Geoff Lee
Edinburgh College of Art
University of Edinburgh

MacAD.UK

macadmins.slack: @geoffklee
https://github.com/UoE-macOS

Agenda

- ☐ (Re)packaging and installer technologies
- ☐ Vendor vs Sysadmin
- ☐ The quest for reproducibility
- ☐ Deep dive - web debugging proxies & python
- ☐ Some brief case studies

(Re)Packaging

Autopkg: my hero

AutoPkg is an automation framework for macOS software packaging and distribution, oriented towards the tasks one would normally perform manually to prepare third-party software for mass deployment to managed clients.

These tasks typically involve at least several of the following steps:

- ☐ downloading an application and/or updates for it, usually via a web browser
- ☐ extracting them from a multitude of archive formats
- ☐ adding site-specific configuration
- ☐ adding sane versioning information
- ☐ "fixing" poorly-written installer scripts

Source: https://github.com/autopkg/autopkg

Vendor imagines:

- Single user sitting on the console during installation
- Is only user of machine
- Is probably an admin user
- Wants most bleeding-edge possible version of product
- At all times
- Available to click license agreements and enter codes on each machine
- Is a rapacious pirate

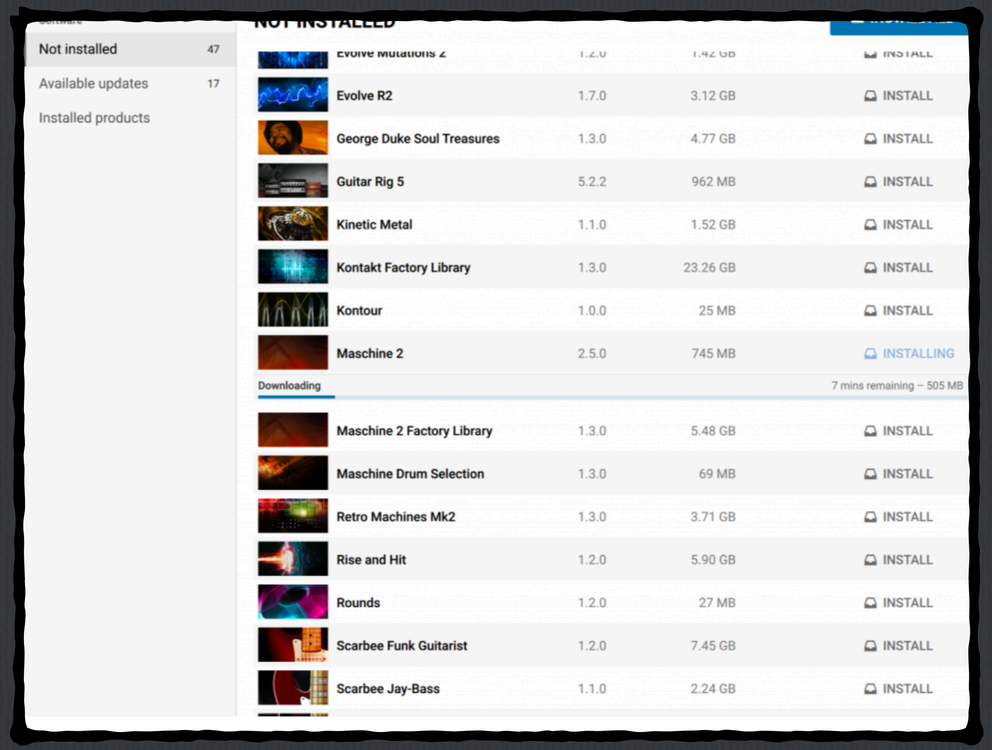
macOS lab environment:

- Unattended installs with nobody on console
- Multi-user machines
- No admins
- Want a specific version, not necessarily the latest
- No updates during work
- I need to license multiple machines at once, at install time
- Just wants to do work

Cross-platform installer tech	Supports silent installation?
VISE X	NO
InstallAnywhere	YES
QT Installer Framework	NOPE: https://bugreports.qt.io/browse/QTIFW-166
InstallBuilder	YES
Electron	NO

‘Download Manager’

- ‘Opaque’ application
- Serves as installer and update manager
- Gives vendors a ‘live’ platform to talk to users
- Often cross-platform
- This may be all we get



Snapshot packages:

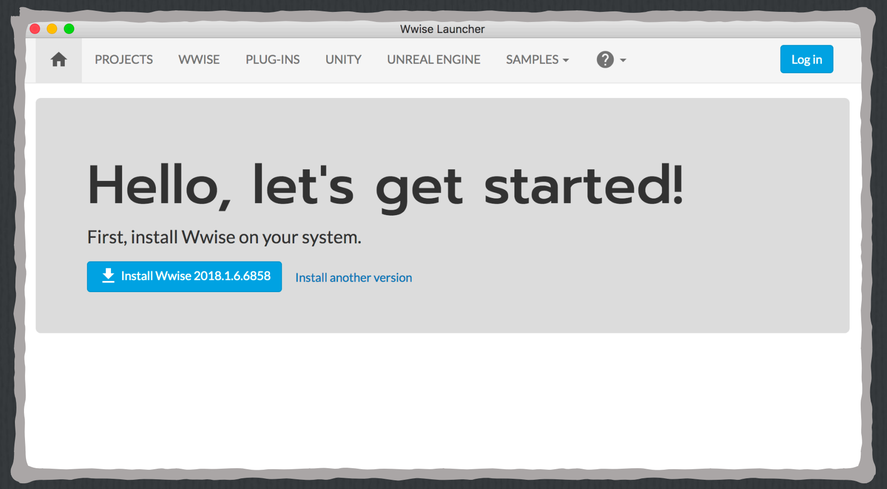
- ☐ Only enable file-level changes
- ☐ Break installation databases
- ☐ Break licensing databases
- ☐ Generation can’t be automated

radmind: you served us well

What's an admin to do?

RECAP:

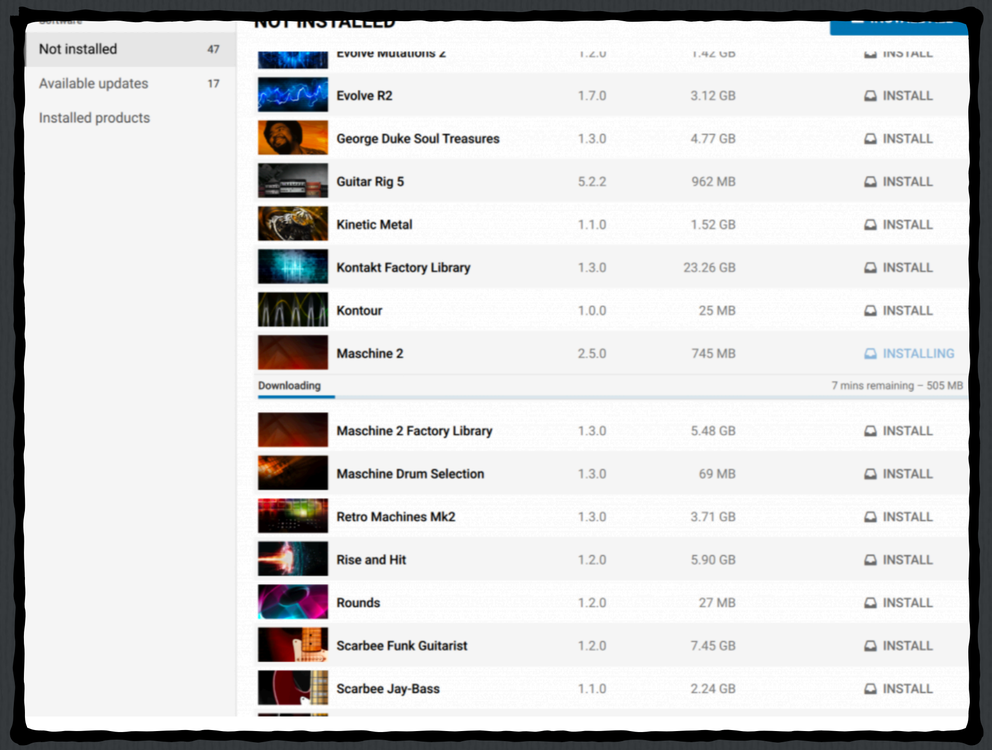
- ☐ We want reproducible, unattended installs.
- ☐ We want to be able to build a new package on demand
- ☐ And we need the install to proceed silently



Wwise Launcher

‘Download Managers’

- ‘Opaque’ application, but…
- Talks to vendor’s servers
- Retrieve files to install in archive form
- Unpack and (silently!) install them



Can we impersonate the installer…

…and talk directly to the Vendor’s infrastructure?

A toolkit for installer impersonation

Web debugging proxy

Scripting language

Determination

“Article 6
Decompilation:

1. The authorization of the rightholder shall not be required where reproduction of the code and translation of its form within the meaning of Article 4 (a) and (b) are indispensable to obtain the information necessary to achieve the interoperability of an independently created computer program with other programs, provided that the following conditions are met:

(a) these acts are performed by the licensee or another person having a right to use a copy of a program, or on their behalf by a person authorized to do so;

(b) the information necessary to achieve interoperability has not previously been readily available to the persons referred to in subparagraph (a); and

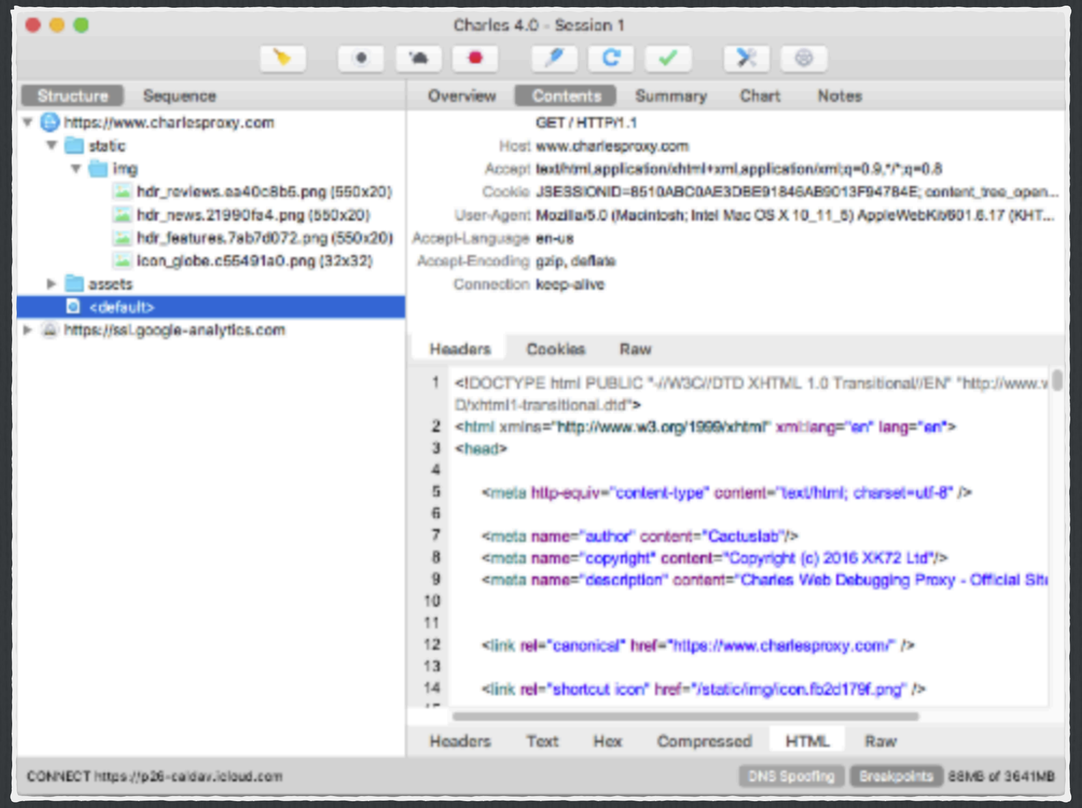
(c) these acts are confined to the parts of the original program which are necessary to achieve interoperability.”

COUNCIL DIRECTIVE of 14 May 1991 on the legal protection of computer programs (91/250/EEC) (OJ L 122, 17.5.1991, p. 42)

I.A.N.A.L

Web debugging proxy

- ☐ Installer will be talking to an API
- ☐ Probably (hopefully!) over HTTPS
- ☐ MITM ourselves…
- ☐ Now we can listen in!



Charles proxy

Demo - getting started with Charles

What’s going on?

Installer: `Hi Server, what stuff you got?`

Server: `Here’s a big blob of json/xml`

Installer: `*Parses blob, checks against installed stuff*`

Installer: `OK, gimme SuperCatExplorer-2.1.1.pkg`

Server: `Here y’go`

Installer: `*Does installation*`

We have 4 tasks

- ☐ Ask server for available stuff
- ☐ Parse list of stuff
- ☐ Download installable artefacts
- ☐ Install (or otherwise process) them

Image: hannaford on Flickr

```python
# We are going to demonstrate three modules:
import urllib2
import json
import xml.etree.ElementTree as ElementTree

# The urllib2 library contains everything you need to make HTTP/s requests
# Lots of people prefer the Requests module, but Requests isn't part of the
# standard library - urllib2 is even available in this Jupyter notebook :-)

# In its most basic form, we can issue a GET request to retrieve a
# URL by passing the url as a string to the urllib2.urlopen() method:

url = 'https://www.charlesproxy.com/latest.do'

# urlopen() returns a Response object
response = urllib2.urlopen(url)

# Which we can then read(), and hopefully get some text
text = response.read()

print "The latest version of Charles Proxy is: ", text
```

```
The latest version of Charles Proxy is:  4.2.8
```

https://notebooks.azure.com/i-walrus/projects/macad-uk-speak-api-with-python/html/api_examples.ipynb

```
# If the response is JSON...
json_url = 'https://update.code.visualstudio.com/api/update/darwin/stable/05f146c7a8f7f78e8026
1aa3b2a2e642586f9eb3'
```

```
response = urllib2.urlopen(json_url)

print response.read()
```

```
{"url":"https://az764295.vo.msecnd.net/stable/a3db5be9b5c6ba46bb7555ec5d60178ecc2eaae4/VSCode
-darwin-stable.zip","name":"1.32.3","version":"a3db5be9b5c6ba46bb7555ec5d60178ecc2eaae4","pro
ductVersion":"1.32.3","hash":"2015bb998f454b90871cc6aeb21a19d89f9ea54b","timestamp":155260377
7614,"sha256hash":"9a4274be0b3ab5c35e729f1cf5bab2459d32cef4feb8cd1517cbdf54f2bd4d3e"}
```

```
# We can parse the json response by passing it directly to the
# json.load() function.
response = urllib2.urlopen(json_url)
data = json.load(response)

# The result is a Python dict which can be queried and
# manipulated in the usual way:

print "Latest version of Visual Studio Code:"
print "Version: ", data['name']
print "URL: ", data['url']
```

```
Latest version of Visual Studio Code:
Version:  1.32.3
URL:  https://az764295.vo.msecnd.net/stable/a3db5be9b5c6ba46bb7555ec5d60178ecc2eaae4/VSCode-d
arwin-stable.zip
```

https://notebooks.azure.com/i-walrus/projects/macad-uk-speak-api-with-python/html/api_examples.ipynb

```python
# Let's try accessing LibreOffice's update service
xml_url = 'http://update.libreoffice.org/check.php?pkgfmt=dmg'

response = urllib2.urlopen(xml_url)
print response.read()
```

```
<html>
<head><title>LibreOffice Update Service</title></head>
<body>
  <h1>LibreOffice Update Service</h1>
  <p><b>Error:</b> Only LibreOffice can access the update service.</p>
</body>
</html>
```

Uh huh?

```python
# Aha! We are going to need to impersonate LibreOffice.
# We can do this by setting the User-Agent: header
# Charles's 'copy cURL command' gave us this:
#
# curl -H 'Host: update.libreoffice.org'
#      -H 'Pragma: no-cache'
#      -H 'Accept-Language: en-US'
#      -H 'User-Agent: LibreOffice 5.2.6.2 (a3100ed2409ebf1c212f5048fbe377c281438fdc; MacOSX;
X86_64; )'
#      --compressed 'http://update.libreoffice.org/check.php?pkgfmt=dmg'
#
# The only bit we need here is the User-Agent header.

xml_url = 'http://update.libreoffice.org/check.php?pkgfmt=dmg'

# A dictionary of headers can be built like this
headers = {'User-Agent': 'LibreOffice 5.2.6.2 (a3100ed2409ebf1c212f5048fbe377c281438fdc; MacOS
X; X86_64; )'}

# We then need to build a Request object and add our headers to it:
request = urllib2.Request(url=xml_url, headers=headers)

# And pass that to urllib2.urlopen()
response = urllib2.urlopen(request)

# This time we get the data we want:
print response.read()
```

https://notebooks.azure.com/i-walrus/projects/macad-uk-speak-api-with-python/html/api_examples.ipynb

```
<?xml version="1.0" encoding="utf-8"?>
<inst:description xmlns:inst="http://update.libreoffice.org/description">
  <inst:id>LibreOffice 6.1.5</inst:id>
  <inst:gitid>90f8dcf33c87b3705e78202e3df5142b201bd805</inst:gitid>
  <inst:os>MacOSX</inst:os>
  <inst:arch>X86_64</inst:arch>
  <inst:version>6.1.5</inst:version>
  <inst:buildid>9999</inst:buildid>
  <inst:update type="text/html" src="https://www.libreoffice.org/download/download/?lang=en-US&amp;version=6.1.5" />
</inst:description>
```

```python
# It's tempting to just search this for the information we want, but
# let's parse it properly using the ElementTree module:

# Grab a fresh response
response = urllib2.urlopen(request)

# Load it into our ElementTree
tree = ElementTree.fromstring(response.read())

# Set up the XML namespace - to save space
namespace = {'inst': 'http://update.libreoffice.org/description'}

# Then search for the data we want
version = tree.findall('inst:version', namespace)[0].text
build = tree.findall('inst:buildid', namespace)[0].text

print "Latest version info for LibreOffice:"
print "Version: ", version
print "Build: ", build
```

```
Latest version info for LibreOffice:
Version:  6.1.5
Build:  9999
```

https://notebooks.azure.com/i-walrus/projects/macad-uk-speak-api-with-python/html/api_examples.ipynb

Back to AutoPkg

Wwise AutoPkg Recipe

Introduction

Wwise is installed via an opaque Electron application called the Wwise launcher. This recipe takes the approach of remiplementing enough of the behaviour of the installer application to download the necessary archives and install them to a package root, then create a package.

The Wwise application itself is a Windows application which is packaged to run on macOS under Crossover. This recipe also patches the install so that the necessary extra Windows libraries are downloaded on first run without user intervention

Configuration

Several configuration variables are available:

- `VERSION` : The version to install. Setting this to LATEST or an empty string will give you the latest version available
- `EMAIL` : If you have a Wwise login and wish to access proytected assets, set this variable to your email address
- `PASSWORD` : ... and this one to the relevant password
- `INSTALL_STYLE` : Should be set to `mini` or `maxi` . `mini` will give you just the authoring packages, which is all that is required in our environment. `maxi` (experimental) will install all Wwise packages that are available, including SDKs for numerous platforms. If your email address has access to restricted content, `maxi` may install it, but this hasn't been tested.

Implementation

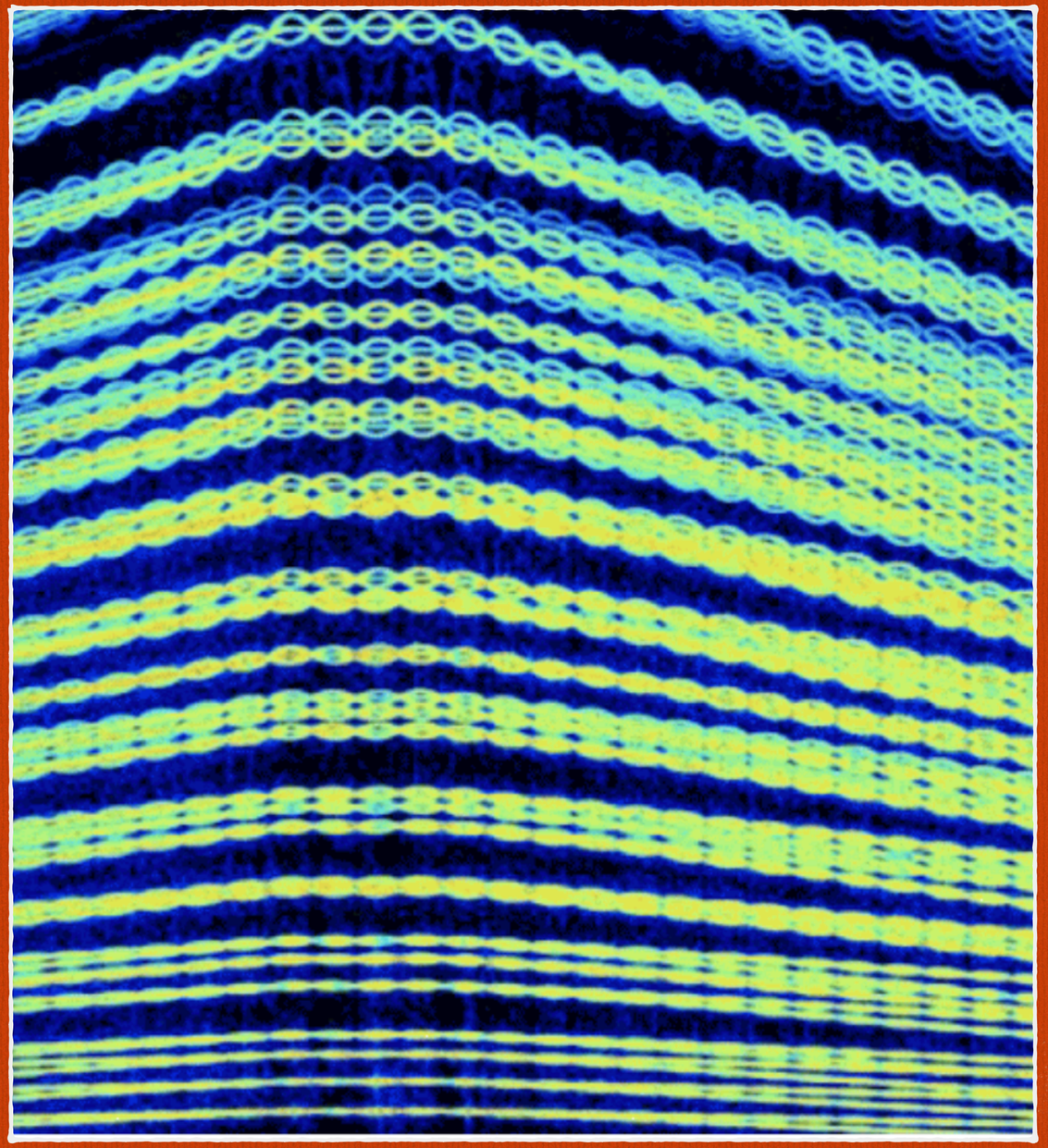
The `WwiseInstallProvider.py` processor is a thin wrapper around the `wwise_helper.py` tool in this directory. `wwise_helper.py` can be used on its own and has various options for controlling the installation. Run it directly for more

https://github.com/UoE-macOS/autopkg-recipes/tree/master/Wwise

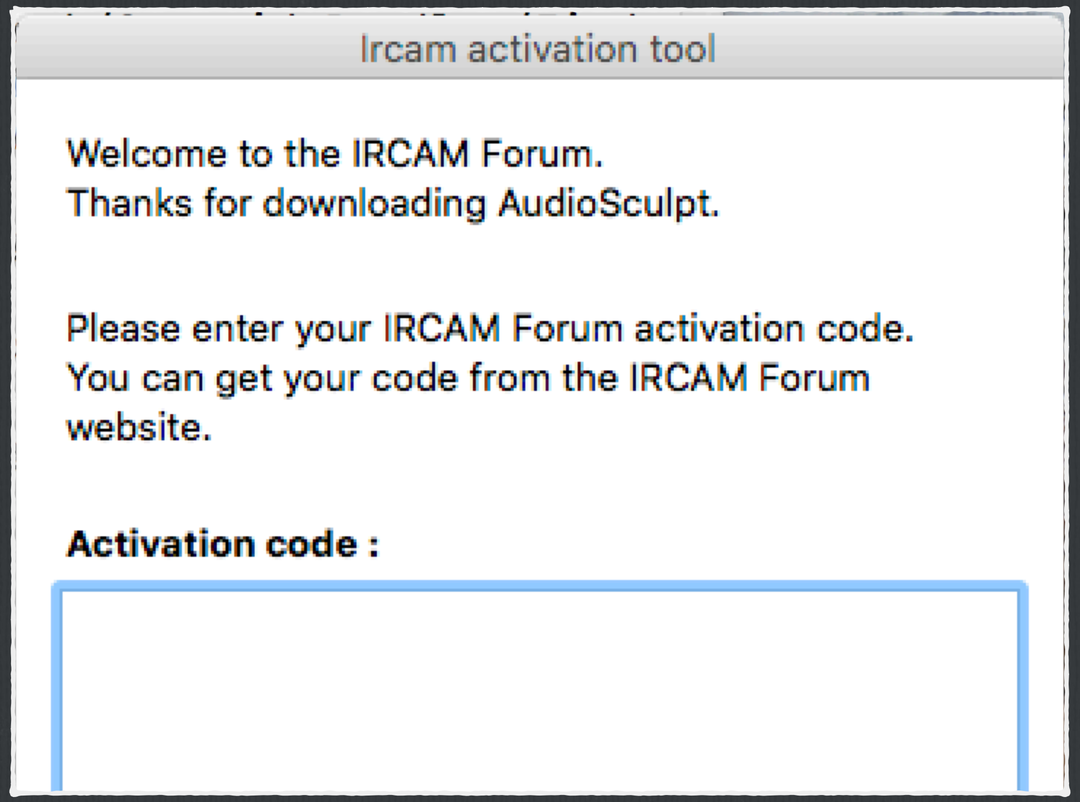
- ☐ Now, we can write an autopkg processor
- ☐ Lots of possible approaches
- ☐ Build in flexibility
- ☐ We can tailor for our own priorities
  - ☐ Bandwidth
  - ☐ Timely updates

Case Study

Hidden command line options

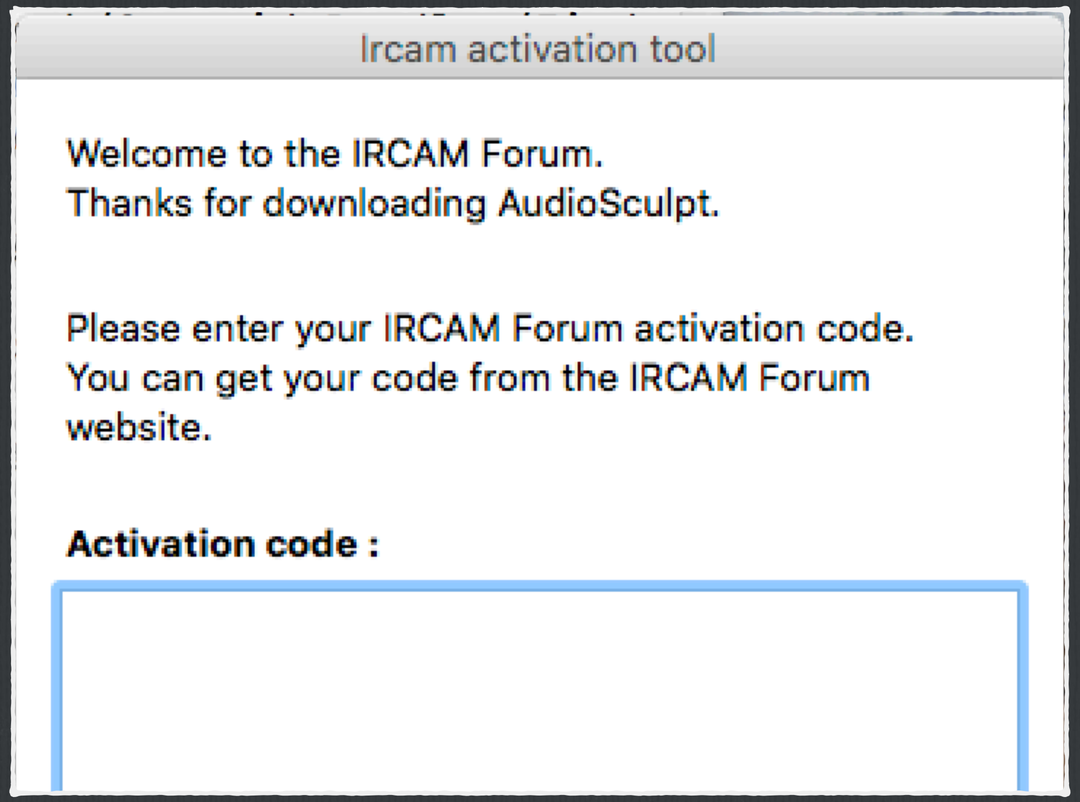


IRCAM Forum



- ☐ Software written by extremely clever people
- ☐ Who hate sysadmins
- ☐ Activation codes are downloaded from a portal
- ☐ Are valid for 14 days
- ☐ And need to be renewed annually
- ☐ We deploy this to 30 machines

IRCAM Forum



- Oh, and registration creates a hidden file in the home directory of the current user
- We would register the software in our support account
- A login script copied the activation file to each new user on login
- Conversations with vendor stalled…

STRINGS(1) STRINGS(1)

NAME

strings - find the printable strings in a object, or other binary, file

SYNOPSIS

strings [-] [-a] [-o] [-t format] [-number] [-n number] [--] [file ...]

DESCRIPTION

Strings looks for ASCII strings in a binary file or standard input. Strings is useful for identifying random object files and many other things. A string is any sequence of 4 (the default) or more printing characters [ending at, but not including, any other character or EOF]. Unless the - flag is given, strings looks in all sections of the object files except the (__TEXT,__text) section. If no files are specified standard input is read.

One fine day…

Running `strings` on a subcomponent of the main application:

```
user$ strings /Applications/AudioSculpt\ 3.4.6/AudioSculpt\ 3.4.6.app/Contents/MacOS/pm2 | grep activ

Please register an activation key using -init_key_file Ircam_Activation_Key.txt
Ircam activation tool : error 0x
Please paste your IRCAM Forum activation code down here.
Please enter your activation code.
Please enter your activation code.
Please register an activation key using -init_key_file Ircam_Activation_Key.txt
Ircam activation tool : error 0x
Please paste your IRCAM Forum activation code down here.
Please enter your activation code.
Please enter your activation code.
```

IRCAM Forumnet Activation Package Builder

Usage

`./build.sh path_to_license_file expiry_date

Description

The IRCAM forumnet software is activated by passing a key file (received via email) to one of any of the commandline tools that make up the suite. This script will build a package which contains the key file and a script to do the activation.

Limitations

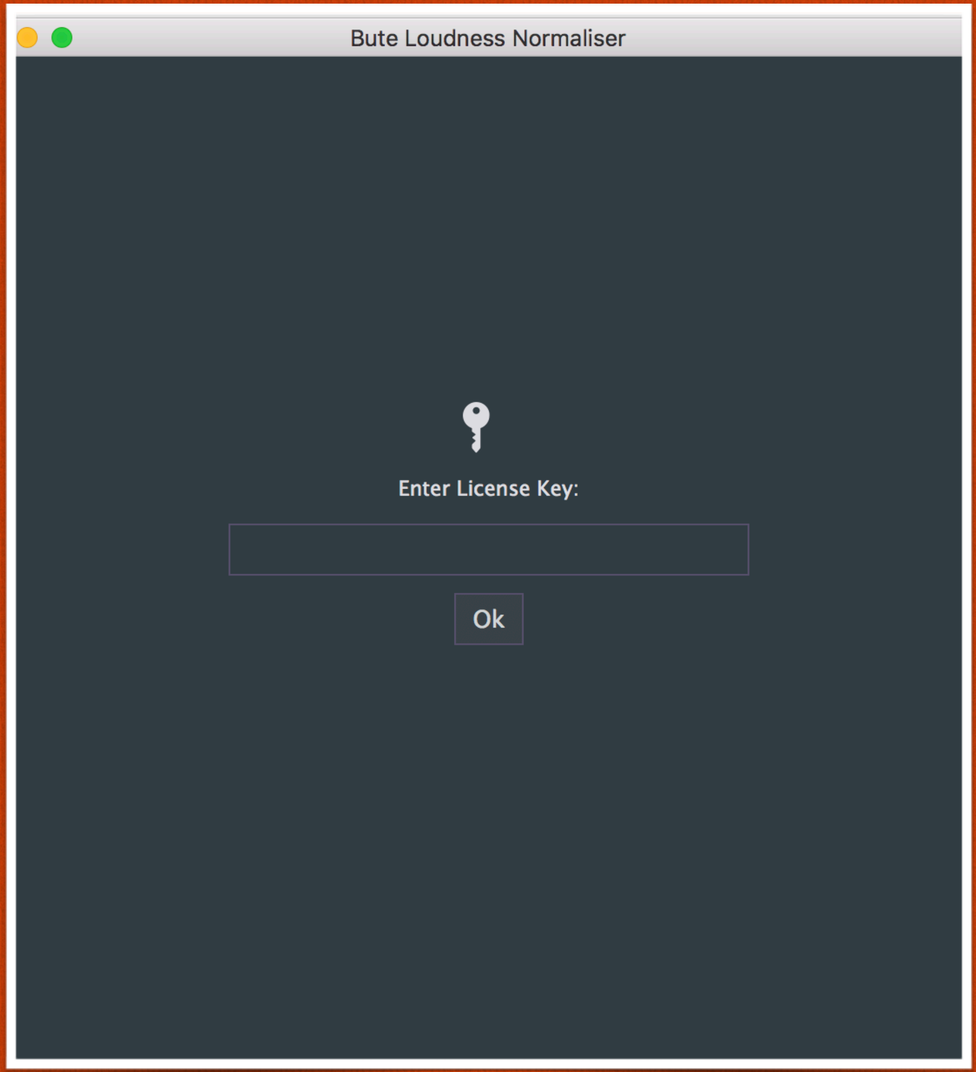
Key files are only valid for 14 days, so it is expected that you'll pass the expiry date as an argument so that it can be included in the package name. Packages used after their expiry date will not authorise the software.

The activation script relies on being able to find one of the IRCAM commandline tools to carry out authorisation. Since at the time of writing, AudioSculpt is the only tool we install, we look for one of its components. The script could easily be modified to look for something else.

https://github.com/UoE-macOS/packages/tree/master/ircam-forumnet-activate

Case Study

Some vendors are awesome



To: support@signumaudio.com

Greetings,

We have a multi-seat license for the Bute LoudnessSuite (Surround) which I need to deploy to 20 machines. Can you advise on how best I can go about this?

I've noticed that on activation the plugin produces a file at ~/Library/Application\ Support/ButeLoudnessSuiteSurround/ ButeLoudnessSuiteSurround.key

and that if I move the file to the system-wide location

/Library/Application\ Support/ButeLoudnessSuiteSurround

then the plugin will be registered for all users of the machine. However, the key file still seems to be locked to a particular machine, and in a managed lab environment it's not practical for us to visit 20 machines to authorise this plugin.

Do you have a commandline option for generating the key file, or some way to generate a key file that can be used on multiple machines?

From: support@signumaudio.com

Since the fingerprint needs to be generated on the machine itself you will have to do the authentication step on the machine.

The only shortcut I can think of if entering the key is too much work is an executable that generates the keyfile on execution. You could run it automatically on boot up..

…

yes a commandline tool that has your key hardcoded and would run on machine bootup to ensure the keyfile …
…

Note that we don't currently have such an auto keyfile generator, but it shouldn't be too much of an effort to write you one.

Happy packaging!