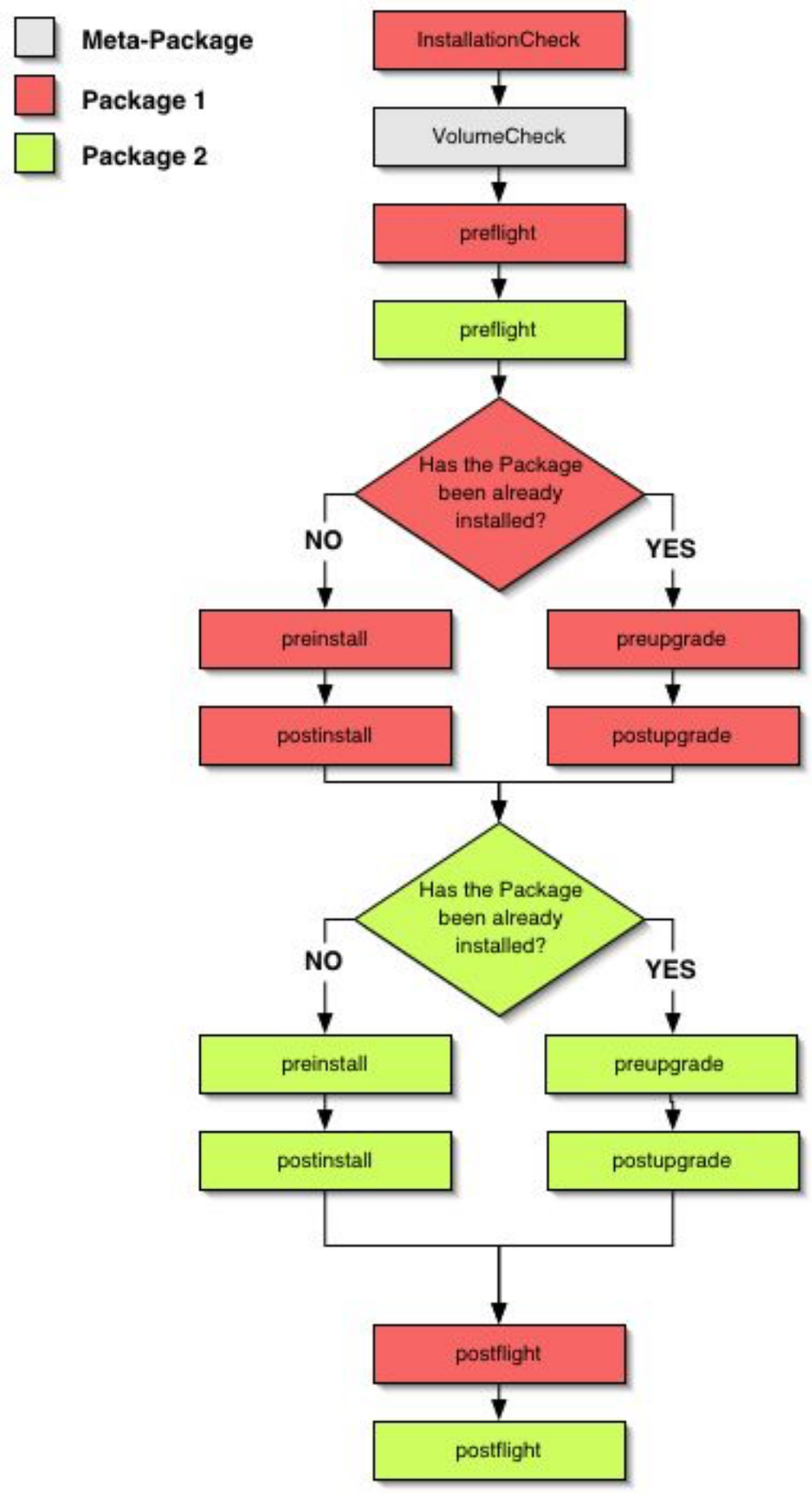
Package locations:

Description forthcoming

You can also select a metapackage to be a sub-package of a metapackage thus creating a nested hierarchy. This will result in something looking like this in the custom phase:



When a metapackage is being installed, scripts are run in almost the same sequence as with a package. The difference is with the preflight and postflight scripts.



Can I build an updater/upgrader with PackageMaker?

This is absolutely possible. Since there are pre/postupgrade scripts, it means that updating/upgrading is possible.

Basically, an updater/upgrader is just an installer which just contains the files that have been modified or added.

Some suggestions to use a package as an updater/upgrader:

- Use the `VolumeCheck` script to determine if the updater/upgrader can be applied to a volume.
- If you need to remove old files, use the `postupgrade` script.

How can I force the installation to be made on specific volumes?

Depending on your needs, there are 2 solutions:

- If you just need to restrict the installation to the startup volume, check the "Root Volume Only" flag of the packages.
- If you want to prevent the installation on volumes that don't meet some requirements, you can use the VolumeCheck strings.

  Note: There's a refresh bug in `Installer.app` which will display some volumes as potential candidates for the installation when they are not. Scrolling the list of volumes fix this.

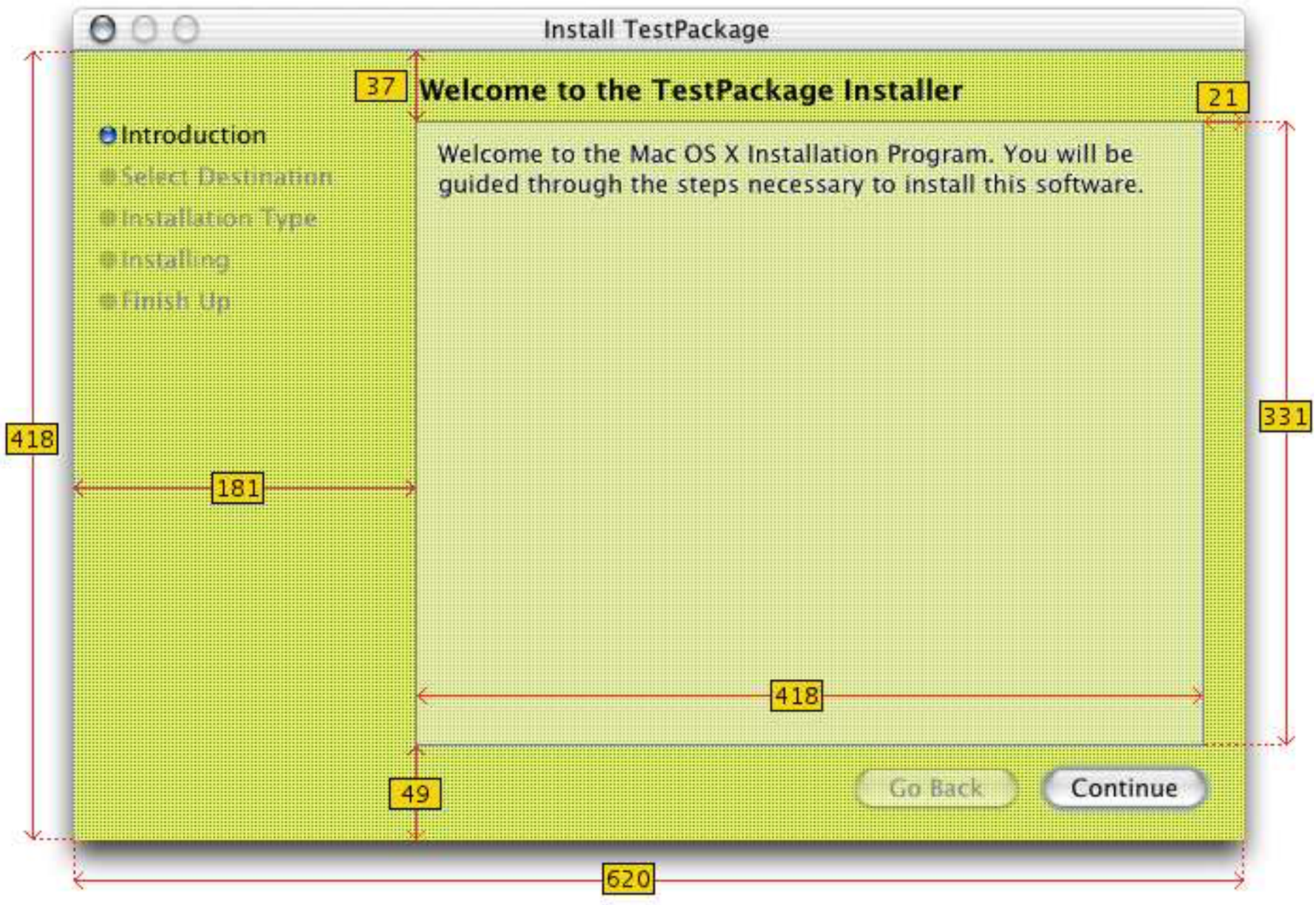
How can I add a custom background picture for my package or metapackage?

Beginning with Mac OS X 10.2, the background picture displayed in the `Installer.app` window can be customized by the package.

To set this background image, you just have to add a file to your package `Resources` folder called `background.suffix` where 'suffix' may be:

- jpg
- tif
- gif
- pict
- eps
- pdf

The ideal size for the picture is: **620 x 418**. If the size is different, the picture will be scaled.



Tip: If you're building a package using the 10.1 version of `PackageMaker`, you can use this background picture too. It won't be displayed when the installation is made on a Mac OS X version prior to 10.2, but it will for 10.2 and later.

How can I do a multiple localization installer?

Description forthcoming

How can I distribute my packages?

First thing to note is that since a package is a bundle, you can't distribute it "as is".

Some solutions that can be used:

- Archive the package using the `Finder` Create Archive of command.

  Description forthcoming

- Put the package in a disk image

  Description forthcoming

Caution

Avoid using Stuffit to archive/compress your package

Previous versions of Stuffit had issues with filenames bigger than 31 characters. If the name of the package is a bit long and you have some scripts in it, the name of one script might be bigger than 31 characters. So when it will get unarchived by Stuffit, the name will be screwed and the package won't install properly.

Yet, if all the names of the files in the package are less than 31 characters, you can use Stuffit.

Don't change the name of your package

It's really a good idea to avoid changing the name of your package once it has been built. Because on Mac OS X 10.1, `Installer.app` is using the name of the package to find the file archive and scripts. If the name is not the original one, the installation is going to fail.

Can I call an external application from an installer?

Yes, you can call an external application from an installer. You can do this with one of the installation scripts. For instance the following postflight script will launch `Preview.app`:

```
#!/bin/sh

# Launch Preview.app

open /Applications/Preview.app

exit 0
```

Does PackageMaker support files with Resource fork?

Yes, it does. If some files in your file archive contains both a Data Fork and a Resource Fork, the following alert will be displayed:

If you select the **Split Forks** option, files with Data and Resource Fork are going to be splitted in 2 files. The 2 forks will be re-united during the Installation process.

Is it possible to write a script to build a package?

Yes, depending on the version of the OS you're using, the tool is not the same:

- Mac OS X 10.2.x

  Use `package` which is available in `/usr/bin`. To get information on the required parameters, type in `Terminal.app`:

  ```
  $ package
  ```

- Mac OS X 10.3.x

Use `PackageMaker` which is available in `/Developer/Applications/Utilities/PackageMaker.app/Contents/MacOS`. To get information on the required parameters, type in `Terminal.app`:

```
$ /Developer/Applications/Utilities/PackageMaker.app/Contents/MacOS/PackageMaker -help
```

Man page on-line: man: packagemaker

And if you're a real thrill seeker, you may also want to build a package without these tools and just by using command line tools. Here is an example: packaging script

Is there any tool to inspect packages once they have been built?

If you want to know which files are archived in a package, you have 3 solutions:

- Installer.app:

  Since during the installation process, you can have access to the list of files in the package, one solution is to just launch the installation of the package.

  Just before clicking on the Install/Upgrade button, choose File > Show Files.

- The command line:

  A solution for the fans of `Terminal.app` is to use the command line tool `lsbom`:

  `lsbom MyPackage.pkg/Contents/Archive.bom`

- Pacifist:

  This is the best solution. Pacifist is a $20 shareware which allows you to inspect packages by displaying the hierarchy of files.

  With Pacifist, you can even install only some files just by drag and dropping them

  Pacifist is available here: CharlesSoft.

Is there any other tool to create packages and metapackages?

Yes, if you don't want to use `PackageMaker`, check alternative solutions such as Iceberg or Packages here: Packaging.

Copyright 2004-2014 Stéphane Sudre.