Preparing an Application for Installation by the Installer

The preferred way for a user to install a NEXTSTEP software package is to simply copy the package using the Workspace Manager.   This is how you should release your software, unless one of the following conditions applies:

· The software doesn't normally fit on a single floppy, but can be made to fit by using the Installer's compression feature.

· The software, even when compressed, cannot fit on a single floppy.

· The software is difficult for a user to install using the Workspace Manager, because files must be placed in several locations.

· The software requires the execution of installation programs before or after the software files are installed.

· The software is available only on a public FTP server.

If your software falls into any of these categories, you may configure and release your software as an *Installer package*, so that it can be easily installed and deinstalled using the Installer application (located in the **/NextAdmin** directory). An Installer package is represented as a file package (that is, a UNIX directory) with a ª.pkgº extensionÐfor example, **MyApp.pkg**.

This chapter describes the specification for Installer packages, and explains how to make your application conform to this specification. The procedure for preparing an Installer package is described first. The final two sections describe the contents and characteristics of single-volume Installer packages and multiple-volume Installer packages. You don't need to understand the complete specification in order to create an Installer package, since most of the work is done automatically by the **package** and **chunkPackage** utilities. However, the complete specification is provided in case you need it.

Creating an Installer Package

To create an Installer package, you use the **package** utilityÐa UNIX shell scriptÐlocated in the Installer file package:

```
/NextAdmin/Installer.app/package
```

The **package** utility has the following command-line syntax (bracketed arguments are optional):

package [-B] [-f] *root-dir* *info-file* [*tiff-file*] [-d *destination-dir*]

Arguments:

-B	Indicates that the **bigtar** program should be used to create the package, instead of regular **tar**. This is necessary only if the package contains files with pathnames greater than 100 characters. (Note that if this option is used, the **LongFileNames** field in the *MyApp***.info** file should have the value YES. See ªInstaller Package Specification for Single-Volume Packagesª for more information.
-f	Indicates that the *root-dir* argument is the thing to be packaged (the default behavior is to package

everything inside *root-dir*, but not *root-dir* itself).

root-dir The directory containing the files to be installed. The files should exist within this directory in the same locations as they are to be installed on the customer's machine.

info-file Contains information about the installed package and must have the ª.infoºº extension. The contents of the info file are described below in the section ªInstaller Package Specification.ºº

tiff-file Contains the icon for the software contained in this package. The file must have the extension ª.tiffº. More information about the TIFF file is also given in the section ªInstaller Package Specification.º

destination-dir
Specifies where to create the new package. If no destination is specified, the package is created in the current directory.

For example, if *MyApp***.root** in the current directory contains

```
MyApp.root/MyApp.app/MyApp
MyApp.root/MyApp.app/HelpFile
```

then the following command will generate an Installer package **./Packages/MyApp.pkg**:

```
package MyApp.root MyApp.info MyApp.tiff -d ./Packages
```

If a package is too large to fit on a single volume (for example, on a single floppy disk), it can be broken up into multiple smaller package folders, each large enough to fit on a volume. The procedure for breaking up an Installer package is described in the following section.

Breaking Up a Large Installer Package

When a package folder is too large to fit on a single floppy disk, it can be broken up into *n* smaller package folders, each of which fits onto a floppy disk. The smaller packages, when they are on the floppy disks, should each have the original package name *MyApp***.pkg**. Hence floppy #1 will contain a folder *MyApp***.pkg**, as will floppy #2, ..., and floppy #*n*. The package folders must appear in the top-level directory of the floppies.

The package archive file *MyApp***.tar.Z** must be ªchunkedºº into *n* piecesÐ*MyApp***.tar.Z.1**, *MyApp***.tar.Z.2**, ..., *MyApp***.tar.Z.***n*Ðthat will individually fit onto the floppies (along with the other files in the package folder). The **chunkPackage** utility (located in the Installer file package) can be used to break up the Installer package that was generated with the **package** script:

```
/NextAdmin/Installer.app/chunkPackage
```

The **chunkPackage** utility has the following command line syntax (the bracketed argument is optional):

chunkPackage *package* *volume-size* [-p *pad-size*] [-d *destination-dir*]

Arguments:

package	The Installer package folder generated by the **package** script.
volume-size	The capacity in kilobytes of the volumes on which the smaller packages will be stored (note that the actual capacity of the formatted disk, rather than its nominal capacity, must be specified).
pad-size	Specifies an amount of space (in kilobytes) to reserve as ªpaddingº on the first disk. This space could be reserved for a README file, for example.
destination-dir	Specifies where to create the smaller packages. If no destination is specified, the chunks directory is created in the current directory.

Given a package *MyApp***.pkg**, the **chunkPackage** utility creates the directory *MyApp***.chunks** with the subdirectories *MyApp***.1**, *MyApp***.2**, ..., *MyApp***.***n* and stores a smaller sub-package in each of these subdirectories. For example, the following command breaks up the *MyApp***.pkg** into a number of smaller packages that will fit on 2.8-megabyte floppies and places them in **./***MyApp***.chunks**:

```
chunkPackage MyApp.pkg 2400 -d ./MyApp.chunks
```

The **./***MyApp***.chunks** directory has the following structure after running this script:

```
./MyApp.chunks
    /MyApp.1/MyApp.pkg
    /MyApp.2/MyApp.pkg
    ...
    /MyApp.n/MyApp.pkg
```

You'll need to copy the *MyApp***.pkg** folder in *MyApp***.1** onto floppy #1, the *MyApp***.pkg** folder in *MyApp***.2** onto floppy #2, and so on.   Be careful not to copy the   *MyApp***.1** and *MyApp***.2** folders themselves.

Floppy #1 must contain the archive chunk *MyApp***.tar.Z.1**, floppy #2 must contain *MyApp***.tar.Z.2**, and so on, up to *n*.   The concatenation of the files *MyApp***.tar.Z.1**, ..., *MyApp***.tar.Z.***n* should be identical to the original archive *MyApp***.tar.Z**.   As a test, running the command:

```
cat MyApp.tar.Z.1  ...  MyApp.tar.Z.n  |  sum
```

must provide the same checksum as the command:

```
sum MyApp.tar.Z
```

Each floppy-sized package should contain the ª.infoº, ª.bomº, ª.sizesº, and ª.tiffº files for the package.

The last floppy-sized package (that is, the *MyApp***.pkg** folder on floppy #*n*) must contain a file **.last**, that is, *MyApp***.pkg/.last**.   This file is used as an ªendmarkerº by the Installer to know when it has seen the last floppy in a multiple-volume package.   The **.last** file can have size zero.

Adding Executable Scripts to an Installer Package

Installer lets you add up to four programs (typically shell scripts) to an Installer package.   Two of these programs, the **pre_install** and **post_install** programs, will be executed before and after the Install and Expand operations if the programs exist inside the Installer package.   Similarly the **pre_delete** and **post_delete** programs, if they exist, will be run before and after Delete and Compress operations. You can write any or all of these four programs in order to accomplish a variety of preprocessing and postprocessing tasks.

For example, assume that you have created an Installer package named **MyApp.pkg**. To add a program that will be executed before installation, write the program and name it **MyApp.pkg/MyApp.pre_install**. Similarly, programs to be executed after installation, before deletion, and after deletion would be named **MyApp.pkg/MyApp.post_install**, **MyApp.pkg/MyApp.pre_delete**, and **MyApp.pkg/MyApp.post_delete**, respectively. The programs must be readable and executableÐthat is, their UNIX modes must include the octal bits 555.

Each of the four programs is invoked by Installer with two arguments:

· The first argument is the absolute pathname of the Installer package when the program is invokedÐfor example, **/MyAppDisk#1/MyApp.pkg**.

· The second argument is the absolute pathname of the location where the contents of the package will be installedÐfor example, **/LocalApps** or **/Net/Server/User/Apps**.

Installer checks the exit status of a program when it terminates. A program that exits with status 0 is considered to have successfully completed, and the Installer will continue the current operation. If a program exits with non-zero status, Installer aborts the current operation.

Each of these programs should write at least ªOK<CR>º to standard output when it successfully completes, or write ªFAILED<CR>º (perhaps with an explanation) if it fails to complete. This output will appear in the Installer Log view, as in the following example (where the **MyApp.pre_install** program succeeds and the **MyApp.post_install** program fails; their output is shown in bold):

```
Installing MyApp.pkg into /LocalApps ...
    Running installation program ... OK.
    Installing /LocalApps/MyApp.app/MyApp ... OK.
    Running installation program ... FAILED (not enough disk space).
... errors.
```

Creating a Remote Installer Package

Installer supports *remote packages* whose contents (specifically, the compressed tar archive file) are stored on a public FTP server. A remote package contains the usual files that make up an Installer package (including scripts and localized files) except for the compressed tar archive (the ª.tar.Zº file), which is not contained in the remote packageÐinstead, it's stored on an FTP server. Remote packages are small since they don't contain the archive file, and hence can be mailed or posted electronically without major resource consumption. A user who receives a remote package via electronic mail or bulletin board can install the package in the usual way by opening the package icon. When the installation is performed, an anonymous FTP connection is created to the FTP server and the archive file is retrieved and installed on the user's machine.

The file **MyApp.pkg/MyApp.info** must have an **FtpSite** field whose value is the FTP location of the archive file. For example, if the archive file for the package is on **sonata.cc.purdue.edu** in the location **/pub/next/2.0-release/binaries/MyApp.tar.Z**, then the file **MyApp.info** must contain the following field:

```
FtpSite  sonata.cc.purdue.edu:/pub/next/2.0-release/binaries/MyApp.tar.Z
```

The host computer name and the filesystem path are separated by a colon. There can be only one **FtpSite** field per remote package; therefore a given remote package can be used to access only one remote location.

Localizing an Installer Package

Installer supports localization of Installer packages by letting you put localized ª.infoº files, ª.tiffº files, and installation scripts in language-specific subdirectories within the package. When a user opens such a package, the information that is displayed in the Installer Package window is read from the ª.infoº file and ª.tiffº file that are chosen in the standard way based on the user's language preferences. Also, scripts in localized packages are chosen for execution based on the user's current language settings.

You can localize an Installer package **MyApp.pkg** to French and German, for example, by adding the files **MyApp.pkg/French.lproj/MyApp.info** and **MyApp.pkg/German.lproj/MyApp.info**. Edit these files so that their **Title**, **Version**, and **Description** fields are translated into the appropriate language. Installer uses the NXBundle searching mechanism to find **MyApp.info** within **MyApp.pkg**, so if the file is not found in the ª.lprojº directory for the user's primary language then the ª.lprojº directories for secondary languages are searched in order; finally the **MyApp.pkg** directory itself is searched.

Similarly, to use localized icons for the package icon, you could localize the file **MyApp.tiff**. You can also provide localized versions of the **pre_install**, **post_install**, **pre_delete**, and **post_delete** programs by placing them in the appropriate ª.lprojº directory.

Installer Package Specification for Single-Volume Packages

This section defines the contents of a single-volume Installer packageÐthat is, a package folder that fits on one disk (floppy, optical, CD-ROM) volume. The package folder should have permissions 755 or 555, as described in the **chmod**(1) UNIX man page.

A single-volume package named *MyApp***.pkg** *must* contain the following files:

*MyApp***.tar.Z**
*MyApp***.bom**
*MyApp***.info**
*MyApp***.sizes**

*MyApp***.pkg** may also contain the following optional file (a 48-by-48 icon that will be displayed in the Info view of the Installer's Package window):

*MyApp***.tiff**

The contents of these files are specified below. Each file should have permissions 644 (-rw-r--r--) or 444 (-r--r--

r--).

MyApp.tar.Z

This file is a compressed tar archive (see the **compress**(1) and **tar**(1) UNIX man pages). The contents of the archive are the files that are to be installed on the customer's machine. The archive should be created by using **tar** and **compress** on a directory that contains these files. For example, if the files to be installed by the *MyApp* package are contained on the developer's machine in a directory *MyApp***.root**, then the archive can be created in the current directory by the command:

```
(cd MyApp.root; tar cf - . ) | compress -f -c > MyApp.tar.Z
```

The contents of the compressed archive can be listed by the following command:

```
zcat MyApp.tar.Z | tar tf -
```

which prints something like:

```
./bar
./baz
<etc.>
```

Each of the listed files and directories is prefixed by ª./ºº, which will allow them to be installed relative to a user-specified directory. If a file must be installed in a particular fixed location, then the files can be explicitly named in the command that creates the archive. For example, suppose that the file **MyExecutable** must go into **/usr/bin**. In this case, the archive can be created by a command that explicitly archives **/usr/bin/MyExecutable**:

```
(cd MyApp.root; tar cf - . /usr/bin/MyExecutable) | compress -f -c > MyApp.tar.Z
```

which results in an archive whose contents would be listed as:

```
./bar
./baz
```

```
   <etc.>
   /usr/bin/MyExecutable
```

You should make sure any directories that will be installed as part of the package are owner-writable. Otherwise, attempts to install files into those directories will fail. The recommended protection mode for installed directories is 755 (drwxr-xr-x).

MyApp.bom

This file is the ªbill of materialsº for the package. The file contains a listing of the contents of the archive, one file per line. Each line has several fields, delimited by whitespace and terminated with a newline. For example, here is the line for the file **./bar** that would appear in *MyApp***.bom**:

```
   ./bar rwxr-xr-x  0/0  65536 Jun 21 22:38 1990
```

The first field is the name of the file, exactly as it appears in the contents of the archive when the archive is listed as described above. The second field lists the file's permissions in a form similar to that used by the UNIX command **ls**(1). The third field gives the file's uid/gid. The fourth field is the size of the file in bytes. The rest of the line is the file's UNIX modified-time. When the package is installed, the file is converted to an uneditable format.

Note: The names in the ª.bomº file must be names of ordinary files, not names of directories. The output of **tar**(1) with the **-tv** option includes directories, so you must be filter out these directories if you use the output of **tar** to create the ª.bomº files (you can easily determine which of the names listed by **tar** are directories, by looking for those names that end with ª/º). We recommend that you use the **package** utility described earlier to generate the ª.bomº file, since **package** excludes directories automatically.

MyApp.info

This file contains information about the package that is not automatically derived. This file must be manually edited and maintained by the developer.

The ª.infoº file is a list of (*name value*) pairs, one pair per line. Blank lines or lines consisting entirely of whitespace are ignored. Lines whose first non-whitespace character is # are treated as comments and ignored. The *value* part of a line can have a maximum of 1023 characters; subsequent characters are discarded.

The format of each line is:

[*optional-whitespace*] *name* *whitespace* *value*

The *name* must be a single token, not containing whitespace. The *value* is the string that begins with the first non-whitespace character after the name and ends with the last character on the line. The terminating newline is not part of the *value*.

Here is an example *MyApp***.info** file:

```
# The info file for the MyApp package.

# These are the fields that will be displayed in the Info view.
Title                  The MyApp application
Version                5.2  August 15, 1990
Description            The MyApp application helps you do everything from brush your teeth to
wax your car.

# These fields determine where the installed package will go.
DefaultLocation        ~/Apps
Relocatable            YES

# This field is a pattern that matches the floppy disk labels
DiskName               MyApp #%d

# This field states whether the package is a NEXTSTEP application
Application            YES

# User is prompted with this field before deleting the package.
DeleteWarning          Deleting this package will cause the MyApp application to stop working.
```

```
# end of MyApp.info
```

The following fields are required:

Title The value of this field will be displayed at the top of the Info view in the Installer's Package window when the package is opened. It should be short, less than 50 characters.

Version The value of this field will be displayed in the Version field of the Info view. The value should be something like ª23ºº or ª56.3aºº or ªv7.0, Jun 24 1991 14:35ºº.

Description

The value of this field will be displayed in the Description field of the Info view. The description should be a brief message that will tell the user about the contents of the package. The value of this field must be no more than 1023 characters in length.

DefaultLocation

The value of this field determines where the contents of the package will be installed. The value should be a UNIX pathname, preferably **~/Apps** (or possibly **/LocalApps**). If the value of the **Relocatable** field is NO or is missing (in which case it defaults to NO), then the value of the **DefaultLocation** field will be used without modification as the root directory relative to which the contents of the package are installed. On the other hand, if the value of the **Relocatable** field is YES, then the value of the **DefaultLocation** field is used as the initial directory for the InstallIn panel, from which the user chooses a destination root directory for the installation.

Restriction: The value of the **DefaultLocation** field must not contain whitespace (such as blanks and tabs) or the following shell metacharacters:

```
! $ ^ & * ( ) { } [ ] \ | ; < > ? ' " `
```

DiskName

The Installer uses the value of this field to compute the disk name when prompting the user to insert a floppy or optical disk. When the entire package fits on a single disk (that is, when it's a single-volume package), the value of this field should be exactly equal to the disk label. For example, if the *MyApp* package fits onto a single floppy and the floppy disk is mounted with the name ªMyApp

Softwareºº when the disk is inserted, then *MyApp***.info** must define this field as:

```
DiskName    MyApp Software
```

Any whitespace that appears at the end of the line is considered part of the disk name, so the developer should be careful to place a newline immediately after the last character in the disk name.

When the package is to be distributed on more than one floppy, the value of this field should contain a single instance of the substring **%d**. This value will be used as a pattern, with the substring **%d** successively replaced by the numbers 1, 2, 3, ..., and so on. For example, if the *MyApp* package is distributed on floppies labeled ª*MyApp* Software #1º, ª*MyApp* Software #2º, ..., ª*MyApp* Software #*n*º, then *MyApp***.info** must define this field as:

```
DiskName    MyApp Software #%d
```

There should be no other occurrences of the **%** character in this field.

Restriction: The value of the **DiskName** field must not contain the following shell metacharacters:

```
! $ ^ & * ( ) { } [ ] \ | ; < > ? ' " `
```

The following fields are optional:

Relocatable

The value of this field should be either YES or NO (the default is NO). If the value of this field is YES, then the **DefaultLocation** field is used as a hint, providing the initial directory for the InstallIn panel. If the value of this field is NO or if it is omitted, then the user will not be prompted with the InstallIn panel and the value of the **DefaultLocation** field will be used as the root directory relative to which the contents of the package are installed. You should use the value YES if you want to allow the user to choose the destination of the installation.

Application

The value of this field should be either YES or NO (the default is NO). The value should be set to YES if (and only if) the package contains a NEXTSTEP application.

DeleteWarning

The value of this field will be displayed as the contents of an AlertPanel that prompts for confirmation when the user attempts to delete the package. The value of the **DeleteWarning** field should be short, preferably a single sentence. The field should be used when the developer wants to inform the customer of any dangers or side effects of deleting the package. The default value is ªThis action will remove the entire contents of the *package-name* package from your system.ºº

FtpSite

The value of this field will be used as the remote location of the compressed tar archive for a remote package. This field is used only if the package folder does not contain a compressed tar archive. The value of this field should be of the form *rhost***:***path*, where the colon separates the name of the remote FTP server from the pathname of the compressed tar archive on that server.

UseUserMask

The value of this field should be either YES or NO (the default is NO). If the value is YES, the permissions mode of directories that are created during installation will be determined by the formula (0777 & *umask*), where *umask* is the user's umask value (typically 022). If the value of this field is NO, then the permissions mode of directories that are created during installation will be preserved from the compressed tar archive. Setting the value of this field to YES can be used as a workaround when existing compressed tar archives (for example, on an FTP server) contain read-only directories, which cause installation failures when the valueis NO.

LongFileNames

The value of this field should be either YES or NO (the default is NO). If the files in a package have pathnames that are longer than 100 characters (the normal tar limit) then this field should be set to YES, which increases the pathname limit to 225 characters. The **-B** switch must be given to the **package** utility for this to work. See the section ªCreating an Installer Package.º

MyApp.sizes

This file is automatically generated by the **package** script. It contains various quantities that relate to the size of the package. Like the info file, this file contains a set of (*name value*) pairs, one pair per line. Blank lines or lines consisting entirely of whitespace are ignored. Lines whose first non-whitespace character is # are treated as comments and ignored.

The *value* part of each entry in the ª.sizesº file should be an integer number. The format of each line is:

[*optional-whitespace*] *name* *whitespace* *numeric-value*

The *name* must be a single token, not containing whitespace. The *value* is the first token following the *name*, and must consist entirely of digits.

Here is an example *MyApp***.sizes** file.

```
# The sizes file for the MyApp package
NumFiles              59
InstalledSize      69012
CompressedSize         3456
# end of MyApp.sizes
```

All three fields are required:

NumFiles The value of this field must be an integer that is the number of files in the package archive. This number can be generated by the command:

```
cat MyApp.bom | wc -l
```

InstalledSize

The value of this field must be an integer that is the size, in kilobytes, of the package in its installed form. The *MyApp* developer can compute this number by running the following command on the *MyApp***.root** directory (that is, the directory from which the archive was generated):

```
du -s MyApp.root
```

The output of this command is the size, in kilobytes, of the contents of the package. To this number, you should add the sizes of the *MyApp***.info**, *MyApp***.sizes**, *MyApp***.bom**, and *MyApp***.tiff** files (round up to the next highest kilobyte).

CompressedSize

The value of this field must be an integer that is equal to the size, in kilobytes, of the package in its uninstalled form (or nearly equivalently, its installed-and-compressed form). You can compute this field by generating the package with a dummy value for this field, and then running the command:

```
du -s MyApp.pkg
```

The output of this command is the size of *MyApp***.pkg** in kilobytes. This number can now replace the dummy value in the *MyApp***.sizes** file (round up to the next highest kilobyte).

MyApp.tiff

This optional file can contain an icon for the package that will be displayed in the Info view of the Package window. If the package contains an application, normally this file will contain the application icon. The icon should be a 48-by-48 TIFF image. This file can be in ordinary TIFF format, or it can be compressed using **tiffutil**(1).