Install version 1.3.1: Binary Manual

May 30, 2007

Contents

List of Figures

List of Tables

1 Introduction

This is the Binary Manual for Install, written for licensees holding a current Binary-only license for Install. It is a technical document whose intended audience comprises

programmers and developers with knowledge of XML plist files[1], Mac OS X bundles[2] and with (optional) experience in shell programming.

The document strives to be clear, concise, comprehensive and unambiguous. If you identify a specific aspect that should be clarified or corrected please contact Wincent at `win@wincent.com`.

1.1 Disclaimer

Please read this manual carefully and in its entirety before starting to work with Install. Install is a powerful program that can run with root privileges to perform many tasks, including copying and deleting files, moving files to the Trash, adding or removing login items, and running shell scripts. As such, configuration should only by attempted by people thoroughly familiar with the operation of the software.

The software and related documents and materials are provided "as is" without any express, implied or statutory warranty of any kind. In no event shall Wincent or its officers, employees, directors, subsidiaries, representatives, affiliates and agents have any liability to you or any other third party, for any lost profits, lost data, loss of use or costs of procurement of substitute goods or services, or for any indirect, special or consequential damages arising out of the license agreement.

For more detailed information about this disclaimer, please see the Install License.

1.2 Core ideas

Install is easy to configure. At the most basic level you need only provide a configuration file (described in more detail below) and a set of RTFD or RTF files containing text to be shown to the user as the installation progresses. More complicated installers can be constructed by including shell scripts, and it is even possible to embed other binaries inside Install and trigger them from within shell scripts. In all cases, everything that needs to be customized or added resides in the `Resources` folder within the Install application bundle.

1.2.1 Install steps

Install is written in Objective-C, primarily using Apple's Cocoa frameworks. It takes advantage of the Object Oriented design principles to provide a modular, extensible, easily-maintainable architecture. It is structured around the central idea of the install "step". Any installation configuration can be expressed in terms of a series of install steps. The information about the install steps — their order and their characteristics — is stored in an XML plist file, `InstallConfig.plist`, inside the Install application bundle. The steps are briefly described below, in the order that

[1] At the time of writing, Apple documentation on this topic is available at: `http://developer.apple.com/documentation/Cocoa/Conceptual/PropertyLists/Concepts/XMLPListsConcept.html`

[2] See: `http://developer.apple.com/documentation/MacOSX/Conceptual/SystemOverview/Bundles/chapter_44_section_1.html`

they would typically appear in an installation sequence. Detailed information about how to define the steps in the plist file appears in Section 3.

WOInstallerStepReadMe Most installers should start with a `WOInstallerStepReadMe` step to provide the user with information about what will be installed. The document that will be displayed should be an RTFD or RTF document stored within the `Resources` folder of the Install application bundle (the file must have an appropriate file extension, either "`.rtfd`" or "`.rtf`", which identifies its type). You can specify whether to display a white background behind the document, or allow the default Aqua window stripes to show through as shown in Figure 1. You can also specify whether the view which displays the document should have a scroll bar (the document in Figure 1 does not have a scroll bar). If you choose to display a white background then Install allows the user to select the text and to use the `Copy` item in the `Edit` menu.

If you provide a `WOInstallerStepReadMe` step as the first step in your installer, then Install can also display an "Uninstall" button if an existing install is detected on the target system[3]. In this way, your users can use Install not only to install and upgrade software, but to remove it as well. Pressing "Continue" moves onto the next step. Note that the "Go Back" button in Figure 1 is ghosted because there is no previous step.

If you desire, you can include multiple `WOInstallerStepReadMe` steps in your configuration, although this is usually not necessary because most of the other `WOInstallerStep` subclasses inherit from the `WOInstallerStepReadMe` class and can therefore also display RTFD or RTF files.

WOInstallerStepLicense The `WOInstallerStepLicense` step is a subclass of the `WOInstallerStepReadMe` step, which means that it inherits the characteristics of that step such as the ability to determine whether to display a background or to display scroll bars, but it adds a mechanism for obtaining consent to enter into a license agreement. Once again, the type of document to be displayed should be an RTFD or RTF file (with an "`.rtfd`" or "`.rtf`" file extension). The user is presented with an "Disagree"/"Agree" sheet attached to the main Install window as shown in Figure 2. By default, the "Agree" button retains the focus and the user can accept the license agreement by pressing Return or Enter.

Note that in the window shown in Figure 2 that the view has been set to show a background and to include a scroll bar.

The `WOInstallerStepLicense` step is optional, and if you decide to use it your installer should include one and only one such step. When the user clicks the "Continue" button Install will display the sheet and on agreement proceed to the next step. If the user disagrees Install remains at the `WOInstallerStepLicense` step, and if the user again presses "Continue" then the sheet will be displayed again. If the user later uses the "Go Back" button to return to the `WOInstallerStepLicense`

[3] As of Install version 1.1.2, the `WOInstallerStepLicense`, `WOInstallerStepShell`, `WOInstallerStepDomain` and `WOInstallerStepCopyFiles` steps will also display an "Uninstall" button if they are the first step in your installer and an existing install is detected.

Figure 1: Sample WOInstallerStepReadMe step

step after having proceeded beyond it, then Install will not re-display the sheet, but will proceed directly to the next step (Install only requires the user to agree to the license agreement once per run).

WOInstallerStepShell The `WOInstallerStepShell` step is an optional step which can be used to run shell scripts which you store inside the `Resources` folder of the Install application bundle. This capability allows you to construct powerful installers that can perform almost anything that can be described within a shell script. You could even embed binaries of your own design inside the Install application bundle and trigger them from within shell scripts. One potential application for the `WOInstallerStepShell` step is the implementation of complex pre-install tests to verify whether a target system is compatible with software to be installed.

Scripts can be set to run with or without root privileges (in the former case, the user will be asked to introduce a valid administrator username and password). They can be run using any shell present on the system (safe choices for the shell include `/usr/bin/perl`, which is present on all versions of Mac OS X, `/bin/tcsh`, which is the default on Mac OS X versions prior to 10.3, `/bin/bash`, which is the default on Mac OS X 10.3, and `/bin/sh`, which is actually an alias to `/bin/bash`. Scripts can also be supplied with zero or more arguments as specified in the Install configuration file. You can specify which exit codes should be considered fatal errors (stopping the installer) and which should be considered non-fatal (showing an explanatory sheet and allowing the user to choose between "Retry" or "Quit"). You can also specify custom messages to be displayed in a sheet on the completion

Note: If the user selects "Retry" then only the failed script will be re-run, not any other scripts which appeared before it in the sequence.

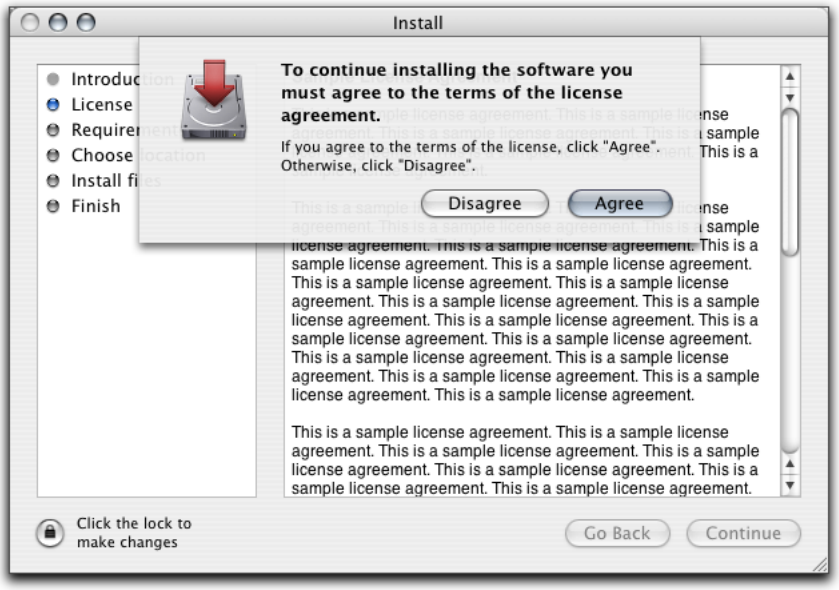

Figure 2: Sample WOInstallerStepLicense step

of a given script. In order to ensure that the Install user interface remains responsive during script execution, scripts are run in a separate thread, and a progress indicator appears along with a status string in the main Install window (as shown in Figure 3) whenever a script is being run.

As a `WOInstallerStepReadMe` step subclass, the `WOInstallerStepShell` step requires an RTFD or RTF document to display (with an appropriate "`.rtf`" or "`.rtfd`" file extension), and you can turn the document background and scroll bars on or off. If one or more of your scripts is to execute with root privileges then it is recommended that your `WOInstallerStepShell` document alert the user that they will have to enter a valid administrator username and password before the step can complete. The user can authenticate as an administrator at any time by clicking the padlock icon in the lower left corner of the Install main window, but Install will automatically prompt the user if authentication is required but has not yet been obtained. If authentication will be required Install will ask for it prior to running any of the scripts in the `WOInstallerStepShell` step. Having obtained authentication, it then runs all of the scripts in the step, launching the appropriate scripts with root privileges and running the others without privilege elevation. In this way the prompt for authentication appears as soon as the user clicks the "Continue" button, and not at some intermediate point (for example, in the middle of five or ten-script sequence) to avoid confusing the user.

Where multiple shell scripts are specified, they are executed in the order that they appear in the configuration file. The same is true for shell scripts executed within the `WOInstallerStepCopyFiles` and `WOInstallerStepUninstall` steps that will be

Figure 3: Sample WOInstallerStepShell step

discussed below. Order is not a consideration in the `WOInstallerStepFinish` step (also discussed below), because you can only run a single script within that step. You may specify multiple `WOInstallerStepShell` steps within the Install configuration file if you desire.

WOInstallerStepDomain This is an optional step for installers which wish to differentiate between the "Global" domain and the "User" domain. Material installed in the Global domain should be available to all users on the machine (for example, material installed into `/Library/`) whereas material in the User domain should be available to the current user only (in this example, material installed into `~/Library/`). Installations into the Global domain require a valid administrator username and password, and file operations are performed as root (for example, files and folders are copied using the equivalent of the commandline `sudo ditto` command, and deletions are performed with the equivalent of `sudo rm`[4]). Installations into the User domain do not require special privileges, as they should only write to areas of the disk for which the user running Install has write access (for example, within the user's home folder). Unlike in Global domain installs, in User domain installs files are copied using Cocoa APIs and moved to the Trash rather than being immediately deleted.

The user, however, is shielded from these implementation details by a simple

[4]In all cases, `sudo` itself is never invoked by Install; rather Install uses Apple's Security Framework to launch commands with elevated privileges. Install itself does not run with root privileges, but it can launch subprocesses which do.

Figure 4: Sample WOInstallerStepDomain step

choice presented as shown in Figure 4 which asks them to "Choose where to install the software": either, "Install only for the current user", or, "Install for all users on this machine".

If you do not specify a `WOInstallerStepDomain` step in your configuration, Install will use the User installation domain as default, although you can override this and force Install to use the Global domain as a default (as described in Section 3.4 on page 20). You may optionally override the default text displayed in the top third of the Install main window by supplying an appropriate RTF or RTFD with custom text. You may wish to do this if you prefer to use text that features the actual name of the software to be installed, rather than generic text such as "Choose where to install the software" and "If you choose to install only for the current user, the software will be installed in your home directory".

WOInstallerStepCopyFiles This is the most important step in any configuration because it is where the work of actually installing the software is performed. In reality, this step is comprised of the following substeps which are executed in order:

Preinstall kill processes You may specify a list of target processes using bundle identifiers (for example, `com.apple.systempreferences`) which must be quit before installation can continue. You decide whether to quit the processes by sending them a "quit" Apple Event (which would mean that the target application could provide the user with a chance to save or apply any unsaved work or changes before exiting) or by sending them with a SIGTERM or

SIGHUP signal. The attempt to quit target processes occurs in a separate thread to ensure that the Install user interface remains responsive throughout. Install effectively enters an idle state while waiting for target processes to exit because it waits for notification from the operating system that this has occurred (in other words, it does not do any polling). Install employs this technique for all processes, even faceless background applications that can not normally be detected using the Cocoa APIs. It is sensible to kill older versions of the software to be installed (if you are attempting an upgrade), or other processes that might be accessing files which you wish to modify, remove or overwrite as part of the installation. You can specify any number of processes.

Preinstall trash files and folders Install can effectively be used not only for clean installations, but for upgrades as well. In this substep you could remove old versions of the software before installing the new versions. Items in the Global domain are Deleted using root privileges and items in the User domain and moved to the Trash. Once again this occurs in a separate thread to guarantee responsiveness. You can specify any number of files or folders.

Remove login items You can remove login items during this substep, which may be useful to you if, for example, you want to upgrade any old versions of your software that may be present on the target system and plan on removing references to the old version in the login items and later replacing them with references to the newly installed version. You can specify any number of login items. You can also optionally specify a `User domain only` or `Global domain only` key in order to remove a given item only under certain types of installations.

Run preflight scripts In this substep, prior to any files being copied, you can run shell scripts much as you can in the `WOInstallerStepShell` step; with or without root privileges, an optional custom shell setting, and optional arguments. Like the `WOInstallerStepShell` step you can control Install's behaviour in response to the different exit codes. You can specify any number of scripts, and they will be executed in the order in which they appear in the configuration plist. Script execution occurs in a separate thread to maintain user interface responsiveness.

Install files and folders This is the substep in which the files and folders are copied to the target machine in the locations that you specify, in accordance with whether the user has opted to install into the Global domain or the User domain. Files to be copied into the Global domain are copied with root privileges, and files to be copied into the User domain are copied without elevated privileges. In both cases Macintosh-specific filesystem information such as resource forks and metadata is preserved when writing to an HFS+ target disk. Once again, this substep occurs in a separate thread to maintain responsiveness. You may specify any number of files and folders in the configuration plist. The order in which items are copied is not defined.

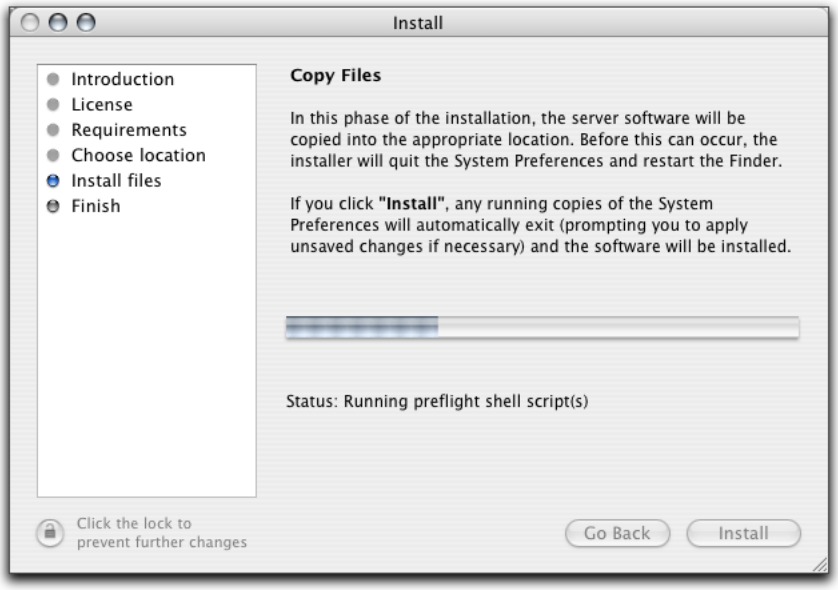

Figure 5: Sample WOInstallerStepCopyFiles step

Run postflight scripts In this substep, after all the files or folders have been succesfully copied, you can specify one or more postflight scripts that will be run to perform post-installation tasks. As is the case with the preflight scripts, you can run scripts with or without root privileges, a custom shell, and optional arguments, and you can specify Install's response to possible exit codes. You can specify any number of scripts, which will be executed in a separate thread, in the order specified in the configuration file.

Add login items As a counterpart to the `Remove login items` substep above, the `Add login items` substep can be used to add one or more files or applications to the installing user's login items. There is no limit to the number of login items that you may specify in the Install configuration file. And like the `Remove login items` substep you may also optionally specify a `Global domain only` or `User domain only` key.

Figure 5 shows the user interface for the `WOInstallerStepCopyFiles` step. As this is a `WOInstallerStepReadMe` subclass, you must specify an RTFD or RTF document to display during the step. This is a good place in which to provide further information to the user exactly what will happen during the installation.

It is in theory possible to include more than one `WOInstallStepCopyFiles` step in your Install configuration, although in most cases it would be more desirable to do everything in a single instance of the step.

WOInstallerStepFinish The `WOInstallerStepFinish` step is a highly-recommended step which should appear as the final step in any Install configuration. It is used to display information to the user and can can be used to launch installed applications or open documentation. In fact, it can open (or launch) any set of files with any application that you optionally specify. It is another subclass of `WOInstallerStepReadMe` that requires you to specify an RTFD or RTF document (with corresponding file extension, "`.rtfd`" or "`.rtf`") providing information to the user as shown in Figure 6. As with the parent class, you can instruct the `WOInstallerStepFinish` step to show or not show a background, and include or not include a scrollbar.

You may also run a single shell script from within this step, with or without root privileges. Note that you may only run one such script in this step, unlike the other shell-script-capable steps in which you can specify a sequence of scripts to run. Also, unlike the other steps, there is no progress indicator in the Install user interface to indicate that the script is running (although the script does run in a separate thread for the usual reasons), so you should endeavour to make sure that your script is quick to run. The principal purposes for running a script in this step are to launch installed programs with root privileges, if appropriate (a valid example would be launching a background daemon process with elevated privileges), or to advise the user to logout (or even initiate a logout, using AppleScript and the `osascript` command line tool).

As visible in Figure 6, the `WOInstallerStepFinish` step is capable of displaying three buttons. In the example, the rightmost button has been assigned the custom label, "Launch". The purpose of this button is described under `Primary button action` section in the configuration file. If you do not specify a `Primary button action` then the button retains its default label ("Continue") but is ghosted. In this case, the action causes a shell script to be run with root privileges which in turn launches the installed server daemon. The script must return an exit code of 0 (zero) to indicate success, otherwise Install will consider the operation to have failed and will terminate with a warning dialog.

The leftmost button has been given the custom label of "Help" and when clicked will open the help files using the application specified in the configuration file, as defined in the `Secondary button action` section of the configuration. If you do not specify a `Secondary button action` then the secondary button will not appear in the user interface.

Note that for each button action you can specify a set of files to open and applications to launch them with, *or* you can specify a single shell script to launch, but you cannot do both types of operation within a single button action. In all cases, after the action has been successfully completed, Install quits, so you should take care when writing the RTFD (or RTF) file for the `WOInstallerStepFinish` step that you make it clear to the user that for each button they might choose to click, the final result will be that the installer will exit.

The `WOInstallerStepFinish` step is technically not compulsory, in that Install will function correctly even without such a step, but it is highly recommended because the absence of such as step is likely to be highly confusing to the user. For example, if a `WOInstallerStepCopyFiles` step were the last step in the in-

Figure 6: Sample WOInstallerStepFinish step

stallation, Install would immediately exit as soon as the step successfully finishes, effectively disappearing from the screen, which the user may incorrectly perceive as a crash.

WOInstallerStepUninstall Unlike the steps described above the `WOInstallerStepUninstall` step is never used as part of an normal install sequence. Rather, it is used as an independent step that is activated in isolation when the user presses the "Uninstall" button upon launching the installer. Note that the button will only appear in the user interface if Install detects an existing installation already on the target machine.

Like the `WOInstallerStepCopyFiles` step, this class is actually composed of a number of substeps. The substeps run in the following order: `Uninstall kill processes`, `Run preflight scripts`, `Uninstall trash files and folders`, `Remove login items` and `Run postflight scripts`. The functioning and specification of these substeps is equivalent to that of the corresponding substeps in the `WOInstallerStepCopyFiles` step so consult the information starting on page 8 for more details. The core points to remember are:

- The processes to be exited in the `Uninstall kill processes` phase are specified by using bundle identifiers.
- Processes can be caused to exit by sending a "quit" Apple Event or by using the SIGTERM and SIGHUP signals.
- Any number of scripts can be run in the `Run preflight scripts` and `Run`

`postflight scripts` substeps, and scripts are executed in the order that they appear in the configuration file.

- Scripts can be run with or without root privileges, you can specify a custom shell and optional arguments, and scripts must return an exit code of 0 (zero) or Install will consider the operation to have failed.

- You can specify any number of login items to be removed in the `Remove login items` substep.

The `Uninstall trash files and folders` substep differs slightly in terms of how it is described in the configuration plist file. This is because at the time the `WOInstallerStepUninstall` step is initiated the target domain (Global or User) has not been defined and so Install cannot know exactly which files will need to be removed. Therefore, the user is presented with a list of possible files (or folders) to remove as shown in Figure 7 and uses checkboxes to indicate which files or folders should be uninstalled and which files or folders left alone. For each item that might be removed, you define whether or not the checkbox should start off checked by default and whether the item will require a valid administrator user name and password to remove. A general rule is that files in the Global domain will require elevated privileges (and be marked with a padlock icon in the user interface) while files in the User domain will not. Install makes sure that the list shown to the user only contains files and folders actually present on the users system. In general it is advisable to encourage the user to leave items such as preference files for installed software intact (by specifying that these items should not have their checkboxes checked by default).

Once the user has clicked "Continue" to initiate the uninstallation, Install begins the necessary operations in a separate thread and updates the file listing in real time to show what substep is underway and which file or folder is in the process of being deleted (or has successful been deleted). On success, the user may exit by pressing "Quit", or press the "Go Back" button to be returned to the first Install step (usually a `WOInstallerStepReadMe` step) and if desired re-install the software immediately.

2 Internal structure of the Install application bundle

2.1 Support files

As already stated, the Install support files (including the configuration plist, any RTFD or RTF document files and shell scripts) are stored inside the Install application bundle in the Resources folder. The base location for this folder is `Install.app/Contents/Resources/`, and so you might have a "ReadMe" file at `Install.app/Contents/Resources/ReadMe.rtf`.

If you plan to distributed internationalized versions of your software then it makes sense to store these resources in language specific subdirectories of the `Resources` folder. In the case of the "ReadMe" example just given, you could instead put the English language version of the file at `Install.app/Contents/Resources/`

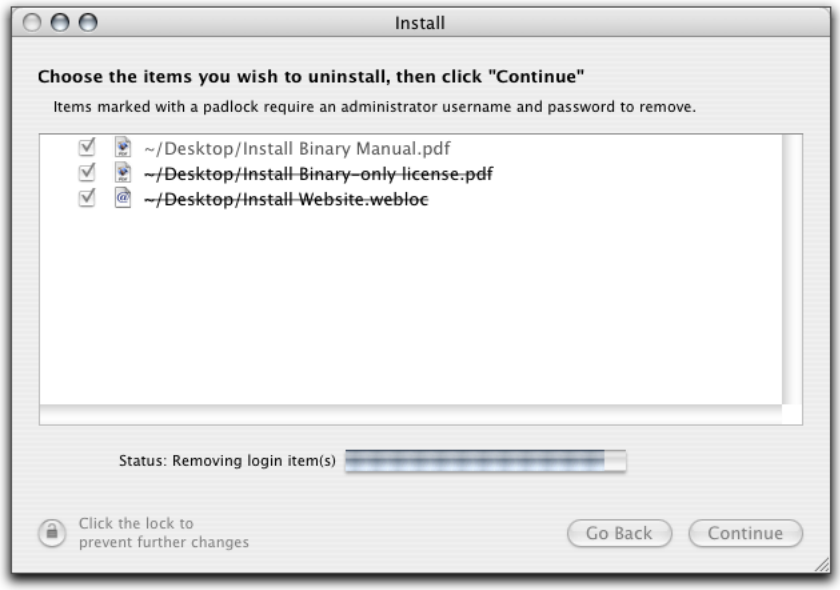

Figure 7: Sample WOInstallerStepUninstall step

`English.lproj/ReadMe.rtf` and Install would automatically make use of it. Even if you do not currently have plans to ship non-English versions of your software, it is considered to be best practice to store your English language resources in the `English.lproj` folder in case your internationalization plans change in the future.

For the most part, shell scripts can reside directly in the `Resources` folder because they do not necessarily produce and user-visible output (unless you write to the console from the script), but if you wish you could provide language-specific variants and store them in the `English.lproj` and other lproj folders.

It is common practice for the configuration plist to reside in `Install.app/Contents/Resources/English.lproj/InstallConfig.plist` rather than directly in the `Resources` folder, because the configuration plist file does contain user-visible information, such as the words used for the installation steps (visible in the left-hand side of the Install main window), and any custom response messages you may have defined for shell scripts, or modifications to the default button names used for the `Primary button action` and `Secondary button action` in the `WOInstallerStepFinish` step.

2.2 Files to be installed

2.2.1 Localization of the Files folder

The files to be installed are stored inside the Install application bundle at `Install.app/Contents/Resources/Files.localized/`. If you view this directory in the Terminal you will see that it's name is indeed `Files.localized` and that it contains

a further, invisible folder called `.localized` which contains a number of strings files. These strings files allow the Finder to display a localized title for the `Files.localized` directory. If you look at the folder in the Finder, you might see its name as "Files" (the Finder hides the ".localized" extension). As an example, if your system language is set to Spanish, you would see the filename as "Archivos", which is the Spanish word for "Files".

If, however, you have your Finder preferences set so that file extensions are always shown, then you will not see the localized variant of the folder name and you will only ever see it as "Files".

The reason Install goes to such trouble to localize the name of a folder buried inside the application bundle is that Install itself features a "Reveal files on disk..." menu item which users can use to expose the folder and browse it using the Finder. This can be useful for users who are simply curious, for situations in which expert users wish to do a manual (drag and drop) installation, and for cases where a copy operation during an installation fails and Install advises the user to reveal the files on disk and attempt the copy manually.

2.2.2 Structure of the Files folder

The `Files.localized` directory contains a numbered series of folders, each containing a single item that Install will copy during the installation. The numbering starts at 0 (zero) and increases in whole number increments, so if your configuration required Install to copy four distinct items then the `Files.localized` directory would contain four folders — `0`, `1`, `2` and `3` — and each of those folders would contain one and only one of the items to be installed.

By placing each item in a separate folder, Install can easily allow you to exercise fine-grained control over what will be installed and where. Each item — where an "item" can be either a single file or a folder (potentially containing other items) — has a set of corresponding parameters in the configuration plist which specify what the item is called, where it should be installed if installing to the Global domain, where to install if installing to the User domain.

Although the order in which files will actually be copied is undefined (and you should not depend on it as the current order may change in subsequent releases of Install), there is a direct correspondence between the numbering of the folders inside the `Files.localized` directory and the order in which the items are defined in the configuration plist. The following example, based on an installer that installs two PDF documents and an application called "DXP", illustrates:

`Man.pdf` The first file to be defined in the configuration plist, and so it must be stored in the first subfolder inside the `Files.localized` directory; in other words, at `Install.app/Contents/Resources/Files.localized/0/Man.pdf`. The configuration plist specifies that it should be placed in `~/Documents/DXP Documents/` for User domain installs and in `/Users/Shared/Documents/DXP Documents/` for Global domain installs. In both cases, Install will create the target directory if

required and create any other necessary intermediary directories along the way.

`DXP.app` The second file to be defined in the configuration plist, and so it must be stored in the second subfolder inside `Files.localized`, which is means that it will be stored at `Install.app/Contents/Resources/Files.localized/1/DXP.app`. The configuration plist specifies that it should be stored in `/Applications/` or in `~/Applications/` depending on the installation domain. Note that in this case the item being installed is actually a directory (an application bundle) but it is still only considered as a single item, and therefore is compliant with the Install requirement that each folder inside `Files.localized` only contain a single item.

`FAQ.pdf` The third file defined in the configuration plist, and so it must be stored at `Install.app/Contents/Resources/Files.localized/2/FAQ.pdf`. The configuration specifies that it should be installed into `/Users/Shared/Documents/DXP Documents/` or `~/Desktop/` depending on the installation domain.

3 The plist configuration file

Install takes its instructions from an XML (plist) configuration file stored inside its application bundle. The format of this file is quite straightforward, but it as essential that you adhere exactly to the format as specified in this manual. If you stray from the format Install will throw a `WOMalformedPlistException`, causing the program to exit immediately and log a message to the console. It is therefore very important to read the following sections of the manual with care, and to always test your customizations to make sure that they work as you expect, before distributing files to your customers using the customized installer.

To be a valid plist file the configuraiton must begin with the following lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
```

The example configuration templates then contain the following comment. This is to disuade users from looking inside the Install bundle and editing the plist file manually. Because Install bases its actions on the contents of the configuration file, and because it engages in potentially destructive actions – including copying and deleting files, moving files to the Trash, adding and removing login items, and running shell scripts, with or without root privileges – it is important that the end user does not attempt to manipulate the settings. Only a licensed developer fully familiar with the Install documentation and plist configuration file format should attempt to edit the file.

```
<!-- WARNING: DO NOT EDIT THIS FILE -->
```

The next element and only element in the file is an array containing a sets of installation instructions. This array and its contents will be discussed in the following sections. In the example below, a comment shows where the installation instructions should appear. The final entries in the file are the closing </array> and <plist> tags.

```
<array>
    <!-- the installation instructions go here -->
</array>
</plist>
```

3.1 Configuration file: Root level structure

As noted above at the root of the configuration file there is only one object: an array containing sets of installation instructions.

In the simplest and most common configuration, there is only one set of instructions in the array. It is, however, possible to specify different installation instructions depending on the version of Mac OS X running on the target machine. For example, you may wish to install one item on machines running Panther, and a different item on machines running Jaguar.

The example skeleton below shows how three different sets of installation instructions would appear in the configuration plist file. Each set of instructions is contained in a dictionary:

```
<array>
    <dict>
        <!-- first set of installation instructions -->
    </dict>
    <dict>
        <!-- second set of installation instructions -->
    </dict>
    <dict>
        <!-- third set of installation instructions -->
    </dict>
</array>
```

A simpler configuration file might have only one set of installation instructions:

```
<array>
    <dict>
        <!-- first and only set of installation instructions -->
    </dict>
</array>
```

3.2 Major components within each set of installation instructions

Within each set of installation instructions there are up to four possible major components.

1. The `Target OS version` dictionary (optional) which specifies the range of versions of Mac OS X for which the set of installation instructions should apply. The format of this dictionary is detailed in Section 3.3 below.

2. The `Default installation domain` string (optional) which specifies whether items should be installed for all users (the "Global" domain) or for the current user only (the "User" domain). Discussion of this string appears in Section 3.4 below.

3. The `Install check` array (optional) which specifies a list of files or folders which, if present on the target system, indicate the presence of a pre-existing installation of the software. The format of the array is described in Section 3.5.

4. The `Installation steps` array (compulsory) which specifies the steps the installer will execute during installation. The format of the array is covered in Section 3.6 on page 21.

5. The `Uninstall step` dictionary (optional) which provides instructions on how Install should perform an uninstallation. Note that if you specify an `Install check` array then you must also provide an `Uninstall step` dictionary, and vice a versa. The format of the Uninstall step dictionary is described in Section 3.7 on page 39.

The four components are siblings, in that they appear at the same level within the configuration plist, as illustrated in the following skeleton example:

```
<key>Target OS version</key>
<dict>
    <!-- target OS specification -->
</dict>
<key>Default installation domain</key>
<string><!-- Global or User --></string>
<key>Install check</key>
<array>
    <!-- check for pre-existing installation -->
</array>
<key>Installation steps</key>
<array>
    <!-- the steps that comprise an installation -->
</array>
<key>Uninstall step</key>
<dict>
```

```
    <!-- how to perform an uninstall -->
</dict>
```

3.3 Specifying different installation instructions depending on the version of Mac OS X on the target machine

Each set of installation instructions may contain the Target OS version key and associated dictionary. Within the dictionary a string formatted in hex notation is used to denote the operating system version of the machine onto which the software will be installed. Examples of valid hex strings include: 0x00001020 (Mac OS X 10.2), 0x00001030 (Mac OS X 10.3) and 0x00001027 (Mac OS X 10.2.7). The Target OS version dictionary looks like this:

```
<key>Target OS version</key>
    <dict>
    <key>Maximum system version</key>
    <string>0x00001030</string>
    <key>Minimum system version</key>
    <string>0x00001020</string>
</dict>
```

The Target OS version dictionary is itself optional. If you decide to include it, within the dictionary you must specify at least one of Minimum system version and Maximum system version. By combining these entries you can create quite sophisticated installations customised for specific versions of the OS.

The following example would ensure that the software is only installed on Jaguar and above:

```
<key>Target OS version</key>
<dict>
    <key>Minimum system version</key>
    <string>0x00001020</string>
</dict>
```

The following example would ensure that the software is installed on any version of Jaguar, but not on Panther or above:

```
<key>Target OS version</key>
<dict>
    <key>Maximum system version</key>
    <string>0x00001028</string>
    <key>Minimum system version</key>
    <string>0x00001020</string>
</dict>
```

The following example would ensure that the software is only installed on Mac OS X 10.3.1:

Figure 8: Dialog displayed when installer is run on incompatible OS

```
<key>Target OS version</key>
<dict>
    <key>Maximum system version</key>
    <string>0x00001031</string>
    <key>Minimum system version</key>
    <string>0x00001031</string>
</dict>
```

In your installers you can use any number of customised sets of installation instructions tailored to specific versions of the OS, or you can just use a single set for all versions.

The ordering of items in the configuration plist file is important. The installer will search through the file looking at each set of installation instructions. If it finds a set of instructions without a Target OS version dictionary, it will stop scanning and proceed using those instructions. Otherwise, it continues scanning, and whenever it encounters a `Target OS version` dictionary it tests to see with the target machine falls within the boundaries specified by Minimum system version and/or Maximum system version. On the first successful match, the installer stops scanning and proceeds using those instructions. If no successful matches are found, the installer displays a dialog as shown in Figure 8 explaining that the software cannot be installed.

3.4 The Default installation domain

A sibling of the optional `Target OS version` dictionary is the `Default installation domain` string, which is also optional. There are only two possible values for this string. In the first case, the `Global` case, Install will by default install items for all users on the machine:

```
<key>Default installation domain</key>
<string>Global</string>
```

The other possible case is the `User` case, in which Install will by default install items for the current user only.

```
<key>Default installation domain</key>
<string>User</string>
```

If you do not specify a `Default installation domain` then Install will use the User domain as the default. As such, you only need to specify the `Default installation domain` when you wish to force Install to use the Global domain as the default; in all other cases you may omit it. If you wish to allow the user to choose the installation domain, then your installer should include a `WOInstallerStepDomain` step.

3.5 The Install Check

A sibling of the optional `Target OS version` dictionary and `Default installation domain` string is the `Install check` array, which is also optional. You can specify files and folders in this array (as strings containing the paths to the items), and Install will search the disk of the target machine for those files or folders. If one or more of the items is present then the machine is considered to have a pre-existing installation of the software, and Install will include an "Uninstall" button when it first launches (if and only if the first step defined in the plist file is of type `WOInstallerStepReadMe`). An example `Install check` array looks like this:

```
<key>Install check</key>
<array>
    <string>/Users/Shared/Install Manual.pdf</string>
    <string>~/Desktop/Install Binary Manual.pdf</string>
    <string>~/Desktop/Install Website.webloc</string>
</array>
```

You can include any number of files or folders in the array, and you may use ~ as a shorthand to signify the user's home directory. There is no need to escape spaces that may appear in the paths of the files.

You can safely omit the `Install check` array from your configuration if you desire but if you do so then your custom installer will not offer any uninstallation functionality. If you *do* include the `Install check` array then it is compulsory to also supply the `Uninstall step` dictionary, which contains the instructions for how to perform the uninstallation. If you fail to supply the dictionary and the user clicks the "Uninstall" button Install will log an error message to the console, and if the user attempts to continue with the uninstall operation Install will "perform" an uninstallation which consists of zero substeps (in other words, it will appear to the user that the uninstallation has succeeded instantly, but Install will not have many any changes to the target machine). Detailed information on the `Uninstall step` specification appears in Section 3.7 on page 39.

3.6 Installation steps

The `Installation steps` array is a sibling of the optional `Install check` array and `Target OS version` dictionary discussed above. It is compulsory, and is the

most important component of the configuration file because it defines the steps which will be executed in running the installer. If you fail to provide such an array then Install will log an error message to the console and refuse to launch.

A basic skeleton for the `Installation steps` array is the following:

```
<key>Installation steps</key>
<array>
    <!-- first installation step (show ReadMe) -->
    <!-- second step (show License) -->
    <!-- third step (check system requirements) -->
    <!-- fourth step (choose target domain) -->
    <!-- fifth step (copy files etc) -->
    <!-- sixth step (show Finish document) -->
</array>
```

Steps are executed in the order in which they appear in the array. It is recommended although it is not compulsory that you always include a `WOInstallerStepReadMe` step as the first step in any installation, so that you can provide information to the user and offer them the opportunity to uninstall as well (the "Uinstall" button will only appear in the first installation step, and only if that step is of class `WOInstallerStepReadMe`).

Similarly, it is also strongly recommend that you include a `WOInstallerStepFinish` step as the last installation step, so that you can present the user with notification that the installation finished successfully and provide them with the opportunity to launch the installed software, read the documentation, or perform some other task. If you do not specify a `WOInstallerStepFinish` step then the installer will simply exit after completing the installation, providing no notification to the user. You may only specify one `WOInstallerStepFinish` step, and it must be the last item in the array of steps.

The `WOInstallerStepCopyFiles` step is compulsory, and is the step which actually does the work of installing the software. It is in theory possible to include multiple `WOInstallerStepCopyFile` steps in your installation, but it is generally much better to specify all of the needed tasks in a single `WOInstallerCopyFiles` step.

The other possible steps, `WOInstallerStepLicense`, `WOInstallerStepShell` and `WOInstallerStepDomain` are optional. You should include no more than one `WOInstallerStepLicense` step, and no more than one `WOInstallerStepDomain` step in your configuration. In the case of the `WOInstallerStepShell` step, it is permitted to include multiple different steps of this class, but in general it is advisable for reasons of clariy to incorporate all of the needed functionality in a single step (the step can run multiple shell scripts).

In every case, a step is defined by a dictionary. The following sections describe the formats used for creating dictionaries corresponding to each installer step class.

3.6.1 WOInstallerStepReadMe

A basic `WOInstallerStepReadMe` step dictionary might look like this:

```
<dict>
    <key>Document</key>
    <string>ReadMe</string>
    <key>Show background</key>
    <false/>
    <key>Show scrollbar</key>
    <false/>
    <key>Step class</key>
    <string>WOInstallerStepReadMe</string>
    <key>Step name</key>
    <string>Introduction</string>
</dict>
```

The following key/object pairs may appear in the dictionary:

Document A string containing the filename, without an extension, indicating the RTFD or RTF document to display in the Install main window when the step executes. Install looks in the `Resources` folder first for a file with the specified name and an "`.rtfd`" extension, and if one cannot be found it then tries an "`.rtf`" extension. If a language-specific localization of the file is present in an lproj subdirectory inside the `Resources` folder then that file will be used first. This key/object pair is compulsory.

Show background A boolean value indicating whether to display a white background behind the document. If you omit this key/object pair, Install will use the default setting of true.

Show scrollbar A boolean value indicating whether to include a scroll bar when showing the document. If you omit this key/object pair, Install will use the default setting of true. It should be noted that setting `Show background` to false and `Show scrollbar` to true is a valid combination but one which looks ugly on the screen.

Step class A string indicating the class of the step, in this case `WOInstallerStepReadMe`. This key/object pair is compulsory. If you omit it Install will throw a `WOMalformedPlistException` and exit.

Step name The name of the step as it should appear in the steps listing on the left-hand side of the Install main window. This key/object pair is compulsory. If you omit it, an unattractive blank entry will appear in the listing.

3.6.2 WOInstallerStepLicense

`WOInstallerStepLicense` is a subclass of `WOInstallerStepReadMe`, which means that it inherits the capabilities and settings of its parent class. You may use the same key/object pairs as described in Section 3.6.1 above. A sample dictionary might be:

```
<dict>
    <key>Document</key>
    <string>License</string>
    <key>Step class</key>
    <string>WOInstallerStepLicense</string>
    <key>Step name</key>
    <string>License</string>
</dict>
```

The following key/object pairs may appear in the dictionary:

Document As per `WOInstallerStepReadMe`.

Show background As per `WOInstallerStepReadMe`.

Show scrollbar As per `WOInstallerStepReadMe`.

Step class A string indicating the class of the step, in this case `WOInstallerStepLicense`. This key/object pair is compulsory. If you omit it Install will throw a `WOMalformedPlistException` and exit.

Step name As per `WOInstallerStepReadMe`.

Note that in the example provided above, the dictionary does not specify values for Show background and Show scrollbar, and so the default values of true and true are used. This is usually approriate for license documents because they tend to be too long to fit within the Install main window without scrolling.

3.6.3 WOInstallerStepShell

The `WOInstallerStepShell` class is a subclass of `WOInstallerStepReadMe` and so inherits its capabilities and settings while adding some important additions. The basic structure of the `WOInstallerStepShell` step dictionary is shown in the following skeleton dictionary:

```
<dict>
    <key>Document</key>
    <string>Requirements</string>
    <key>Scripts</key>
    <array>
       <!-- an array of scripts -->
    </array>
    <key>Step class</key>
    <string>WOInstallerStepShell</string>
    <key>Step name</key>
    <string>Requirements</string>
</dict>
```

The following key/object pairs may appear in the dictionary:

Document As per `WOInstallerStepReadMe`.

Show background As per `WOInstallerStepReadMe`, but with one important difference: If this key/object pair is omitted from the configuration, then a default value of false will be used (unlike `WOInstallerStepReadMe` which uses a value of true as the default). The reason for this is that the primary function of the `WOInstallerStepShell` step is to perform operations on the machine, not to provide the user with large amounts of information, and so a default no-background/no-scrollbar appearance is better suited.

Show scrollbar As per `WOInstallerStepReadMe`, but the default value is false rather than true.

Scripts An array of scripts to run, using the order in which they appear in the array. The script definition format is described below.

Step class A string indicating the class of the step, in this case `WOInstallerStepShell`. This key/object pair is compulsory. If you omit it Install will throw a `WOMalformedPlistException` and exit.

Step name As per `WOInstallerStepReadMe`.

Note that the `Show background` and `Show scrollbar` fields in the example above are omitted, and so Install would automatically use the default values (false and false).

Script definition format Each script defined in the script array is specified by a dictionary. The dictionary should contain a key/object pair for identifying which script to run, and optional key/object pairs specifying which shell to use, arguments to be supplied to the script, whether root privileges are required, and how to respond to exit codes. This is an example dictionary:

```
<dict>
    <key>Shell</key>
    <string>/usr/bin/perl</string>
    <key>Script filename</key>
    <string>preinstall-test.sh</string>
    <key>Script arguments</key>
    <array>
        <string>Hello world</string>
        <string>from Install</string>
    </array>
    <key>Run script with root privileges</key>
    <true/>
    <key>Error handling</key>
    <array>
        <!-- error handling dictionary 0 -->
        <!-- error handling dictionary 1 -->
```

```
            <!-- error handling dictionary 2 -->
        </array>
    </dict>
```

The following key/object pairs may appear in the dictionary:

Shell A string defining the shell to be used to execute the shell script. You should take care to only define a shell which you know is available on the target version of Mac OS X to which you are installing. For example, `/bin/sh` is available on all versions (as an alias), `/bin/tcsh` is Apple's default on Mac OS X 10.2, `/bin/bash` is Apple's default on Mac OS X 10.3, and `/usr/bin/perl` should be available on both. This key/object pair is optional, and if not specified Install will use a default value of `/bin/sh`.

Script filename A string containing the filename of the script to be executed. The script should be stored in the `Resources` folder of the Install application bundle. Install will use a language-specific variant if one is available. This key/object pair is compulsory.

Script arguments An array of strings containing arguments to be passed to the script. It is not necessary to escape spaces within the argument strings. There is no arbitrary limit to the number of arguments that can be supplied. This key/object pair is optional.

Run script with root privileges A boolean indicating whether the script should be run with root privileges. If true, it is recommended to advise the user in that they will be asked for a valid administrator username and password. This can be done in the `Document` that corresponds to the `WOInstallerStepShell` step. The `Run script with root privileges` key/object pair is optional, and if omitted Install uses a default value of false.

Error handling An array of error handling dictionaries, each dictionary specifying how should respond to a particular exit code. The format for an error handling dictionary is discussed below. The `Error handling` key/object pair is itself optional. If you omit it, Install will consider non-zero exit codes to be fatal errors and will terminate the installation; exit codes of 0 (zero) are considered to indicate success and the installation proceeeds.

Script error handling arrays A script error handling array comprises one or more error handling dictionaries. A basic error handling dictionary could look like this:

```
<dict>
    <key>Exit code</key>
    <integer>1</integer>
    <key>Text to display on error</key>
    <string>You must shut down filesharing.</string>
    <key>Error Type</key>
    <string/>Non-fatal</key>
</dict>
```

The following key/object pairs may appear in the dictionary:

Exit code An integer showing the exit code to which the dictionary corresponds. This key/object pair is compulsory. You can even define an "error handling dictionary" even for scripts that execute successfully, that is with an exit code of 0 (zero). In this case Install will show the custom message that you specify using the `Text to display on error` key (below). Generally, however, it is best to let Install use the default behaviour by not providing an error handler for the exit code 0 (zero) case, in which case Install merely continues without notifying the user. If you do not define an error handling dictionary for a particular non-zero exit code and Install encounters that exit code, the error is considered to be fatal and installation terminates.

Text to display on error A string specifying the text to be displayed to the user if this error occurs. This key/object pair is optional. If you do not specify it, Install will supply a reasonable default.

Error type A string value indicating whether the error should be considered as: `Success` (installation continues; if `Text to display on error` is provided then the user is shown the text and presented with a "Continue" button); `Non-fatal` (noted as an error, but installation continues; once again, if `Text to display on error` is provided then the user is presented with a "Continue" button); `Can retry` (the user can choose to either retry or quit); and `Fatal` (installation halts). In the case of the `Can retry` error type, if the user does elect to retry after a script produces an error code, only the failed script is retried, not any other scripts that have been previously executed with success. This key/object pair is optional. If it is omitted then Install will assume that an exit code of zero implies `Success` and a non-zero exit code implies a `Fatal` error. Note that in all cases where the exit code is non-zero, a message will be displayed to the user and the user will be required to click a button ("Continue", "Retry" or "Quit" depending on the context). Where the exit code is zero, a message will only be displayed if you explicitly provide one, or mark the exit code as `Fatal` or `Can Retry` (although doing so is not recommended because in UNIX operating systems the exit code of zero is traditionally considered to signify success).

3.6.4 WOInstallerStepDomain

The `WOInstallerStepDomain` subclass should look like the following example:

```
<dict>
    <key>Step class</key>
    <string>WOInstallerStepDomain</string>
    <key>Step name</key>
    <string>Choose location</string>
</dict>
```

Alternatively, if you wish to override the default text displayed in the top third of the Install main window you can specify an RTF or RTFD document as illustrated:

```
<dict>
    <key>Document</key>
    <string>ChooseLocation</string>
    <key>Step class</key>
    <string>WOInstallerStepDomain</string>
    <key>Step name</key>
    <string>Choose location</string>
</dict>
```

The following key/object pairs may appear in the dictionary:

Document As per `WOInstallerStepReadMe`. This key/object pair is entirely optional. You may wish to specify a custom document here in order to change the font size or formatting, or to replace generic references in the default text such as "the software" or "home directory" to specifically refer to the name of the software to be installed and its exact destination on the disk.

Step class A string indicating the class of the step, in this case `WOInstallerStepDomain`. This key/object pair is compulsory. If you omit it Install will throw a `WOMalformedPlistException` and exit.

Step name As per `WOInstallerStepReadMe`.

As noted in Section 3.4 on page 20, it is possible to omit the WOInstallerStepDomain step entirely, in which case install will default to installing in the "User" domain (for the current user only). You can force install to default to the "Global" domain (and install for all users) by using the `Default installation domain` string.

3.6.5 WOInstallerStepCopyFiles

The `WOInstallerStepCopyFiles` step class is the most sophisticated of the Install classes. It comprises a number of substeps that are executed in this order, if present: `Preinstall kill processes`, `Preinstall trash files and folders`, `Remove login items`, `Run preflight scripts`, `Install files and folders`, `Run postflight scripts` and `Add login items`.

None of the substeps are compulsory, although clearly the `Install files and folders` substep is generally an essential requirement to produce and installer that does any useful work. Each of the substeps is identified by a key inside the `WOInstallerStepCopyFiles` dictionary. A sample skeleton dictionary which uses all of the substeps is:

```
<dict>
    <key>Document</key>
    <string>ReadMe</string>
    <key>Preinstall kill processes</key>
```

```
    <array>
        <!-- processes to quit before proceeding -->
    </array>
    <key>Preinstall trash files and folders</key>
    <dict>
        <!-- items to remove -->
    </dict>
    <key>Remove login items</key>
    <array>
        <!-- login items to remove -->
    </array>
    <key>Run preflight scripts</key>
    <array>
        <!-- scripts to run before installing -->
    </array>
    <key>Install files and folders</key>
    <array>
        <!-- files and folder to be installed -->
    </array>
    <key>Run postflight scripts</key>
    <array>
        <!-- scripts to run after installing -->
    </array>
    <key>Add login items</key>
    <array>
        <!-- login items to add -->
    </array>
    <key>Step class</key>
    <string>WOInstallerStepCopyFiles</string>
    <key>Step name</key>
    <string>Introduction</string>
</dict>
```

They following key/object pairs may appear in the dictionary:

Document As per `WOInstallerStepReadMe`. It should be noted that the amount of room available in the Install main window during a WOInstallerStepCopyFiles step is considerably reduced due to the presence of the progress bar and status indicator. As such, documents specified for this step should be significantly shorter and you should test that all the text remains visible even when the Instal main window is reduced to its minimum size.

Preinstall kill processes An array of processes to kill, as defined below.

Preinstall trash files and folders A dictionary defining files and folders to be trashed if installing to the User domain and/or installing to the Global domain. The format of this dictionary is defined on page 31.

Remove login items An array of login items to remove, as defined on page 32.

Run preflight scripts An array of preflight scripts to run before installing files and folders onto the target machine, as defined on page 33.

Install files and folders An array of files and folders to install, as defined on page 34.

Run postflight scripts An array of postflight scripts to run after installing files and folders onto the target machine, as defined on page 35.

Add login items An array of login items to add, as defined on page 35.

Step class A string indicating the class of the step, in this case `WOInstallerStepCopyFiles`. This key/object pair is compulsory. If you omit it Install will throw a `WOMalformedPlistException` and exit.

Step name As per `WOInstallerStepReadMe`.

The following paragraphs document the formats used to define the various possible substeps within a `WOInstallerStepCopyFiles` step. Note that each substep is defined as an array, with the exception of the `Preinstall trash files and folders` step which is defined as a dictionary.

Preinstall kill processes The following sample shows how to define a `Preinstall kill processes` array:

```
<array>
    <dict>
        <key>Identifier</key>
        <string>com.apple.systempreferences</string>
        <key>Method</key>
        <string>Apple Event</string>
    </dict>
    <dict>
        <!-- a second process to kill -->
    </dict>
    <dict>
        <!-- a third process to kill -->
    </dict>
</array>
```

Within the array, each process to be terminated is defined by a dictionary that may contain the following key/object pairs:

Identifier A string specifying the process to be quit in terms of its bundle identifier. This key/object pair is compulsory. If you omit it Install will not even attempt to terminate the target process.

Method A string specifying the method by which the target process shall be terminated. This key/object pair is compulsory. If you omit it Install will not be able to terminate the target process and will proceed with the installation anyway, which will probably not produce the intended results. The allowed values are:

Apple Event This method sends a "quit" Apple Event to the target process, thus providing the user with the opportunity to save or apply any unsaved changes prior to the process terminating. You should only specify this method for target processes which know how to handle the "quit" Apple Event (this is true of most foreground applications; it may not be true of faceless background processes or UNIX-style daemon processes).

SIGHUP This method sends the SIGHUP signal to the target process.

SIGTERM This method sends the SIGTERM signal to the target process.

Note that you can specify any number of processes to terminate by adding a dictionary for each process to the `Preinstall kill processes` array. The order in which processes will be terminated is not defined, because Install issues all of the termination commands at once and then waits to receive notification from the operating system that the processes in question have terminated.

The design is such that even if Install must wait for an extended interval (for example, while a user makes some final changes and then saves their work in an application targetted for termiantion) all of the other processes defined in the `Preinstall kill processes` array will effectively be terminated as soon as possible, without any bottlenecks caused by the need to wait for user intervention.

Preinstall trash files and folders Here is a sample `Preinstall trash files and folders` dictionary:

```
<dict>
    <key>User domain files</key>
    <array>
        <string>~/Library/Finalize</string>
        <string>~/Library/Caches/Finalize</string>
        <!-- additional items to remove -->
    </array>
    <key>Global domain files</key>
    <array>
        <string>/Library/Finalize</string>
        <string>/Library/Caches/Finalize</string>
        <!-- additional items to remove -->
    </array>
</dict>
```

The dictionary contains only two keys, `User domain files` and `Global domain files`, each defining a corresponding array of strings listing the files and folders which should be removed.

User domain files An array of strings identifying files that should be removed prior to installing in the User domain. This key/object pair is optional. If you omit it, Install will not remove any files prior to installing in the User domain.

Global domain files An array of strings identifying files that should be removed prior to installing in the Global domain. This key/object pair is optional. If you omit it, Install will not remove any files prior to installing in the Global domain (although it may remove files in the User domain if you listed any in the `User domain files` array; see below for more information).

It should be noted that because of the multi-user architecture of Mac OS X, when a user elects to install to the Global domain, Install will not only remove the items listed in the `Global domain files` array, but also those listed in the `User domain files` array. This is because in a correctly configured and implemented system, the operating system will favour resources that are more "local" with respect to the user. The following example illustrates: imagine that you wish to install a preference pane called `Example.prefPane` in the Global domain. In this case you should not only remove any pre-existing copies of `Example.prefPane` which might reside in the Global domain (in this case, in `/Library/PreferencePanes/`) but you should also remove any copies which might exist in the User domain (in `~/Library/PreferencePanes/`), because otherwise the more "local" copy in the user's home directory could in theory mask the newly installed copy in the system-wide directory.

Remove login items The following sample shows how to define a `Remove login items` array:

```
<array>
    <dict>
        <key>Absolute path</key>
        <string>/Applications/Example.app</string>
    </dict>
    <dict>
        <key>Item name</key>
        <string>Example.app</string>
        <key>User domain only</key>
        <true/>
    </dict>
    <dict>
        <!-- third login item to remove -->
    </dict>
</array>
```

The `Remove login items` array may contain any number of dictionaries, each defining one login item that should be removed. The only key/object pair in each dictionary is the following:

Absolute path A string containing the absolute path to the login item to be removed. Tilde expansion is not performed. Either this key/object pair or the alternative `Item name` pair is compulsory. If you omit both then Install will throw a `WOMalformedPlistException`. Absolute paths are recommended instead of mere item names (without paths) because there is less scope for accidental removal of a similarly named item. For example, if you wish to install a faceless background application called `Helper.app` which should launch at login, it is best to specify the path in absolute terms so that your choice of name will not clash with the that of an application made by another developer who had the same idea of using something called `Helper.app`. In keeping, when you add login items you should also always use absolute paths.

Item name A string containing the name of the item to be removed with no path information. You must specify either this key/object pair or the `Absolute path` pair discussed above. You should use this alternative with caution so as to avoid the unintended removal of items which may share the name as your login item. It is possible to specify both an `Item name` and an `Absolute path` but in general it is only necessary to specify one (for example, it would be redundant to specify `/Applications/Example.app` as an `Absolute path` if you also specified `Example.app` as an `Item name`).

User domain only An optional boolean value that you may specify to ensure that a given item is removed only in User domain installations. If you do not specify this key/object pair then Install uses a default value of False.

Global domain only An optional boolean value that you may specify to ensure that a given item is removed only in Global domain installations. If you do not specify this key/object pair then Install uses a default value of False. You may not specify both the `User domain only` and the `Global domain only` values as True.

You may wonder why the items to be added are defined as dictionaries and not as simple strings. The answer lies in the fact that defining things in this way, the dicitionaries used in setting up the `Remove login items` substep can be regarded as subsets of the dictionaries used later on in the `Add login items` substep. The former only ever contain one key/object pair, but the latter can make use of an additional key/object pair. See the description of Add login items on page 35 for more information.

Run preflight scripts The following sample shows how to define a `Run preflight scripts` array:

```
<array>
    <dict>
        <key>Script filename</key>
        <string>preflight.sh</string>
    </dict>
```

```
    <dict>
        <!-- second preflight script to run -->
    </dict>
    <dict>
        <!-- third preflight script to run -->
    </dict>
</array>
```

The `Run preflight scripts` array is simply an array of script definition dictionaries, as already described in *Script definition format* on page 25.

The example above shows the simplest script definition possible, because it only contains one key; the `Script filename` key. In this case Install would use the default values for `Shell` (`/bin/sh/`), and `Run script with root privileges` (false), and would execute the script without passing any arguments.

Install files and folders The following sample shows how to define a `Install files and folders` array:

```
<array>
    <dict>
        <key>Name</key>
        <string>Read Me 2004-01-01.pdf</string>
        <key>User domain location</key>
        <string>~/Desktop</string>
        <key>Global domain location</key>
        <string>/Users/Shared</string>
    </dict>
    <dict>
        <!-- second item to install -->
    </dict>
</array>
```

The `Install files and folders` array contains one or more dictionaries, each dictionary defining which file should be installed and where it should be placed depending on whether the install is to be in the Global domain or in the User domain. As already stated, the actual order in which files are copied is undefined, but the ordering of dictionaries within the configuration plist must correspond to the folder numbering used inside the `Resources/Files.localized/` folder inside the Install application bundle. In the example above, the first file (`Read Me 2004-01-01.pdf`) would be stored inside folder `0` (zero) and the second item would be stored in folder `1`. See Section 2.2 on page 14 for more information.

The dictionaries contained in the `Install files and folders` array may contain the following key/object pairs:

Name A string containing the exact name of the file or folder to be installed. This key/object pair is compulsory.

User domain location A string defining the folder into which the item should be installed when installing into the User domain. If the folder does not already exist Install will create it and any other intermediary folders as necessary. This key/object pair is compulsory.

Global domain location A string defining the folder into which the item should be installed when isntalling into the Global domain. Once again, if the folder does not already exist Install will create it. When installing into the Global domain items are copied using root privileges and so it becomes possible (and it is of course necessary) to write into shared locations such as /Library or elsewhere outside of the user's home folder. This key/object pair is only optional if your installer configuration does not contain a `WOInstallerStepDomain` step (because in this case Install defaults to installing in the User domain).

Run postflight scripts Postflight scripts are defined in exactly the same manner as preflight scripts (see *Run preflight scripts* above for more information). This is a sample of a `Run postflight scripts` array which contains two basic script definitions:

```
<array>
    <dict>
        <key>Script filename</key>
        <string>postflight-1.sh</string>
    </dict>
    <dict>
        <key>Script filename</key>
        <string>postflight-2.sh</string>
    </dict>
</array>
```

Add login items The following sample shows how to define a `Add login items` array:

```
<array>
    <dict>
        <key>Absolute path</key>
        <string>/Applications/Example.app</string>
        <key>Hide on launch</key>
        <true/>
    </dict>
    <dict>
        <!-- second login item to add -->
    </dict>
    <dict>
        <!-- third login item to add -->
    </dict>
</array>
```

The `Add login items` array may contain any number of dictionaries, each defining one login item that should be added. The dictionary is a superset of the dictionary used in the `Remove login items` substep. The dictionary may contain the following key/object pairs:

Absolute path As per the `Remove login items` substep.

Hide on launch A boolean indicating whether the added login item should be hidden on starting up. This key/object pair is optional. If you omit it, Install will use the default setting of false.

User domain only As per the `Remove login items` substep.

Global domain only As per the `Remove login items` substep.

Note that unlike the `Remove login items` substep, the is no Item name key used in the `Add login items` substep.

3.6.6 WOInstallerStepFinish

The `WOInstallerStepFinish` class is a subclass of `WOInstallerStepReadMe` and so inherits its capabilities and settings while adding some important additions. The basic structure of the `WOInstallerStepFinish` step dictionary is shown in the following skeleton dictionary:

```
<dict>
    <key>Document</key>
    <string>Finish</string>
    <key>Show background</key>
    <false/>
    <key>Show scrollbar</key>
    <false/>
    <key>Step class</key>
    <string>WOInstallerStepFinish</string>
    <key>Step name</key>
    <string>Finish</string>
    <key>Primary button action</key>
    <dict>
        <!-- the primary button action -->
    </dict>
    <dict>
        <!-- the secondary button action -->
    </dict>
</dict>
```

The WOInstallerStepFinish step dictionary may contain the following key/object pairs:

Document As per `WOInstallerStepReadMe`.

Show background As per `WOInstallerStepReadMe`.

Show scrollbar As per `WOInstallerStepReadMe`.

Step class A string indicating the class of the step, in this case `WOInstallerStepFinish`. This key/object pair is compulsory. If you omit it Install will throw a `WOMalformedPlistException` and exit.

Step name As per `WOInstallerStepReadMe`.

Primary button action An optional dictionary that defines an action to be performed when the user clicks the right-most button (usually labelled "Continue") in the Install main window.

Secondary button action An optional dictionary that defines an action to be performed when the user clicks the left-most button (the "Uninstall" button) in the Install main window.

You may specify a custom `Primary button action`, a custom `Secondary button action`, or both. Where you do not specify an custom action the button in question will remain inactive and have no effect. In all cases there is a "Quit" button in the Install main window that the user can use upon finishing the installation. The follow section describes how to define custom button actions in the configuration plist.

Assigning a primary and secondary button action The custom button action format is identical for both the `Primary button action` and `Secondary button action` keys. In each action you may specify either a single shell script to run (not an array of scripts), or an item (or items) on the filesystem to open (optionally specifying an application with wich to perform the open operation). You may not specify both a shell script and a `Launch` directive within the same button action definition.

Launch-based custom button actions Below is a sample button action dictionary that contains a `Launch` directive:

```
<dict>
    <key>Button text</key>
    <string>Launch</string>
    <key>Launch</key>
    <array>
        <dict>
            <key>Open</key>
            <string>Manual.pdf</string>
            <key>Global domain location</key>
            <string>/Users/Shared</string>
            <key>User domain location</key>
```

```
            <string>~/Desktop</string>
            <key>With</key>
            <string>/Applications/Preview.app</string>
        </dict>
    </array>
</dict>
```

The `Launch`-based custom button action dictionary must contain the following key/object pairs:

Button text A string specifying the text to appear inside the button. This key/object pair is compulsory. If you do not define it, Install will hide or otherwise disable the button.

Launch An array of dictionaries, each dictionary defining an item that is to be launched. The first three key/object pairs to be used in each dictionary match those used in the `Install files and folders` substep of the `WOInstallerStepCopyFiles` step. These keys are `Name`, `Global domain location` and `User domain location`. The `Name` specifies the name of the item to be opened. `Global domain location` specifies the folder in which the item resides (if installed into the Global domain) and the `User domain location` specifies the folder in which the item resides if installed into the User domain. A fourth, optional, key is the `With` key, which is used to specify an application with which to open the file. If you omit the `With` key then Install will open the file as if you had double-clicked it in the Finder.

Script-based custom button actions The following is an alternative example of a button action dictionary, this one using a `Script` directive instead of a `Launch` directive:

```
<dict>
    <key>Button text</key>
    <string>Script</string>
    <key>Run script</key>
    <dict>
        <!-- a script defintion -->
    </dict>
</dict>
```

The `Script`-based custom button action dictionary must contain the following key/object pairs:

Button text A string specifying the text to appear inside the button. This key/object pair is compulsory. If you do not define it, Install will hide or otherwise disable the button.

Run script The `Run script` dictionary is simply a script definition dictionary, as already described in *Script definition format* on page 25. You may specify an error handling array (see *Script error handling* on page 26).

3.7 The Uninstall step

The `WOInstallerStepUninstall` step is an independent step which never appears within a normal sequence of `WOInstallerStep` steps, but is triggered when the following conditions are true:

- The `Uninstall step` dictionary is present and correctly specified in the configuration plist file.
- The `Install check` dictionary is also present and correctly specified in the configuration plist file, so that Install knows how to determine whether a pre-existing installation already exists on the target machine.
- The first step in the configuration file is of class `WOInstallerStepReadMe` (the "Uninstall" button will only be shown if this is true).
- Files or folders which indicate the existence of a pre-existing installation are found on the target machine at run time.

Like the `WOInstallerStepCopyFiles` step, the `WOInstallerStepUninstall` step comprises several, optional substeps, which will be executed in the following order if present: `Uninstall kill processes`, `Run preflight scripts`, `Uninstall trash files and folders`, `Remove login items` and `Run postflight scripts`. Any or all of the substeps can be omitted from the configuration if you deem them not to be necessary, but to be a meaningful uninstallation then it is usuall necessary to define at least an `Uninstall trash files and folders` substep, as this is the substep responsible for deleting files or folders, or moving them to the Trash.

The configuration of each of these substeps is discussed below, immediately following this example of a skeleton `WOInstallerStepUninstall` dictionary. Note that each of the substeps is specified in terms of an array, and that any of the substeps could be safely omitted if it is not required:

```
<dict>
    <key>Uninstall kill processes</key>
    <array>
        <!-- processes to be terminated -->
    </array>
    <key>Run preflight scripts</key>
    <array>
        <!-- scripts to be executed -->
    </array>
    <key>Uninstall trash files and folders</key>
    <array>
        <!-- items to delete/move to the Trash -->
    </array>
    <key>Remove login items</key>
    <array>
        <!-- login items to be removed -->
```

```
    </array>
    <key>Run postflight scripts</key>
    <array>
        <!-- scripts to be executed -->
    </array>
</dict>
```

3.7.1 Uninstall kill processes

The following is a sample which shows how to define an `Uninstall kill processes` array. The array can contain any number of dictionaries, each dictionary defining the process to be targetted for termination and the method to be used to terminate it. The format is exactly the same as that used in the `Preinstall kill processes` substep of the `WOInstallerStepCopyFiles` step (see *Preinstall kill processes* on page 30 for more details).

```
<array>
    <dict>
        <key>Identifier</key>
        <string>com.apple.systempreferences</string>
        <key>Method</key>
        <string>Apple Event</string>
     </dict>
</array>
```

3.7.2 Run preflight scripts

The following is a sample showing how to define a `Run preflight scripts` array. The format is identical to that of the `Run preflight scripts` array used in the `WOInstallerStepCopyFiles` step (see page 33 for more information).

```
<array>
    <dict>
        <key>Script filename</key>
        <string>preflight-sample.sh</string>
        <key>Run script with root privileges</key>
        <true/>
    </dict>
    <dict>
        <key>Script filename</key>
        <string>preflight-two.sh</string>
    </dict>
</array>
```

3.7.3 Uninstall trash files and folders

Unlike the other substeps of `WOInstallerStepUninstall` which are defined in the same manner as their corresponding substeps in `WOInstallerStepCopyFiles`,

this `Uninstall trash files and folders` substep array has a unique format, as illustrated in the following example:

```
<array>
    <dict>
        <key>Item</key>
        <string>/Users/Shared/ABCD Manual.pdf</string>
        <key>Remove by default</key>
        <true/>
        <key>Administrator privileges required</key>
        <true/>
    </dict>
    <dict>
        <!-- second item to remove -->
    </dict>
</array>
```

As shown above, the `Uninstall trash files and folders` array should contain one or more dictionaries, each dictionary defining the item that should be removed. Each dictionary may contain the following key/object pairs:

Item A string containing the full path of the object to be removed. This key/object pair is compulsory.

Remove by default A boolean indicating whether or not the item should be removed in a default uninstallation procedure. If true, the checkbox next to the item in the listing of items to be removed will be pre-selected. If false, the checkbox will not be pre-selected. This key/object pair is optional and if you elect not to supply it then Install will assume a default value of false.

Administrator privileges required A boolean indicating whether an item will require a valid administrator user name and password to remove. Generally such items will be those that you have previously installed into the Global domain using root privileges. When the boolean is true, the item will be displayed with a small padlock icon in the listing of items to be removed. This key/object pair is optional and in its absence Install assumes a default value of false.

3.7.4 Remove login items

The following is a sample showing how to define a `Remove login items` array. The format is identical to that of the `Remove login items` substep of the `WOInstallerStepCopyFiles` class (see page 32 for a full reference).

```
<array>
    <dict>
        <key>Absolute path</key>
        <string>/Applications/Example.app</string>
    </dict>
</array>
```

3.7.5 Run postflight scripts

The following is a sample showing how to define a Run postflight scripts array. Once again, the format exactly matchs that of the Run postflight scripts substep of the WOInstallerStepCopyFiles class (fully documented on page 35).

```
<array>
    <dict>
        <key>Script filename</key>
        <string>postflight.sh</string>
    </dict>
</array>
```

4 Localization using the InstallConfig.strings file

A number of user-visible strings appear in the `InstallConfig.plist` configuration file described in Section 3. If you wish to provide a fully localized installer then you must localize these strings. Your configuration may contain the following such strings:

- "`Step name`" strings, which appear in the Install user interface in the list view on the left hand side of the main window;
- "`Text to display on error`" strings, which you may (optionally) supply if your installer runs any shell scripts;
- Custom "`Button text`" strings, which you can specify in your `WOInstallerStepFinish` step.

There are two possible approaches to the localization of these strings; either you can localize the `InstallConfig.plist` file itself, providing a different file for each language variant, or you can provide a separate, `InstallConfig.strings` file, which contains only the user-visible strings from the configuration file.

The advantage of using the latter method is that it simplifies the task of localization for translators. They no longer have to deal with a complicated configuration file which may be difficult for them to understand; and there is little or no risk of them editing a part of the file which they should not edit, nor of overlooking something which they should.

The binary distribution of Install itself shows how to employ this later method. If you look inside the Install application bundle itself, or if you examine the sample files included with the distribution, you will see that the following points apply and the result is a Spanish localization of the user-visible strings in the configuration plist file:

- The `InstallConfig.plist` file is not provided in localized variants; rather, a single global copy is provided and stored in the `Resources` folder of the Install bundle and not in a language-specific subdirectory.

- Each user-visible string in the configuration file (whether it be a "`Step name`", "`Text to display on error`" or "`Button text`" string) has been identified and is used as a key in a language-specific `InstallConfig.strings` file stored in the `Spanish.lproj` subdirectory of the `Resources` folder.
- For each key in the `InstallConfig.strings` file (on the left hand side of the equals sign), a Spanish equivalent is provided (on the right hand side); the formatting of this file is identical to that of a standard `Localizable.strings` file.
- Like all `strings` files, the `InstallConfig.strings` file is saved using UTF-16 encoding.
- The keys in the `InstallConfig.strings` file exactly match (including case sensitivity) the equivalent user-visible strings in the `InstallConfig.plist` file.
- It is not necessary to provide an English language variant of the `InstallConfig.strings` file, because the default strings already appear in the `InstallConfig.plist` file.
- In the event that a localized version of the `InstallConfig.strings` file is not present, Install will fall back and use the default, English versions.

5 Advanced configuration

This section of the manual provides some additional information that may be of use to developers wishing to embed shell scripts within their custom versions of the Install application and run those scripts from within `WOInstallerStepShell`, `WOInstallerStepCopyFiles` or `WOInstallerStepUninstall` steps. It also discusses the use of the plutil command line tool to verify plist integrity.

5.1 Using environment variables in shell scripts

5.1.1 Inherited environment variables

A full set of environment variables is passed on before executing any script. The environment variables are inherited from the environement of the Install application itself. This means, for example, that if user `johnsmith` launches Install, then a number of environment variables indicating his username (`johnsmith`) and his home directory (`/Users/johnsmith/`) will be set and can be utilized by the shell script.

Note that this is true even when the a script is run with root privileges. In the case of the example above, a script running with root privileges would see the username as `johnsmith` and not as `root` because the script inherits its environment variables from the parent process, which is Install itself and which was launched by user `johnsmith`. When a script is running with elevated privileges, only the script

Table 1: Typical environment variables passed to shell scripts

Variable	Example value
BASH	/bin/sh
EUID	501
HOME	/Users/wincent
HOSTNAME	sol.local
HOSTTYPE	powerpc
MACHTYPE	powerpc-apple-darwin7.0
PPID	8498
PWD	/Install.app/Contents/Resources
SHELL	/bin/bash
SHLVL	1
TERM	dumb
UID	501
USER	wincent

has those privileges; Install itself continues to run with its previous level of unelevated privileges.

Table 1 shows a subset of some of the typical environment variables that you could expect to have set in the environment of your shell script, along with some sample values for those variables. These particular values were taken from my production machine, running Mac OS X 10.3.2. It is reasonable to expect that you might obtain other variables and other variables under different systems, so you should test to be sure that you have access to the environment variables you require on the target system onto which you plan to deploy your software.

5.1.2 Custom environment variables

In addition to the standard environement variables inherited from the environment of the Install application, Install adds or modifies a number of "custom environment variables".

SCRIPTDIRECTORY Install sets this variable to indicate the absolute path of the directory in which the script resides. This will usually be the `Resources` folder inside the Install application bundles, but it could also be a localized lproj directory with `Resources` if you have chosen to provide a language-specific version of the script. Please note the value of `SCRIPTDIRECTORY` does not necessarily coincide with the `PWD` (present working directory) environment variable, due to differences in the APIs used to launch scripts with or without root privileges[5]. What this means in practical terms it that you should always

[5] In the case of non-privileged scripts, before launching Install sets the current working directory to be the directory in which the script currently resides. In the case of privileged scripts, however, this is not possible because the system will always reset the present working directory to match the path that was current at the time Install itself was launched (for example, if Install was launched by double-clicking in the Finder, then `PWD` will be reset to "/" every time a script is launched. If it was

begin your scripts with a command such as cd to change directory to the SCRIPTDIRECTORY if you want to make use of relative paths inside the script.

INSTALLATIONDOMAIN Install sets this variable to indicate whether the installation is to take place (or has taken place) in the Global domain or the User domain. The possible values of the variable are "Global" and "User". Note that if you do not provide a WOInstallerStepDomain step, or if you run a script prior to that step, then the value of the INSTALLATIONDOMAIN environment variable will be the default value ("User"). Also note that this value has no meaning for the WOInstallerStepUninstall step because users do not elect an uninstallation domain; rather they choose which files shall be removed using checkboxes for items in the list.

5.2 Redirecting output to the console via STDERR

Any scripts you choose to run will inherit STDERR from Install. If you wish to write output to the console (which will be visible to users if they chose to run Apple's Console application, or otherwise inspect the console.log file (stored in /Library/Logs/Console/ on Mac OS X 10.3) then the simplest way is just to direct it through STDERR. The following example script shows how this could be done from within a Bash script:

```
#!/bin/bash

# echo the environment variables to STDERR
set > /dev/stderr

exit 0
```

This example shows one way of directing output through STDERR when using Perl:

```
#!/usr/bin/perl

print STDERR "This message goes to the console\n";

exit 0;
```

5.3 Using plutil to check the configuration plist

You can use Apple's plutil command line tool to check a plist if you suspect that it is faulty. If you have a fault plist then Install will be unable to launch. To check a plist you would use the Terminal to enter a command such as the following:

```
/usr/bin/plutil InstallConfig.plist
```

launched by typing a command in the Terminal such as Install.app/Contents/MacOS/Install then the PWD will match the setting of the parent shell in which the command was typed).

In response, `plutil` scans the file and parses it to see if it is a valid plist. It can provide helpful diagnostic information that can quickly point out what specific problems might exist in your configuration file. This is an example of the information provided by the tool:

```
InstallConfig.plist:
XML parser error:
       Encountered unknown tag ditc on line 5
Old-style plist parser error:
       Expected terminating '>' for data at line 1
```

Note that the `plutil` utility only checks to see if your plist is a validly formatted XML plist document. The utility has no knowledge of Install's application-specific requirements (such as which keys are compulsory and which keys are optional) and so cannot help you to troubleshoot configuration files which are valid XML plists but which nevertheless deviate from the requirements prescribed in this manual. For more information about the capabilities of `plutil`, consult the `plutil` man page.

6 Preparing your own installer

When you obtain a Binary-only license for Install you receive a disk image containing a Developers Kit. The kit includes a copy of Install which you can run and which will place copies of Install itself, the documentation, the license, and a collection of templates and samples on your disk.

These items are placed in the following locations if you elect to install Install for all users on your system:

`/Developer/Applications/Install/Install.app` A "clean" (unconfigured) copy of the Install application bundle.

`/Developer/Applications/Install/Documenation/` Documentation relevant to the Binary-only Install license, including a copy of the license itself, the Install Binary-only manual (the document you are reading now), and a bookmark for the Install website.

`/Developer/Applications/Install/Templates/` A collection of template files (plist files, shell scripts and RTF documents) that you can use as a starting point for creating your own installer.

`/Developer/Applications/Install/Samples/` A collection of sample files (plist files, shell scripts and RTF documents) that model ways in which Install can be configured.

You may also elect to install for the current user only, in which case the items will be placed in the `~/Developer/` folder rooted in your home directory.

6.1 Creating a copy of the Install application bundle

When preparing an installer it is recommended that you work within a temporary working area on your disk. For the purposes of illustration, the examples used in this section through to Section 6.9 will assume that the working area is a folder inside your home directory at `~/work/`, and that you have installed the Developers Kit in `/Developer/`.

Your first step should be to make a clean copy of the Install application bundle on which to work, by making a copy of the master version of `Install.app` stored in `/Developer/Applications/Install/` and placing it inside your work area. This could be done by using "drag-and-drop" in the Finder, or by using the `CpMac` command line tool as follows:

```
/Developer/Tools/CpMac -r \
  /Developer/Applications/Install/Install.app \
  ~/work/
```

This command is broken up across three lines for readability. Note that `CpMac` is used instead of the conventional UNIX `cp` command, because the former can preserve resource forks and other Macintosh-specific information (see the `CpMac` man page for more information). Although Install itself does not by default include any files with resource forks, it is a good to establish a habit of using `CpMac` for copying installers, because you may later add files inside the installer bundle which do contain resource forks (custom folder icons are a common example of files that need resource forks on Mac OS X).

6.2 Placing items to be installed inside the bundle

Open the working copy of the Install application bundle by choosing "Show package contents" from its contextual menu in the Finder, and navigate to `Install.app/Contents/Resources/`. You could also achieve this by issuing the following command in the Terminal:

```
/usr/bin/open ~/work/Install.app/Contents/Resources
```

As documented in Section 2.2, the items to be installed should be placed inside numbered subfolders inside the `Files.localized` directory. An example set of items might be:

- `Install.app/Contents/Resources/Files.localized/0/File 1`
- `Install.app/Contents/Resources/Files.localized/1/File 2`
- `Install.app/Contents/Resources/Files.localized/2/File 3`
- `Install.app/Contents/Resources/Files.localized/3/Folder 1`
- `Install.app/Contents/Resources/Files.localized/4/Folder 2`

You can only place a single item inside each numbered subfolder, but it can be a file or a folder containing other times. Sometimes you will wish to group items together into a folder if they are going to eventually reside together as a group inside that folder on the target marchine. At other times you might wish to split these files up so that they can be installed or removed separately, even if their ultimate destination will be inside the same folder on the target machine.

6.3 Planning the installation steps

Your next task should be to plan the steps that will comprise the installation. In most cases a `WOInstallerStepCopyFiles` step will lie at the heart of the installation. It is generally advisiable to start off the installer with a `WOInstallerStepReadMe` step and finish it with a `WOInstallerStepFinish` step.

If you require the user to agree to a license before installing you should insert a `WOInstallerStepLicense` step inbetween your opening `WOInstallerStepReadMe` step and your `WOInstallerStepCopyFiles` step.

If you want to offer the user a choice between installing for all users on their machine or installing for the current user only then you must insert a `WOInstallerStepDomain` step at some point prior to the `WOInstallerCopyFiles` step.

To incorporate advanced functionality you may also choose to insert one or more `WOInstallerStepShell` steps in which to run shell scripts.

You must also decide whether you wish to offer uninstallation functionality in your installer. If you do, then you will need to define a `WOInstallerStepUninstall` step (to perform the uninstallation) and an `Install check` array (to check for a pre-existing installation).

6.4 Preparing documents

Once you have decided the number, type and order of the installation steps, you will need to prepare appropriate RTFD or RTF documents for each step. All `WOInstallerStep` steps require an accompanying document to display to the user in the Install main window during that step, with the exception of `WOInstallerStepDomain` which will use a reasonable default if a document is not supplied.

All such documents should carry an appropriate "`.rtfd`" or "`.rtf`" file extension to indicate their type.

If you plan to configure any of your installation steps such that the will display the documents without a scroll bar, then you should ensure that you make the corresponding documents short enough that they can entirely fit within the Install main window, even when the window is displayed at its minimum size. You should verify this in the testing phase.

Once your documents are ready you should place them inside the application bundle in the `Install.app/Contents/Resources/` folder. If you have prepared localized variations of the documents in the corresponding lproj folders (for example, `Install.app/Contents/Resources/English.lproj/`, or `Install.app/Contents/Resources/Spanish.lproj/`).

6.5 Preparing shell scripts

If you are sufficiently familiar with shell scripting you may decide to prepare some scripts for execution during the installation. It is beyond the scope of this document to teach how to write shell scripts, but for users already familiar with scripting Section 5 does offer some information about environment variables passed to scripts and redirecting output to the console.

The completed scripts do not need to have their execute permissions set, because Install will run them by invoking a shell and passing the script path as an argument to the shell.

You should place the completed scripts inside the application bundle at `Install.app/Contents/Resources/`, or if you have prepared localized variations, in the corresponding lproj folders.

6.6 Creating the configuration plist file

Having placed the items to be installed, the documents, and any necessary shell scripts in the Install application bundle, you should prepare an appropriate configuration plist named `InstallConfig.plist`, and place it inside the Install application bundle in the `Resources/` folder. Once again, you are free to prepare localized versions and place them in the appropriate lproj subfolder of the `Resources/` folder.

Section 3 starting on page 16 describes in detail how to configure the plist. You may elect to use one or more of the templates installed at `/Developer/Applications/Install/Templates/` or to produce a configuration file from scratch. You can use a text editor such as Apple's Text Edit (included with Mac OS X), or BBEdit from Bare Bones Software[6] (a separate commercial product). If your editor allows you to specify the encoding of saved files, you should specify UTF-8 (the default for Apple plist files).

Another alternative is to use Apple's Property List Editor (included with the Mac OS X developer tools). The advantage of using the Property List Editor as it makes it very difficult to accidentally introduce syntax errors in the document which would make it an invalid XML plist.

Key points to bear in mind while creating a configuration plist file:

- Ensure that every step (except `WOInstallerStepDomain` steps) has an associated document defined.
- When specifying documents, write only the base part of the filename (without an extension).
- If specifying shell scripts, specify the full filename (including the extension).
- Ensure that every item defined in the `Installation steps` has an appropriate `Step name` defined (which will appear in the steps listing on the left-hand side of the Install main window.

[6] See: `http://www.barebones.com/products/bbedit/index.shtml`

- Likewise, ensure that every item defined in `Installation steps` has the correct `Step class` defined.
- If you include an `Uninstall step`, remember that you must also include an `Install check` array.

6.7 Localize user-visible strings in the configuration file

As described in Section 4 on page 42, you can provide localizations of the user-visible strings in your configuration file. These strings will include things like step names (which appear on the left-hand-side of the Install main window), custom error messages (which you may choose to supply if your installer runs shell scripts) and custom button text (which you can display in your `WOInstallerStepFinish` step).

There are two ways to provide these localizations: you may provide a language-specific variant of the `InstallConfig.plist` file itself, or you can create an `InstallConfig.strings` file which contains only the user-visible strings. This latter option may be easier for translators, because they do not need to think about which parts of the configuration file to localize and which parts should be left untouched. A full description of how to produce such a file appears in Section 4.

6.8 Testing

The next and final phase is to test that your customized Install application works as you expect. Run your installer from the command line as follows:

```
cd ~/work
Install.app/Contents/MacOS/Install
```

By invoking the application from the command line you will be able to readily see any messages that Install may output to the console. This may be helpful if you need to diagnose problems. You can also easily find out the exit status once Install finishes running by immediately typing:

```
echo $?
```

If Install returns a non-zero exit code then this indicates an abnormal exit. If Install does not even launch then you may have provided an invalid plist file. See Section 5.3 for advice on using Apple's `plutil` utility to check plist validity.

In addition to watching the console output, you can also run Install and select "Show log" from the Install "File" menu to see how Install reports its own activity.

As with any software, you should rigorously test your Install application before deploying it. The following are common-sense some suggestions for testing:

- If you specify different sets of installation instructions for different versions of Mac OS X, test your installer works as expected on those versions of the OS.

- If you define an `Install check` array, test that the "Uninstall" button appears whenever you first launch the installer on a system with a pre-existing installation. Conversely, the button should not appear if there is no pre-existing installation.
- If you allow the user to install the Global domain or the User domain, test installing to both domains.
- If you define an `Uninstall step`, test that it works as expected. This means that you should test to see if you can uninstall software previously installed to the Global domain, as well as testing to see if you can uninstall software previously installed to the User domain (if you allow the user to choose domains).
- Test installing onto a "clean" system (without a pre-existing install), and test installing onto a "non-clean" system (overwriting a previous install).
- If you define a `WOInstallerStepFinish` step with custom button actions, test that the button actions work as expected. Test after installing to the Global domain, and also after installing to the User domain (if you allow the user to choose domains).
- If you provide localized variations of any files, perform the tests while running the installer under each different language that you support.

The above is not an exhaustive list, but it gives some indication of the key permutations that should be covered in a minimal testing run. Remember that because Install is a software deployment tool specifically designed to perform operations such as copying and deleting files, which are often irreversible operations, extreme care should be taken to ensure that your configuration performs exactly as you intend.

6.9 Packaging

Once you have completed your testing you need only package your installer for distribution. Common packaging methods include Disk Images and Zip archives. It is recommended that you only use the later format if your software is destined for users of Mac OS X 10.3 or above, because previous versions of the OS do not offer an in-built means for decompressing those archives.

Although the phases described in this section are all that is required to produce a working installer, you may also wish to see Section 7 below, which covers some additional, optional customizations you may wish to make to your installer before packaging it for final distribution. This customizations include providing a custom icon file, renaming the application bundle, or adding your own copyright notices.

7 Customizing the Install application bundle

This section discusses the modifications you are pemitted to make to the Install application bundle under the Binary-only license. The modifications discussed here

are entirely optional and are not required to produce a functioning installer. For information on the kind of configuration essential to produce a working installer, consult Section 6.

Under the Install Binary-only license agreement you are granted the right to modify the Install application bundle within certain limits. The main limitation is that you are not permitted to strip any copyright notices from Install. Install itself, both in binary and source code form, is copyright (c) 2003-2004 by Wincent Colaiuta, all rights reserved. You retain authorship of and copyright to any permitted modications that you make to Install. What this means in practical terms is that.

- You may rename the Install application bundle itself. For example, you may wish to rename it from `Install.app` to `Install Our Product.app`.
- If you choose to rename the bundle you are not obliged to maintain the word "Install" in the title.
- You may add your own copyright atttribution to the `CFBundleGetInfoString` and `NSHumanReadableCopyright` fields in the Install `InfoPlist.strings` file (or files, depending on whether localized variants exist), but you may not remove or edit the existing copyright attributions to Wincent Colaiuta in those fields. For example, you may wish to add you own copyright to the original string ("Install version 1.1, Copyright 2003-2004 Wincent Colaiuta.") so that it reads, "Install version 1.1, Copyright 2003-2004 Wincent Colaiuta. Product Name version 2.0, Copyright 2004 Our Company." Any such change will be reflected in the version and copyright information shown about the installer in the Finder and/or in the "About Install" window.
- You may not change the `CFBundleName` or the `CFBundleShortVersionString` fields in the `InfoPlist.strings` file, nor may you alter or remove the `CFBundleSignature` field in the `Info.plist` file.
- You may replace the Install icon file with a custom icon of your own.
- You may not edit the Install `MainMenu.nib` file or otherwise make alterations which have the effect of disabling, removing or altering the "Install website..." Menu Item in the Install Menu.
- You may edit, and optionally convert to RTFD format, the `Credits.rtf` file that is used to construct the "About Install" window when the corresponding Menu Item is selected; but you must preserve the text in the file which attributes "Installer engineering and design" to "Wincent Colaiuta". You may not remove the file unless you replace it with an equivalent RTFD file containing the same attribution. You may add additional lines to the file which attribute authorship for your own products to the corresponding authors, and if you have made modifications to Install as permitted under the license (in the case of the Binary-only license permitted modifications are generally limited to configuration and the provision of resources such as documents, shell scripts

and other materials) then you may add a line such as "Installer modifications" or "Installer customization" attributing authorship of the modifications or customizaton to, for example, "Our Company".

- You are not required to provide an attribution to Wincent in your product documentation, in-program text, supporting materials or website, although any such reference you might make to Wincent and to the Install website (install.wincent.com) would be appreciated.
- You may not modify nor reverse engineer the binary data of the Install executable itself (stored inside the bundle as `Install.app/Contents/MacOS/Install`),

For further information about your rights and obligations under the license, please consult the license.

Index