Building for RISC OS, Online

and what makes it tick

Gerph, November 2020

0. Introduction

November
2020

Introduction

How I'll do this talk

- Lots of technology, some of which may be alien to you.
- The talk is split into 5 sections, with a chance for some questions between them.
- Slides will be available at the end, together with some other resources.
- I'll take questions at the end for as long as people want.

Introduction

What we'll talk about

1. Some background.
2. What JFPatch-as-a-service is.
3. How it works.
4. What powers it.
5. Conclusions.

1. Background

March
2019

Background

Who am I?

- A RISC OS architect and engineer, who's been away from the community for about 15 years.
- I used to do a lot of things with RISC OS, which you can read about on my site if you're interested - gerph.org/riscos
- I'm not going to talk about that past here.
- I would like to think that I probably know RISC OS in design and execution better than anyone.

Background

Dear gosh, why?

- What do I want to do with RISC OS and why?
  - Let's make something for me, because I can.

Background

So, you want to use RISC OS, but…

- Development on RISC OS is tedious
  - The tools aren't great but they only run on RISC OS… and I don't have a RISC OS system (other than RPCEmu)
- RISC OS testing is awful
  - Most RISC OS projects do ad-hoc testing, rely on users; no automation
- RISC OS is awful for testing
  - If something goes wrong, you need to hard reboot; no isolation; no security

Background

How does the real world do things?

- Source control !
- Cross compiling
- Managed development environments
- Automated testing of changes
- Feature and regression testing
- Fleets of systems available for use

Background

How can I do this? (1)

Source control:

- Move things to Git, because CVS is so very painful.

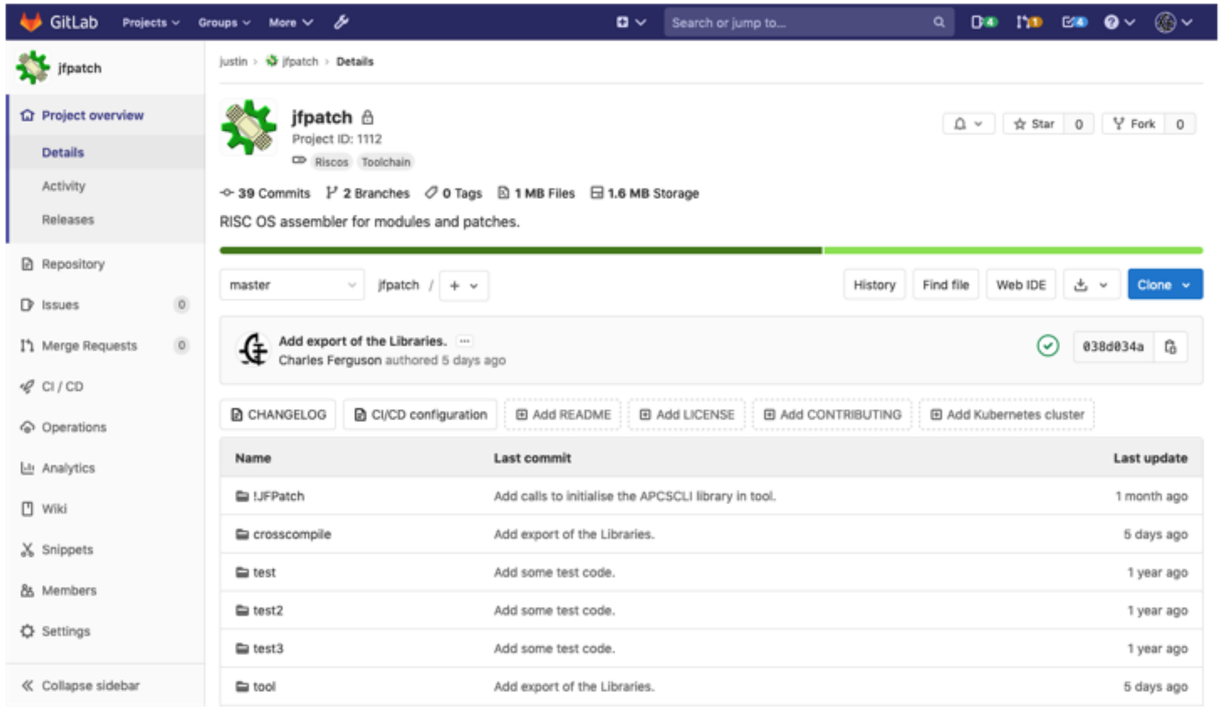
Background

How can I do this? (1)

Source control:

- Move things to Git, because CVS is so very painful.

Tech: GitLab, running on my linux server - it's publicly accessible, but most of the 1000-odd projects are private.



Background

How can I do this? (2)

Cross compiling:

Background

How can I do this? (2)

Cross compiling:

- Already had the toolchain ported to 32bit Linux and Windows, back in 2005.

Tech: Port the toolchain to 64bit Linux and 64bit macOS.

```
charles@laputa ~/pro/RO/mod/ris/Sou/Des/TaskWindow (master)> rm o*/*; riscos-amu
BUILD32=1 ram
riscos-objasm   -Stamp -quit   -I@ -predefine "BUILD_RAM SETL {TRUE}" -apcs
3/32/fpe2/swst/fp -predefine "BUILD_ZM SETL {TRUE}" -predefine "No26bitCode SETL {TRUE}"
-predefine "No32bitCode SETL {FALSE}" -predefine "APCS SETS \"APCS-32\"" -o oz32/Taskman
s/Taskman
ARM AOF Macro Assembler 3.32 (JRF:3.32.38) [07 Mar 2006]
Unrecognised APCS qualifier /fpe2
Unrecognised APCS qualifier /fp
MyDomain = 0000058C
Deprecated form of PSR field specifier used (use _cxsf)
riscos-link -rmf -rescan -C++ -o rm32/TaskWindow,ffa oz32.Taskman
TaskWindow: Module built {RAM}
```

Background

How can I do this? (2)

Cross compiling:

- Already had the toolchain ported to 32bit Linux and Windows, back in 2005.

Tech: Port the toolchain to 64bit Linux and 64bit macOS.

Tech: Tool to extract example code from 'Rosetta Code' for testing (https://github.com/gerph/rosettacode)

Background

How can I do this? (3)

Managed environments:

- How do I get my toolchain? find my libraries? store built components?

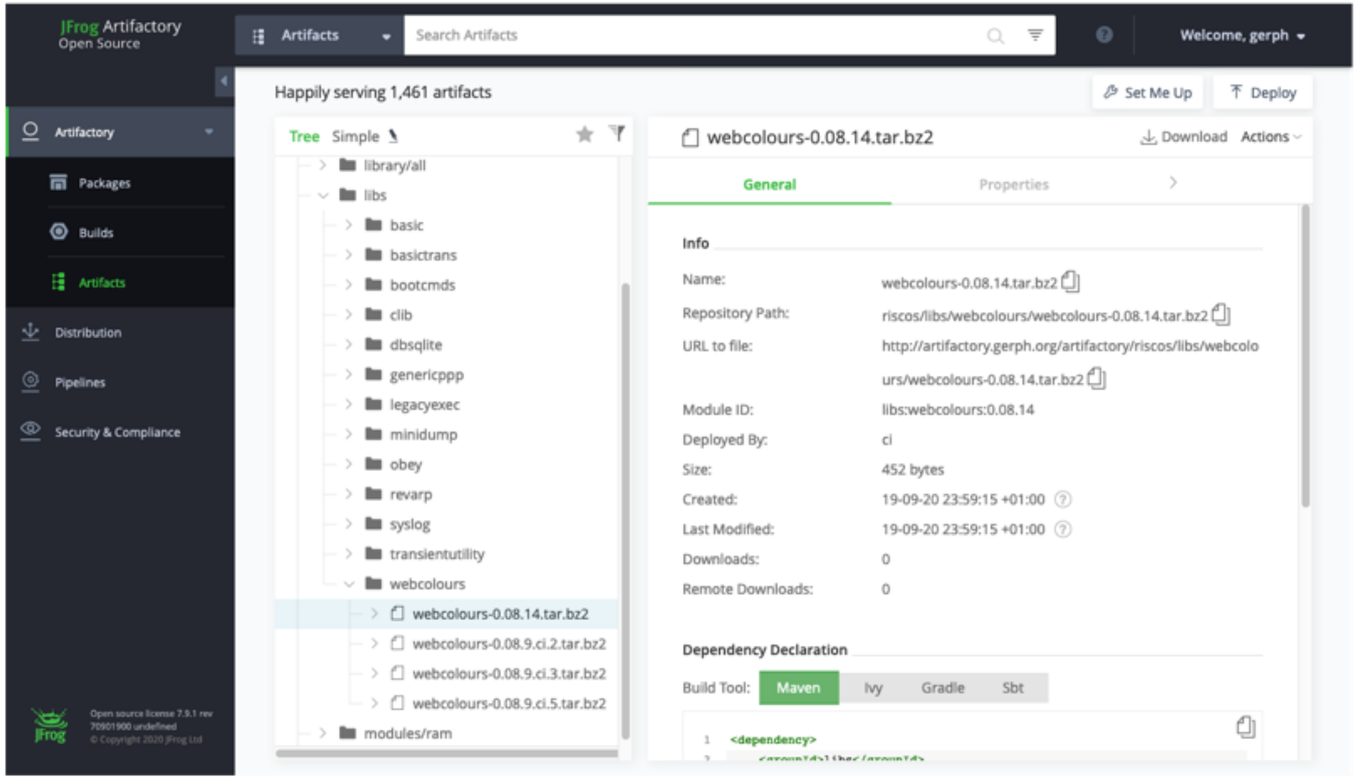
Background

How can I do this? (3)

Managed environments:

- How do I get my toolchain? find my libraries? store built components?

Tech: Artifactory for artifacts, and created some tools for pushing and pulling resources.



Background

How can I do this? (4)

Managed environments: (cont'd)

- What if I don't want to download my toolchain all the time?

Background

How can I do this? (4)

Managed environments: (cont'd)

- What if I don't want to download my toolchain all the time?

Tech: Docker RISC OS development environment.

```
charles@laputa ~/pro/RO/mod/ris/Sou/Des/WindowScroll (master)>
docker run -it --rm -v $PWD:/riscos-source -v $PWD/build:/riscos-build --workdir /riscos-source
gerph/riscos-build riscos-amu
riscos-cmunge   -px -DCMHG   -IC:,RISC_OSLib: -26bit -o oz/modhead cmhg/modhead
CMunge 0.77 (JRF:0.77.47) [13 Jun 2006]
Copyright (c) 1999-2006 Robin Watts/Justin Fletcher
Norcroft RISC OS ARM C vsn 5.18 (JRF:5.18.119)  [Jun  7 2020]
ARM AOF Macro Assembler 3.32 (JRF:3.32.38) [07 Mar 2006]
0 Errors, 2 Warnings suppressed by -NOWarn
riscos-cc   -c  -Wc -fa   -IC:,RISC_OSLib: -za1 -apcs 3/26/fpe2/swst/fp -D__CONFIG=26 -zM -zps1
-o oz/main c/main
Norcroft RISC OS ARM C vsn 5.18 (JRF:5.18.119)  [Jun  7 2020]
"c/main", line 564: Warning: '=': cast of 'int' to differing enum
c/main: 1 warning, 0 errors, 0 serious errors
riscos-link -rmf -rescan -C++ -o rm/WindowScroll,ffa oz.main oz.modhead C:o.stubs
```

Background

How can I do this? (5)

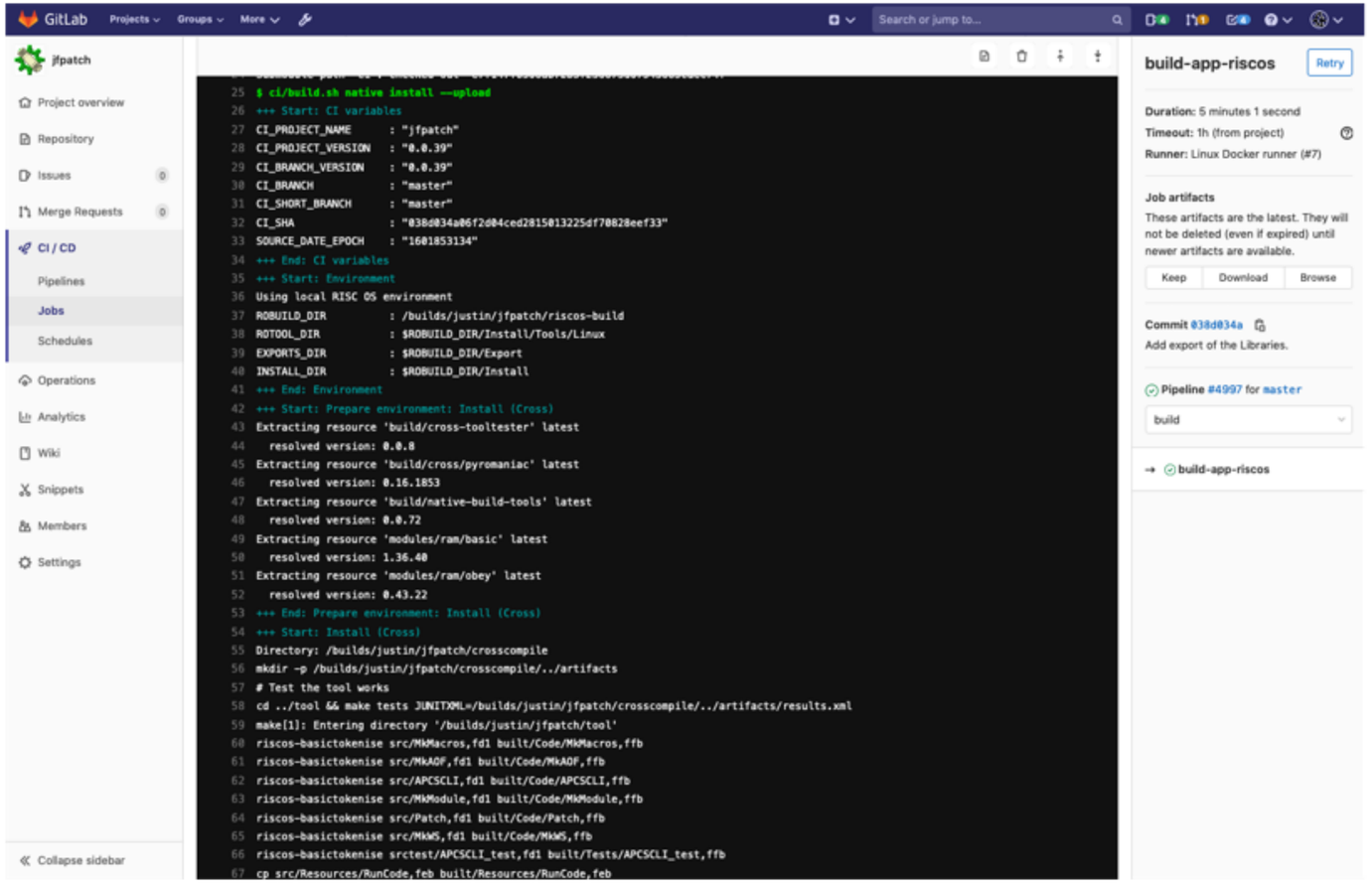
Automated testing:

Background

How can I do this? (5)

Automated testing:

Tech: GitLab CI triggers on every change - pulls resources from Artifactory, builds, pushes result to Artifactory.



Background

How can I do this? (6)

Feature and regression testing:

- Build programs and test code on other platforms.

Background

How can I do this? (6)

Feature and regression testing:

- Build programs and test code on other platforms.
- I need a way to test things on RISC OS, too…

Tech: … we'll come to that later …

Background

How can I do this? (7)

Fleets of systems for people to use:

- That seems a stretch, but maybe it's not so hard...

Tech: JFPatch-as-a-service begins that process

2. JFPatch-as-a-service

March
2020

JFPatch-as-a-Service

Why?

A friend said to me…

> *"I can't wait until you csa.announce this and confuse the bejesus out of the RISC OS civilians."*

To which my answer was…

> *"JFPatch as a service would be a doddle to do right now. A service that nobody asked for, or needed."*

JFPatch-as-a-Service

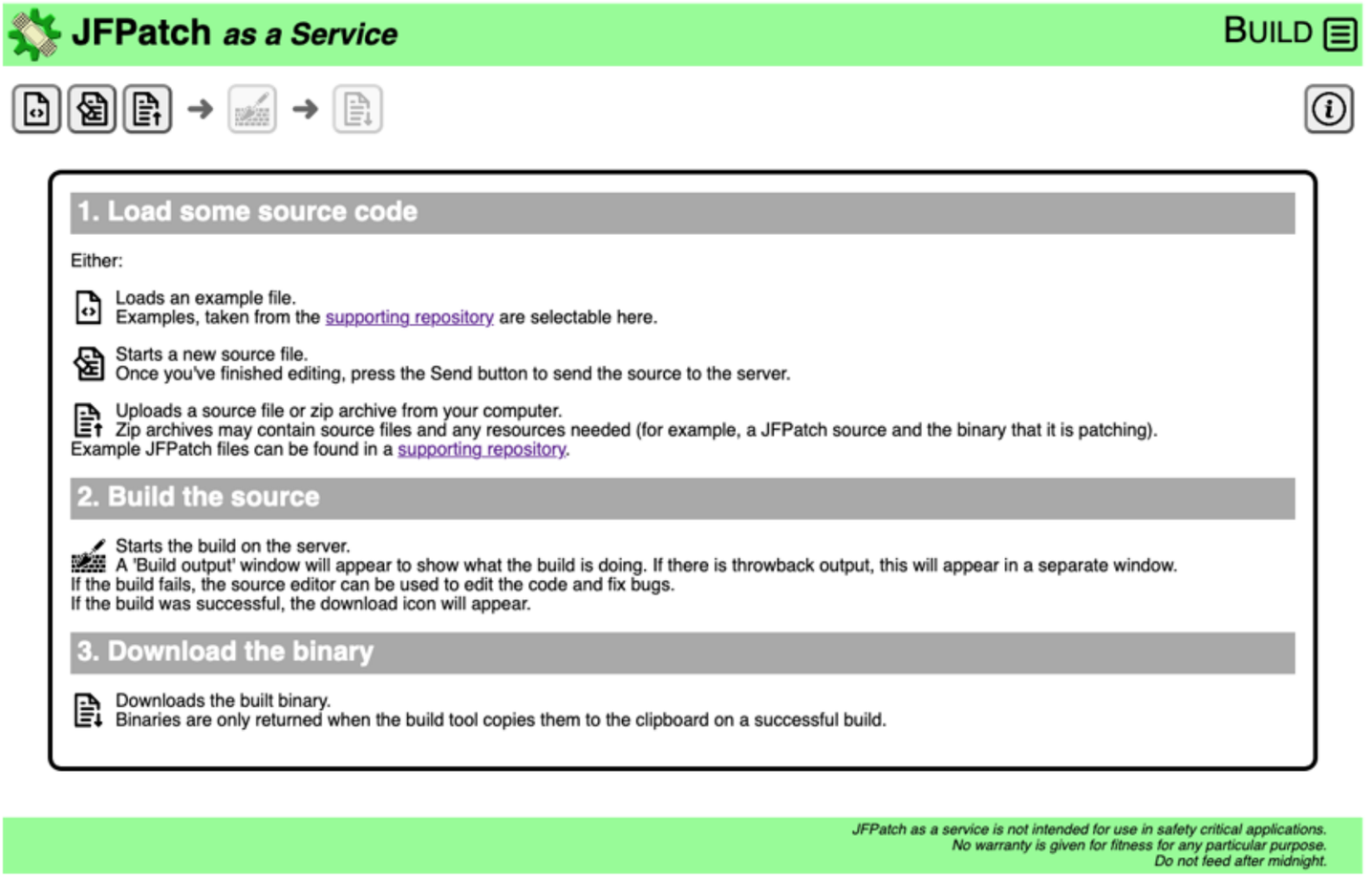
What is JFPatch?

- It's a pre-processor for the BASIC assembler.
- It has its own file format which describes things to patch, or modules to build.
- It converts these to BASIC files, then runs the BASIC, which does the heavy lifting of assembling.
- It is, itself, written in BASIC.
- It was used to write many of my early assembler modules.

JFPatch-as-a-Service

What is the service?

- Takes its inspiration from Matt Godbolt's Compiler explorer - https://godbolt.org/



JFPatch-as-a-Service

What can you build with the service? (1)

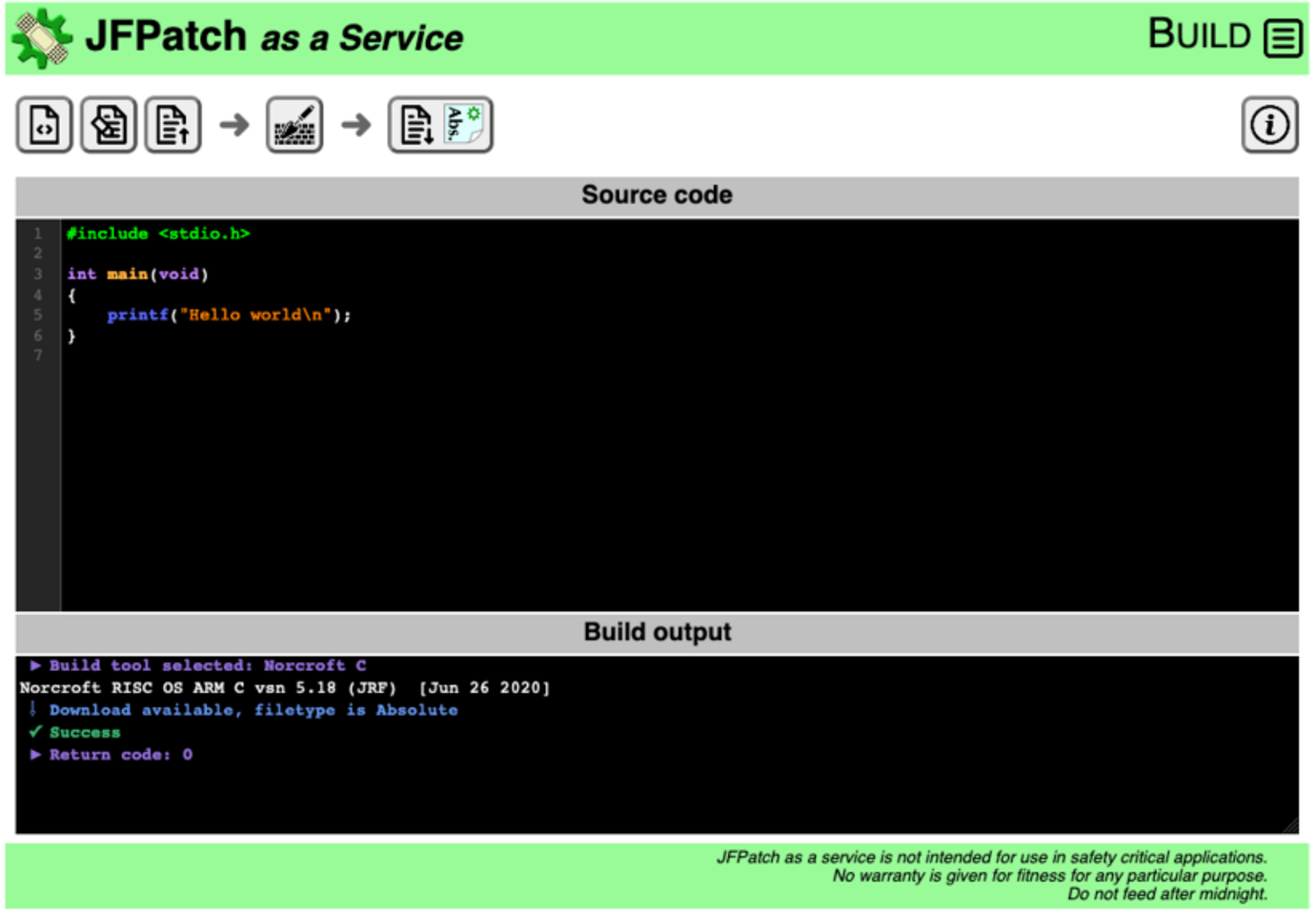
- Any JFPatch code (now builds 32bit code)
- C code that compiles with the Norcroft compiler
- Pascal code (which will be converted to C and compiled with the Norcroft compiler)
- Perl code
- BASIC assembler
- Objasm assembler.

Tech: All the toolchain is built for 32bit RISC OS, automatically taken from Artifactory when the service is built.

JFPatch-as-a-Service

What can you build with the service? (2)

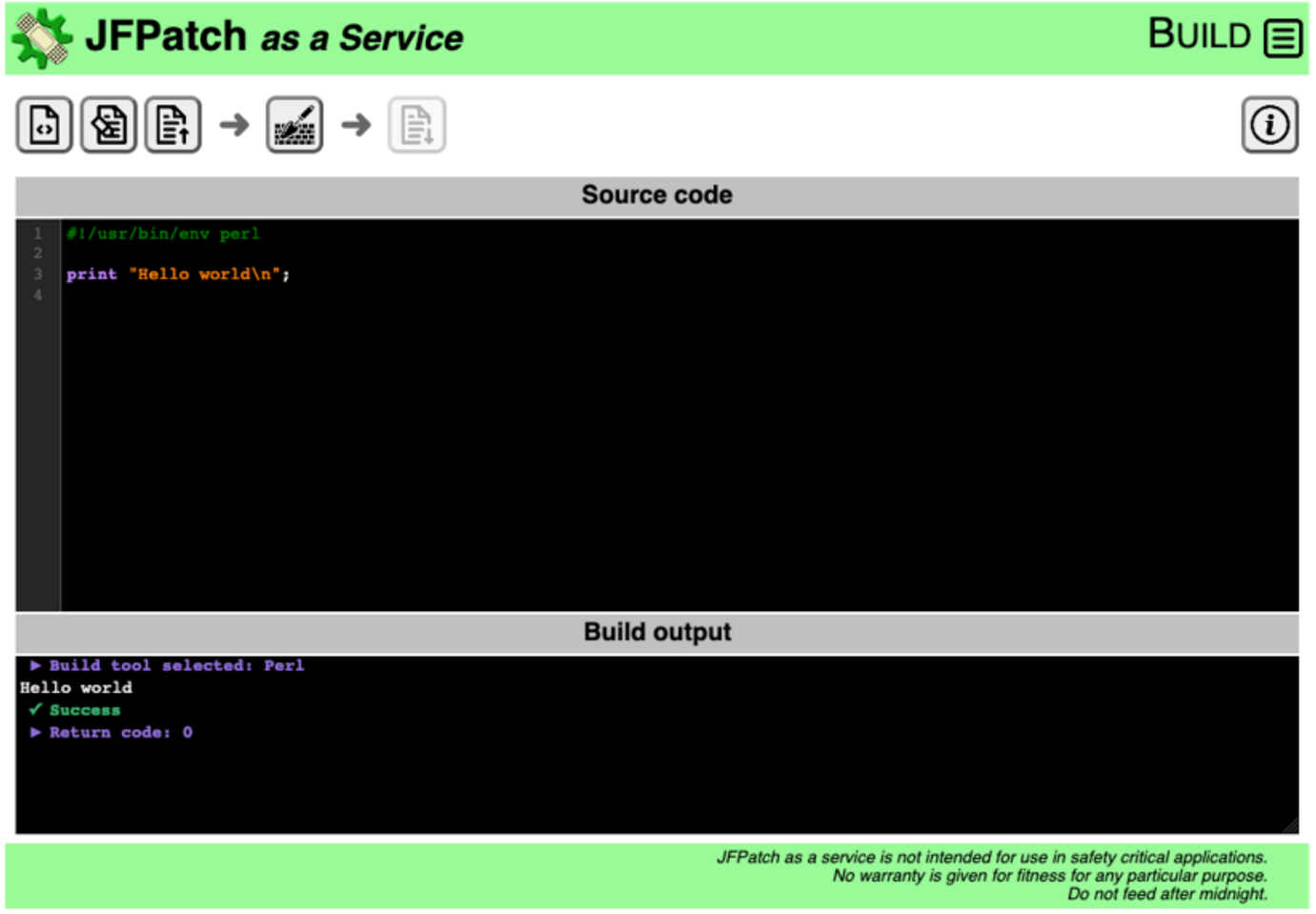
C code...



JFPatch-as-a-Service

What can you build with the service? (3)

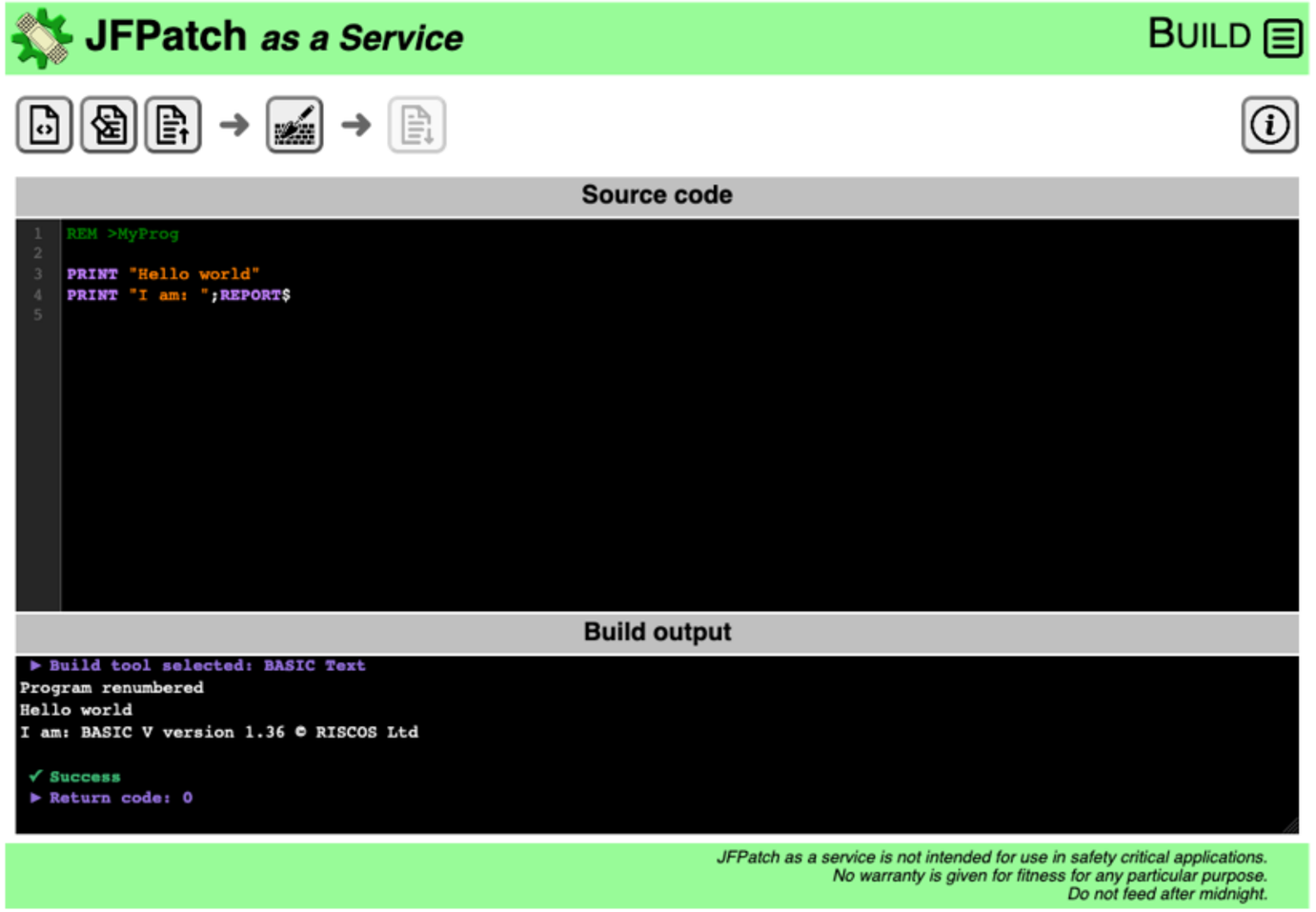
Perl code...



JFPatch-as-a-Service

What can you build with the service? (4)

Plain BASIC...



JFPatch-as-a-Service

What might use the service?

Automated builds can use this:

- LineEditor (BASIC assembler) - https://github.com/philpem/LineEditor
- Nettle (C application) - https://github.com/gerph/Nettle/tree/ci
- CObey (C module) - https://github.com/gerph/cobey
- ErrorCancel (ObjAsm) - https://github.com/gerph/errorcancel
- Pico (C command line tool) - https://github.com/gerph/pico
- DDEUtilsJF (JFPatch module) - https://github.com/gerph/ddeutilsjf

JFPatch-as-a-Service

How do you use the service?

Two interfaces, which are documented:

- JSON API.
- WebSockets API.

Documented on the website: https://jfpatch.riscos.online/api.html
Examples can be found at: https://github.com/gerph/jfpatch-as-a-service-examples

JFPatch-as-a-Service

How do you use the JSON API?

Use your favourite HTTP request library. For example, `curl`:

```
curl -i -F 'source=@source-file'  http://jfpatch.riscos.online/build/json
```

Get a JSON response:

```
{
  "data": "... data goes here ...",
  "filetype": 4092,
  "messages": [
    "Build tool selected: JFPatch",
    "Return code: 0"
  ],
  "output": [
    "JFPatch ARM assembler v2.56\u00df (02 Mar 2020) [Justin Fletcher]\r\n",
    "Pre-processing...\r\n",
    "Assembling...\r\n"
  ],
  "rc": 0,
  "throwback": []
}
```

JFPatch-as-a-Service

How do you use the WebSockets API?

Using the `wsclient.py` example gives a similar output.

```
welcome: u'Linking over Internet with RISCOS Pyromaniac Agent version 1.04'
response: u'Source loaded'
response: u'Started build'
message: u'Build tool selected: JFPatch'
output: u'JFPatch ARM assembler v2.56\xdf (02 Mar 2020) [Justin Fletcher]\r\n'
output: u'Pre-processing...\r\n'
output: u'Assembling...\r\n'
clipboard: {u'filetype': 4092, u'data': u'... data goes here ...'}
rc: 0
message: u'Return code: 0'
complete: True
```

Q: What about when you don't have, or can't use, Python?

A: `robuild-client` handles that.

JFPatch-as-a-Service

What is the robuild-client?

- Created a build client that can be used to do the heavy work.
- Can be found at https://github.com/gerph/robuild-client
- Builds for Linux...

JFPatch-as-a-Service

What is the robuild-client?

- Created a build client that can be used to do the heavy work.
- Can be found at https://github.com/gerph/robuild-client
- Builds for Linux...
- ... then uses the tool it built to submit its code to the service, to build the RISC OS version.

Tech:

- robuild-client.
- port of JSON parse/creation library.
- WebSockets library.

JFPatch-as-a-Service

How does the service know what to do?

- Simple files are recognised by their format.
- Zip files are recognised by their content.
  - The `.robuild.yaml` file can control what is actually run.

JFPatch-as-a-Service

How does the service know what to do?

- Simple files are recognised by their format.
- Zip files are recognised by their content.
  - The `.robuild.yaml` file can control what is actually run.

```
%YAML 1.0
---

jobs:
  build:
    # Env defines system variables which will be used within the environment.
    # Multiple variables may be assigned.
    env:
      "Sys$Environment": ROBuild

    # Commands which should be executed to perform the build.
    # The build will terminate if any command returns a non-0 return code or an error.
    script:
      - dir riscos
      - !BuildAll
      - Clipboard_FromFile client.aif32.riscos-build-online
```

3. How The Service Works

March
2020

How The Service Works

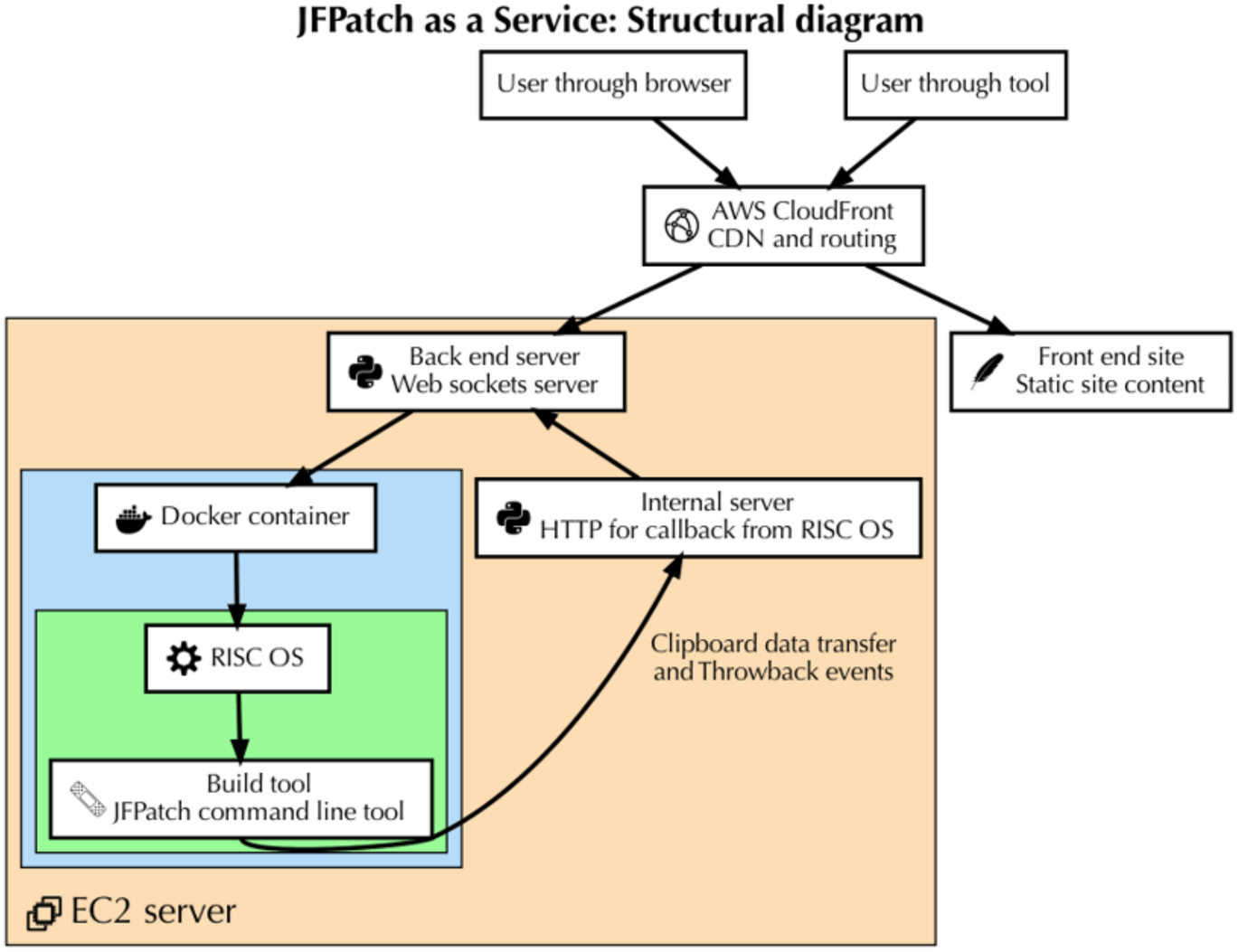
What is the service made of? (1)

Tech:

- Infrastructure - AWS SSL, routing and linux server.
- Front End - Static site, websockets to talk to back end
  - Custom CodeMirror colouring - https://github.com/gerph/CodeMirror/tree/riscos-modes
- Back End - Python REST JSON API and WebSockets service
  - RISC OS Zip file decoding in Python - https://github.com/gerph/python-zipinfo-riscos
- Tools - JFPatch tool, compiler, assembler, linker, amu, etc.

How The Service Works

What is the service made of? (2)

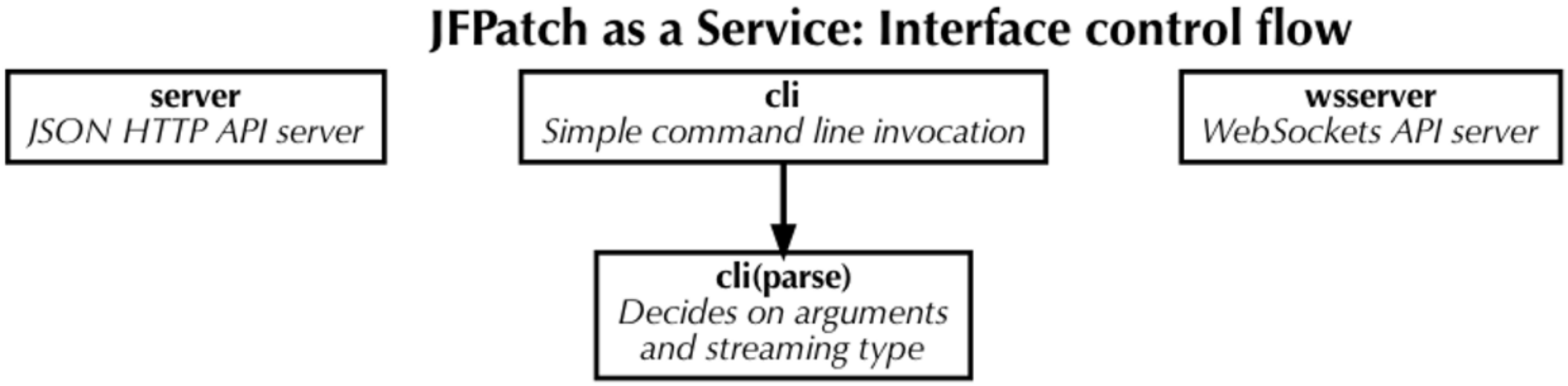


How The Service Works

What runs those services? (1)

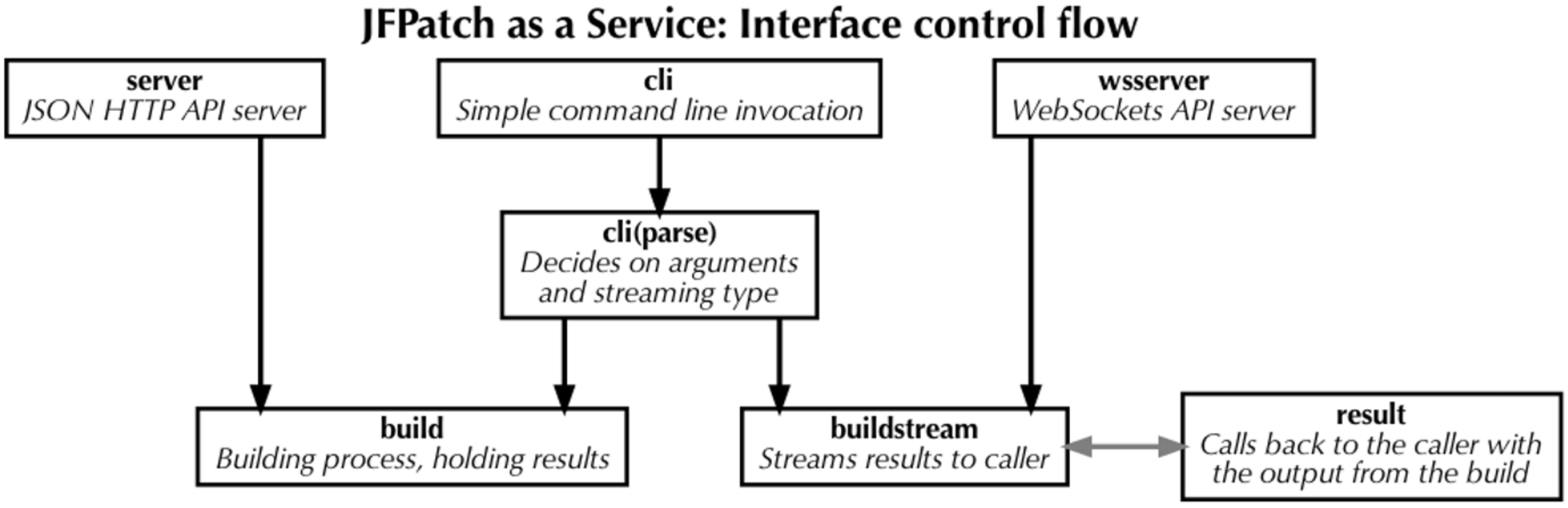
How The Service Works

What runs those services? (2)



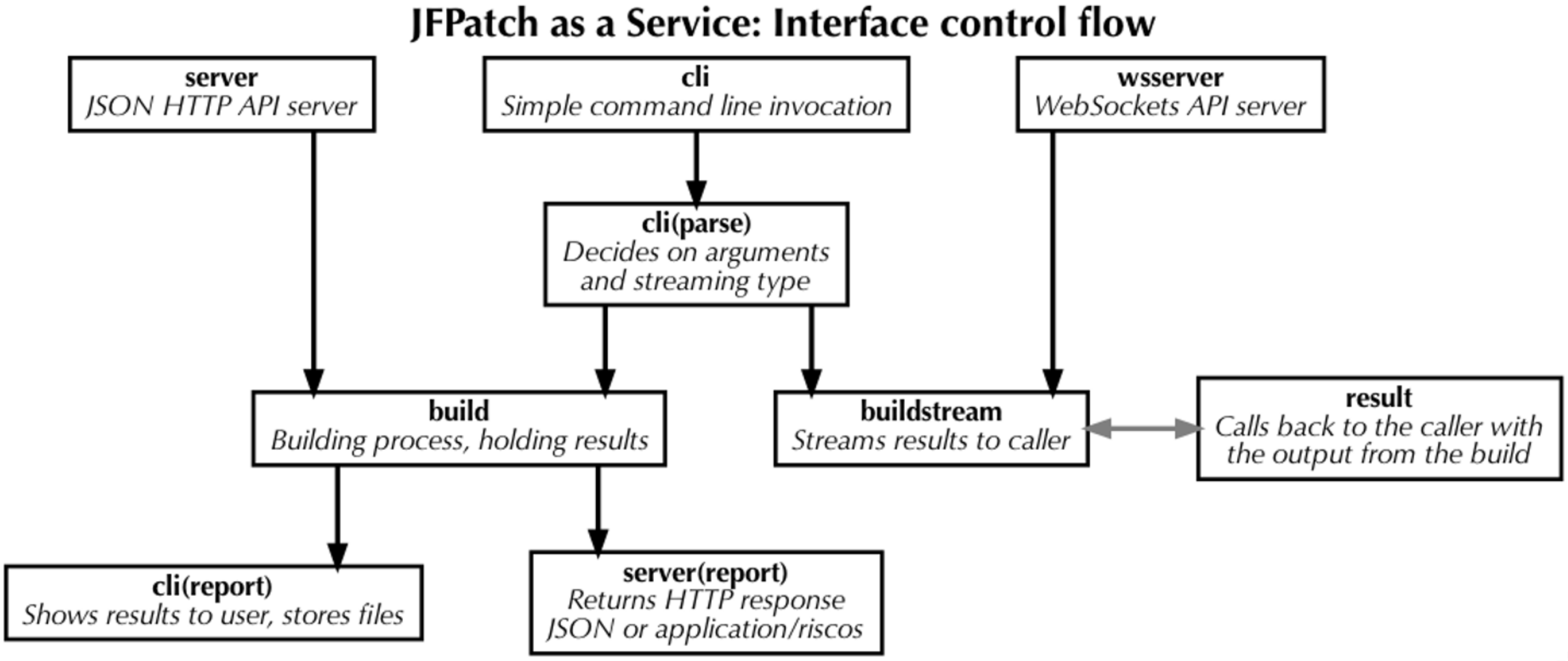
How The Service Works

What runs those services? (3)



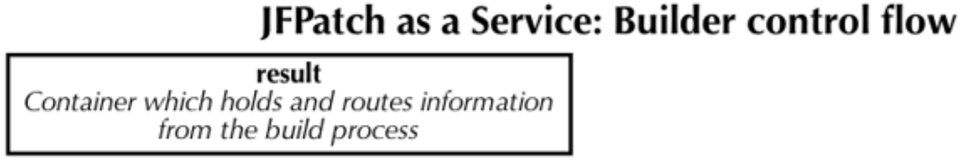
How The Service Works

What runs those services? (4)



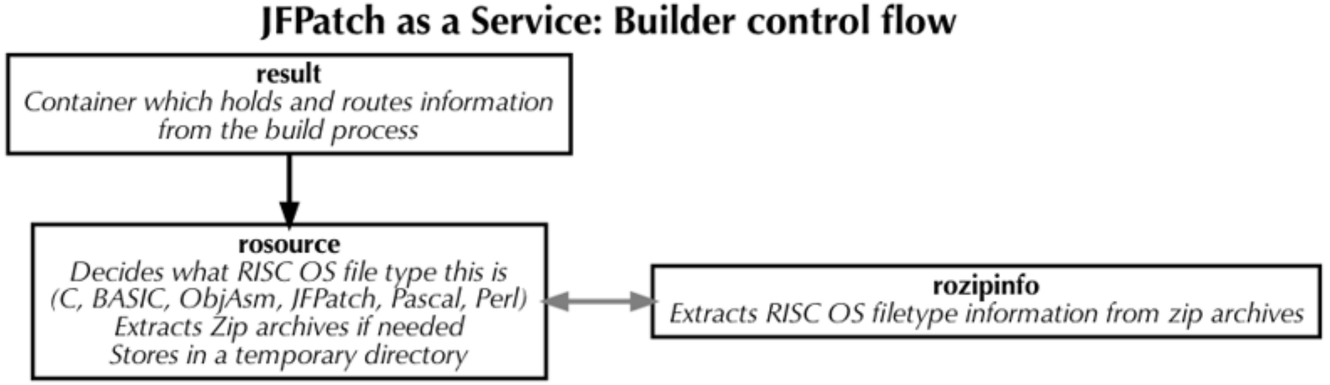
How The Service Works

What runs those services? (5)



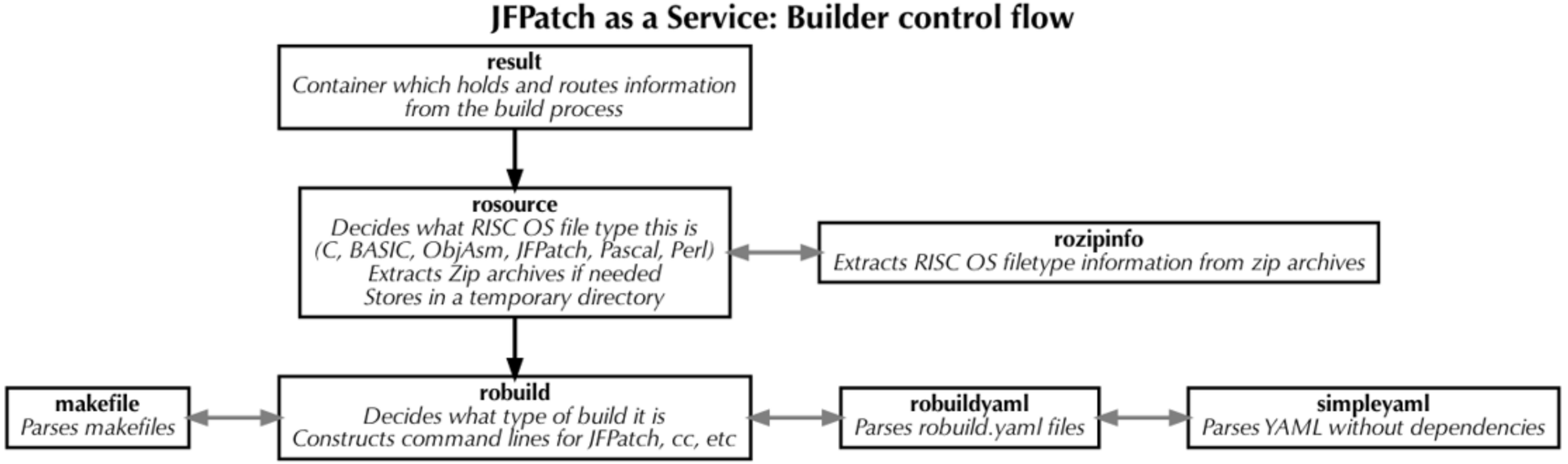
How The Service Works

What runs those services? (6)



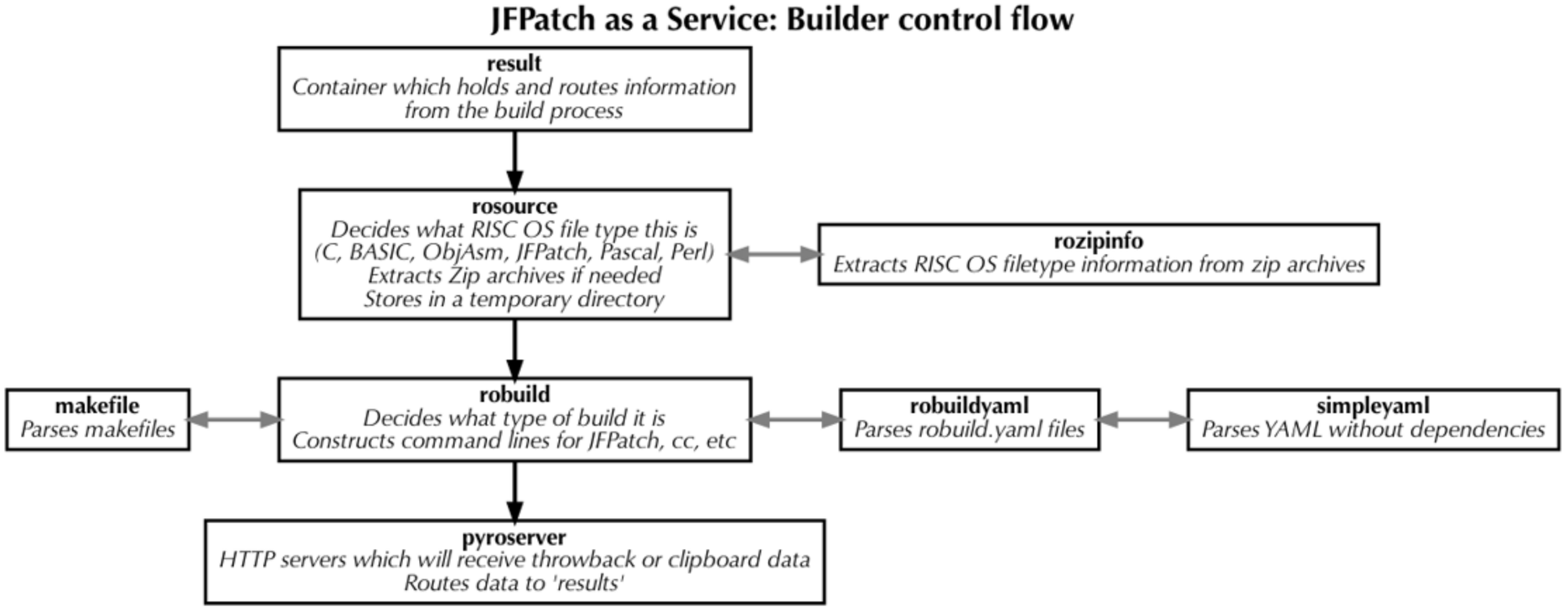
How The Service Works

What runs those services? (7)



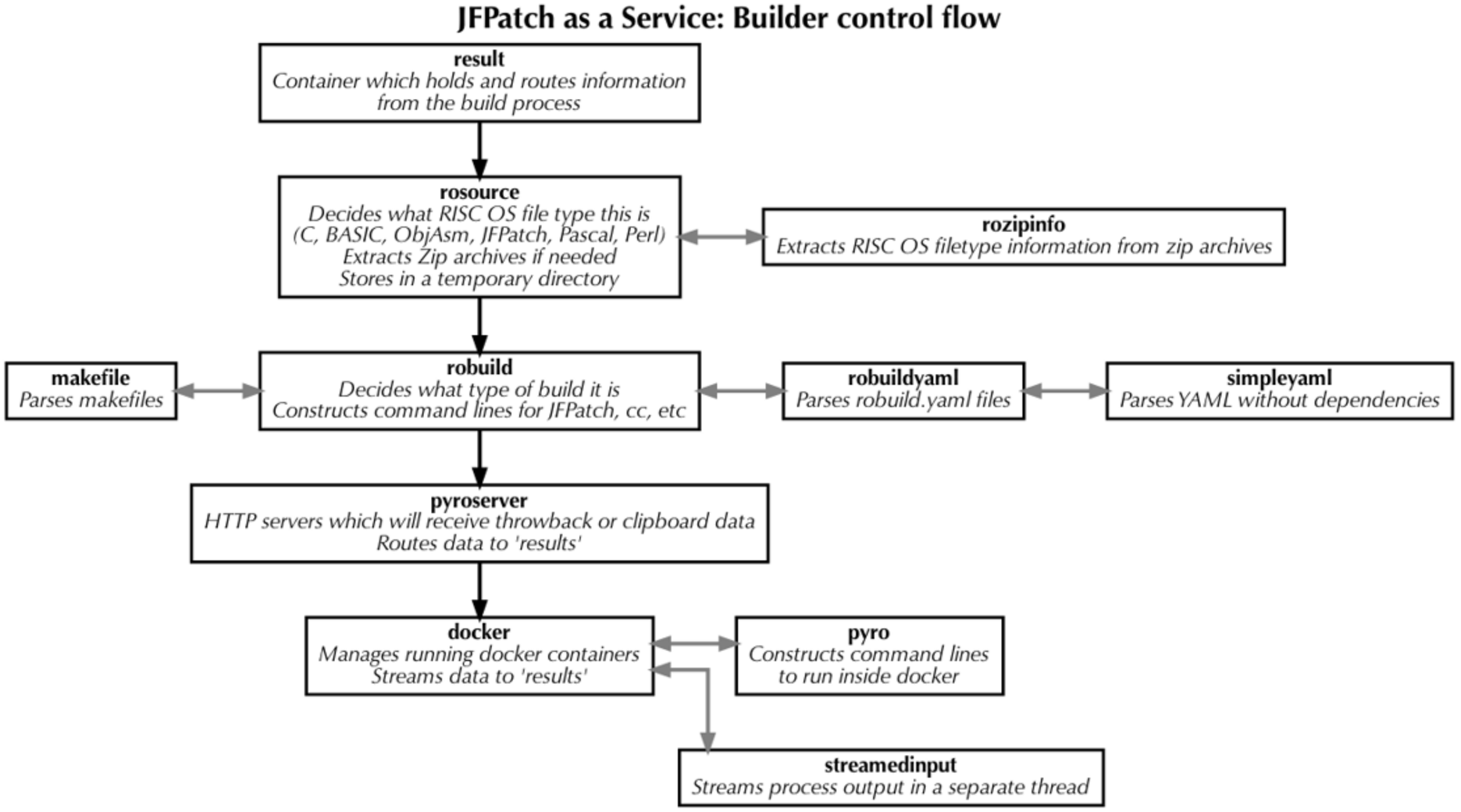
How The Service Works

What runs those services? (8)



How The Service Works

What runs those services? (9)



How The Service Works

What runs those services? (10)

- `robuild` has worked out the RISC OS commands to use.
- `pyro` is given those commands and constructs a command that can run RISC OS with those commands.
- `docker` is given that command, and builds a command to run RISC OS within a docker container.
- ... and the results of all of that gets fed back to the `results` object, which passes it back to the caller.

How The Service Works

Wait what?

> *“ Wait, RISC OS is running in Docker?*
> *But Docker runs on Linux?*
> *You're running RISC OS on Linux then? ”*

• • •

June
2019

4. RISC OS Pyromaniac

June
2019

RISC OS Pyromaniac

How do you test RISC OS software without RISC OS?

- My RiscPC is in storage.
- It's not good for testing.

RISC OS Pyromaniac

How do you test RISC OS software without RISC OS?

- My RiscPC is in storage.
- It's not good for testing.
- RISC OS was originally run semi-hosted from a BBC, using the BBC as the I/O and RISC OS as the main computer.
- That's what I want to be able to do - I want to be able to drive RISC OS from the CLI of a sane machine.

RISC OS Pyromaniac

Surely that's easy? (1)

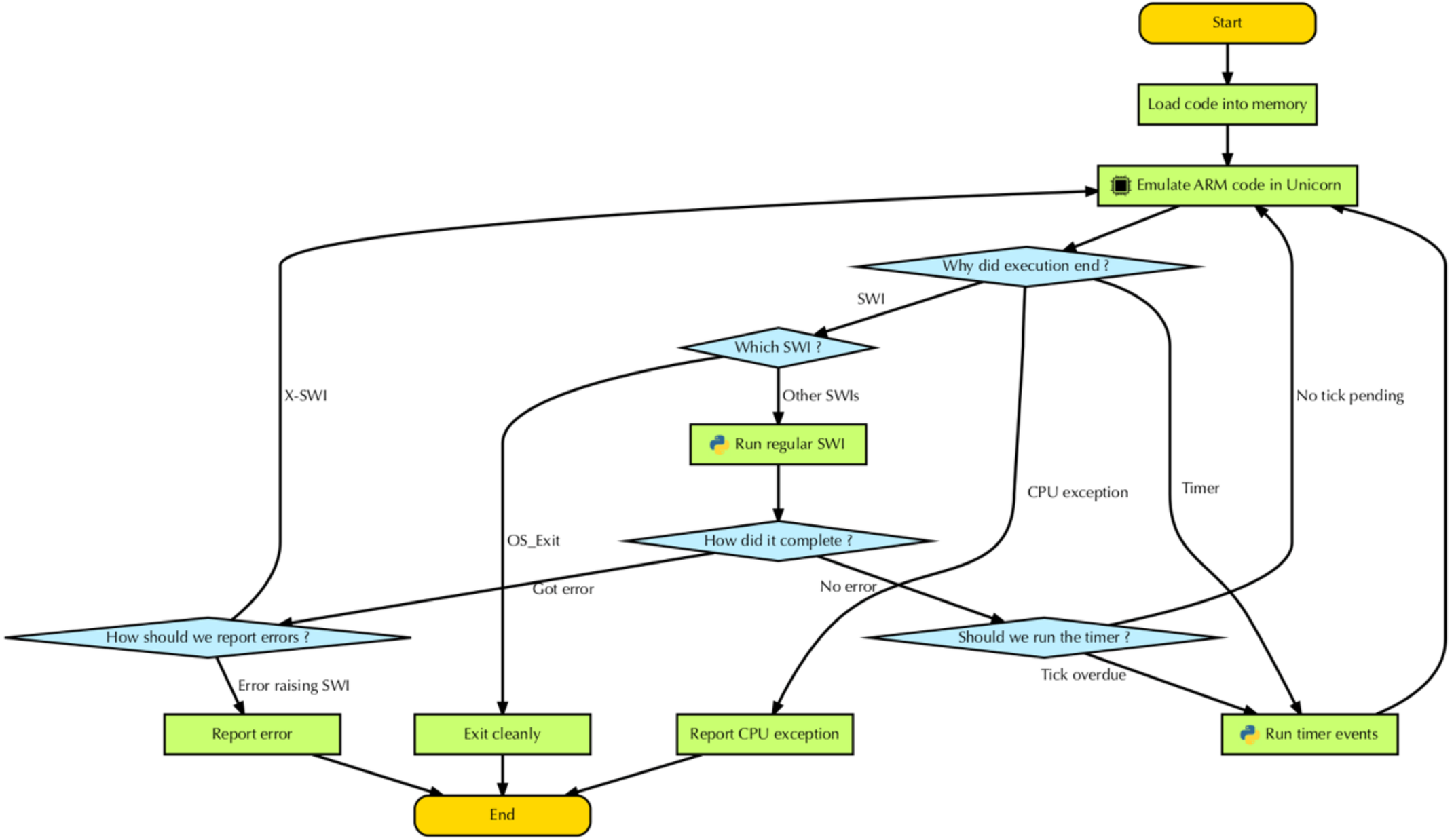
- Surely that's easy? You just run an emulation system until it hits a SWI...
  ... and then you make the SWI do the I/O thing. Then you run some more?

RISC OS Pyromaniac

Surely that's easy? (1)

- Surely that's easy? You just run an emulation system until it hits a SWI...
  ... and then you make the SWI do the I/O thing. Then you run some more?
- Yes - that's exactly what you do.
- The `IfThere` tool ran on June 10th - not well, but it ran.

Pyromaniac: Basic execution



RISC OS Pyromaniac

What is RISC OS Pyromaniac?

- Pyromaniac is an alternative implementation of RISC OS for non-ARM systems.
- It is intended for use as a testing and prototyping environment which may be used during development and automated testing.
- Written in a high level language to make that possible.

RISC OS Pyromaniac

What's in a name?

- Pyromaniac is the system that runs ARM code.
- RISC OS is what's implemented on top of that.

So...

- A name to distinguish it from the ARM implementation.
  - RISC OS Pyromaniac
- A name for the ARM implementations.
  - RISC OS Classic

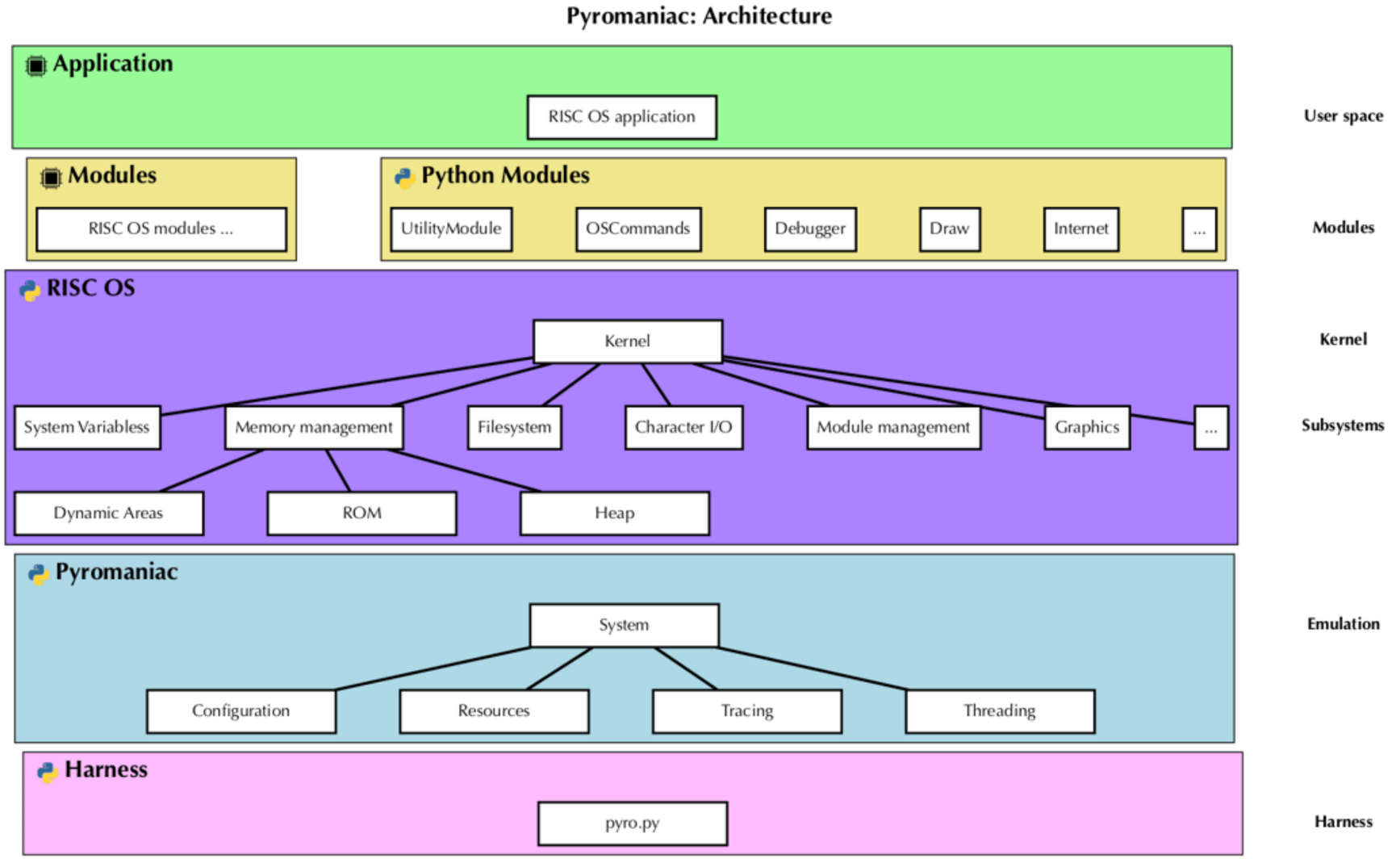
RISC OS Pyromaniac

What makes up Pyromaniac? (1)

- Written in Python.
- Uses Unicorn (a QEmu derived package) for emulating 32bit ARM code.
- All other packages are optional.
  - Disassembly - needs `capstone`.
  - Graphics - needs `python-cairo`.
  - UI - needs `wxpython` or `gtk+3`.
  - Networking - more featured with `netifaces`.
  - Clipboard - interaction with system with `pyperclip`.
  - Sound - needs `python-rtmidi`.

RISC OS Pyromaniac

What makes up Pyromaniac? (2)



RISC OS Pyromaniac

How is it different from other systems?

RISC OS emulation:

- RPCEmu, ArcEm - Hardware emulators.
- Riscose - OS interface replacement.
- Amethyst - ARM unit testing tool.
- Linux Port - Hardware / interface replacement.

Other systems:

- Wine - OS interface replacement.
- Docker - System isolation.
- Rosetta - Dynamic recompilation.

RISC OS Pyromaniac

How does it compare to a bare Operating System?

Has many of the same things:

- Address space management; memory allocation.
- System calls from applications.
- Heap management.
- I/O management.
- Device drivers.

But some are missing:

- Page table management.
- Hardware interrupts.
- Memory mapped devices.

RISC OS Pyromaniac

What does it mean?

- A command line only version of RISC OS.
- A RISC OS which runs 32bit ARM binaries, on Windows, macOS, or Linux.
- A reimplementation, which uses none of the code that went before.
- Focused on being able to test software and diagnose issues more easily.

RISC OS Pyromaniac

What does it mean?

- A command line only version of RISC OS.
- A RISC OS which runs 32bit ARM binaries, on Windows, macOS, or Linux.
- A reimplementation, which uses none of the code that went before.
- Focused on being able to test software and diagnose issues more easily.

Tech: RISC OS Pyromaniac, able to run RISC OS programs on other systems!

RISC OS Pyromaniac

Command line only?

- Command line is the primary interface.
- Graphics implementations exist - either 'headless' or using a window showing the screen - but command line is where it excels.
- For testing, you largely want to be able to exercise things without UI interactions, at least for the lower level tests.

RISC OS Pyromaniac

No graphics, then?

Different parts of the system:

- VDU - VDU4, text output.
- Graphics - VDU5, OS_Plot, Draw, Font.
- Frame buffer - Bitmap of the screen.
- UI - How you see the VDU and Graphics systems.

RISC OS Pyromaniac

No graphics, then?

Different parts of the system:

- VDU - VDU4, text output: **Well supported**
- Graphics - VDU5, OS_Plot, Draw, Font.
- Frame buffer - Bitmap of the screen.
- UI - How you see the VDU and Graphics systems.

RISC OS Pyromaniac

No graphics, then?

Different parts of the system:

- VDU - VDU4, text output: **Well supported**
- Graphics - VDU5, OS_Plot, Draw, Font: **Well supported, but no sprites**
- Frame buffer - Bitmap of the screen.
- UI - How you see the VDU and Graphics systems.

RISC OS Pyromaniac

No graphics, then?

Different parts of the system:

- VDU - VDU4, text output: **Well supported**
- Graphics - VDU5, OS_Plot, Draw, Font: **Well supported, but no sprites**
- Frame buffer - Bitmap of the screen: **Nope**
- UI - How you see the VDU and Graphics systems.

RISC OS Pyromaniac

No graphics, then?

Different parts of the system:

- VDU - VDU4, text output: **Well supported**
- Graphics - VDU5, OS_Plot, Draw, Font: **Well supported, but no sprites**
- Frame buffer - Bitmap of the screen: **Nope**
- UI - How you see the VDU and Graphics systems: **wxWidgets and GTK**

RISC OS Pyromaniac

No graphics, then?

Different parts of the system:

- VDU - VDU4, text output: **Well supported**
- Graphics - VDU5, OS_Plot, Draw, Font: **Well supported, but no sprites**
- Frame buffer - Bitmap of the screen: **Nope**
- UI - How you see the VDU and Graphics systems: **wxWidgets and GTK**

What works...

- The VDU system, and the graphics system work, mostly.
- VDU and graphics are complex so not everything works as it does in RISC OS Classic.
- Not all of it works as documented - after all not all of it is documented!

RISC OS Pyromaniac

How do you use it? (1)

Command line invocation:

```
charles@laputa ~/pyromaniac> ./pyro.py --load-internal-modules --command 'gos'
Supervisor

*fx0

Error: RISC OS 7.16 (03 Oct 2020) [Pyromaniac 0.16 on Darwin/x86_64] (Error number &f7)
*time
Fri,09 Oct 2020 23:00:04
*quit
charles@laputa ~/pyromaniac>
```

RISC OS Pyromaniac

How do you use it? (2)

Running RISC OS programs:

```
charles@laputa ~/pyromaniac> echo '10PRINT "Hello world"' > myprog,fd1
charles@laputa ~/pyromaniac> ./pyro.py --load-internal-modules --load-module
modules/BASIC,ffa --command myprog
Hello world
charles@laputa ~/pyromaniac>
```

RISC OS Pyromaniac

Graphics demo!

RISC OS Pyromaniac

Graphics demo!

Graphics features:

- Fonts.
- DrawFiles.
- Images.
- Screen bank flipping.
- Mouse pointer.

Others:

- Key input

RISC OS Pyromaniac

Graphics demo! (presentation)

Tech:

- Slide presentation system.
- Markdown parser.
- FontMap for font remappings.
- WebColours module for colour parsing.
- ImageFileRender for general image rendering, using DrawFile for vectors.

RISC OS Pyromaniac

Features - What works?

- System - Runs 32bit modules, utilities and applications.
- Interaction - Interacts with the host as its primary interface
- Video - Pretty good VDU and graphics support GTK/WxWidgets, or snapshots of state.
- Sound - SoundChannels mapped through MIDI
- Filesystem - Host filesystem by default, using `,xxx` filename convention.
- Network - Internet module provides limited support for IPv4 and IPv6 networking.
- Compatibility - Many simple programs work, if their support modules are loaded.

RISC OS Pyromaniac

Features - What doesn't work?

- Desktop - Not supported
- Filesystems - No registration of filesystems
- Sound - No wave output
- Graphics - No frame buffer, No Sprites, No true colour modes
- Many other things

RISC OS Pyromaniac

Networking

- Internet module supplied, using host interfaces.
  - Supports `AF_INET`, `AF_INET6`, `AF_UNIX`.
  - Many ioctls are supported, mapped to the host system.
- Resolver module provides IPv4 host name resolution
- EtherPyromaniac provides a DCI4 driver.
  - Provides a virtual network.
- EasySockets, which bypasses Internet.

Tech: Tap-Tun JSON server for Ethernet frames.

RISC OS Pyromaniac

Draw module (1)

- Draw module supplied.
- Can render through the Cairo path system.
- Classic DrawFile works - the 'Gerph' logo is a Drawfile.
- Classic Draw module can be used too.

RISC OS Pyromaniac

Draw module (2)

RISC OS Pyromaniac

FontManager

- FontManager module supplied.
- Can uses Cairo 'toy' fonts.
  - Can be configured to use any 'fontconfig' discovered fonts.
- Supports different alphabets, including UTF-8.

But also

- Classic FontManager works...
- ... if you disable bitmap generation - it just uses Draw.

RISC OS Pyromaniac

Configuration

- RISC OS Pyromaniac is highly configurable - over 240 directly configurable options, in 59 groups.
- Configuration can be on the command line or in configuration files.
- Example:
  - `./pyro.py --config modules.rominit_noisy=true --load-internal-modules --command gos`
  - `./pyro.py --config memorymap.rom_base=90000000 --load-internal-modules --command modules`

RISC OS Pyromaniac

Configuration files

```
%YAML 1.1
---
# Configuration for loading the ROM for RISC OS 5

debug:
  - modules
  - traceregionfunc
  - podules
  - swimisuse

config:
  podule.extensionrom1: ROMs/riscos5
  modules.rominit_noisy: true
  memorymap.rom_base: 0x8800000
  modules.unplug: extrom1:Podule,ParallelDeviceDriver,TaskWindow,SpriteExtend,SystemDevices,...

modules:
  internal: true
```

RISC OS Pyromaniac

Tracing and debugging

- Trace features:
  - Report all instructions.
  - Report basic block execution, function entries.
  - Report SWI entry and exit conditions.
  - Function, memory and SWI traps.
  - Exception and API misuse reports.
- Debug features:
  - Most modules have debug available.
  - Can be enabled at runtime (`*PyromaniacDebug +<option>`).

RISC OS Pyromaniac

Tracing code (1)

Tracing SWI arguments (`--debug traceswiargs`):

```
>GCOL 0, 255,192,0
 383f848: SWI     ColourTrans_SetGCOL
          => r0  = &00c0ff00   12648192  colour
             r3  = &00000100        256  flags
             r4  = &00000000          0  action
          <= r0  = &00000000          0  gcol
             r2  = &00000002          2  log2_bpp
             r3  = &00000000          0  corrupted
```

Tech: OSLib parser and templating system

RISC OS Pyromaniac

Tracing code (2)

```
$ pyro testcode/bin/word_time_string --debug trace
 700013c: ADR     r1, &07000174              ; -> [&00000000, &00000000,
                                                 &00000000, &00000000]
 7000140: MOV     r0, #&e                    ; #14
 7000144: MOV     r2, #0
 7000148: STRB    r2, [r1]                   ; R2 = &00000000, R1 = &07000174
 700014c: SWI     OS_Word
 7000150: SWI     OS_WriteS
 7000164: MOV     r0, r1                     ; R1 = &07000174
 7000168: SWI     OS_Write0
 700016c: SWI     OS_NewLine
Time string: Sun,06 Sep 2020 08:22:38
 7000170: MOV     pc, lr                     ; R14 = &04107fe0
 4107fe0: SWI     &FEED05
```

RISC OS Pyromaniac

Debugging

```
charles@laputa ~/demo> pyro --load-internal-modules --command gos --debug cli,clialias,osfscontrol
CLI: 'gos'
CLI alias: Wildcard 'Alias$gos' start read from None
Supervisor

*.
CLI: '.'
CLI alias: '.' expansion
CLI alias: Expanded to 'Cat '
CLI alias: Execute: Cat
CLI: 'Cat '
CLI alias: Wildcard 'Alias$Cat' start read from None
Catalogue directory '@'
Canonicalise filename '@' using pathvar 0L, path 0L
Read boot option of '$'
Dir. $ Option 02 (run)
Read directory 0
CSD  NoFileSystem:$too
Read directory 3
Lib. NoFileSystem:$
Read directory 2
URD  NoFileSystem:$
example/py  WR/WR     example/pyc WR/WR     wimperror   WR/WR
*
```

RISC OS Pyromaniac

What is it like to work with? (1)

- The Pyromaniac context is usually `ro`, containing...
  - registers (`ro.regs[#]`)
  - memory (`ro.memory[address]`)
  - configuration (`ro.config['group.option']`)
  - resource (`ro.resource['resource']`)
  - methods for execution (`ro.execute`, `ro.execute_with_error`)
  - trace functions (`ro.trace`)
  - the kernel (`ro.kernel`)
- The Pyromaniac layer is all about the lower level execution and setup of the system.

RISC OS Pyromaniac

What is it like to work with? (2)

- The RISC OS Kernel context is `ro.kernel`...
  - dynamic areas (`ro.kernel.da`, `ro.kernel.da_rma`. `ro.kernel.da_appspace`, ...)
  - vectors (`ro.kernel.vectors[#]`)
  - modules (`ro.kernel.modules`)
  - vdu and graphics system (`ro.kernel.vdu`, `ro.kernel.graphics`)
  - input and mouse (`ro.kernel.input`, `ro.kernel.mouse`)
  - filesystem (`ro.kernel.filesystem`)
  - system variables (`ro.kernel.sysvars[varname]`)
  - program environment (`ro.kernel.progenv`)
  - system APIs (`ro.kernel.api`)
- The Kernel object is always referenced explicitly from `ro`.

RISC OS Pyromaniac

What is it like to work with? (3)

```
"""
OS_ReadEscapeState implementation.
"""


from riscos import handlers
import riscos.constants.swis as swis

@handlers.swi.register(swis.OS_ReadEscapeState)
def swi_OS_ReadEscapeState(ro, swin, regs):
    """
    OS_ReadEscapeState

    <= C flag is set if an escape condition has occurred
    """

    regs.cpsr_c = ro.kernel.progenv.escape_condition
```

RISC OS Pyromaniac

What is it like to work with? (4)

Many commands are just a thin wrapper around a system call:

```
def cmd_rmload(self, args):
    """
    Syntax: *RMLoad <module-file> [<args>]
    """
    self.ro.kernel.api.os_module(modhand.ModHandReason_Load, args)
```

RISC OS Pyromaniac

What is it like to work with? (5)

Context handlers can be used to make memory allocation easy:

```
def cmd_time(self, args):
    """
    Syntax: *Time
    """
    with self.ro.kernel.da_sysheap.allocate(128) as time_string:
        time_string[0].word = 0
        self.ro.kernel.call_swi(swis.OS_Word,
                                rin={0: osword.OsWord_ReadRealTimeClock,
                                     1: time_string.address})
        self.ro.kernel.writeln(time_string.string)
```

RISC OS Pyromaniac

What is it like to work with? (6)

Context handlers can also preserve the output state:

```
def cmd_show(self, args):
    """
    Syntax: *Show [<variable>]
    """
    # Preserve and enable VDU paging
    with self.ro.kernel.api.vdupaging():
        # Enumerate and print variables
        for varname, vartype, value in self.ro.kernel.api.os_readvarval_enumerate(args):
            if vartype in (sysvars.TYPE_STRING, sysvars.TYPE_MACRO):
                # String returned parameters should have their value escaped GSTrans style
                value = self.ro.kernel.gstrans.escape(value,
                                                      escape_chars='|"<' if vartype != sysvars.TYPE_MACRO else '',
                                                      escape_control=True,
                                                      escape_topbit=True)

            suffix = ''
            if vartype == sysvars.TYPE_NUMBER:
                suffix = '(number)'
            elif vartype == sysvars.TYPE_MACRO:
                suffix = '(macro)'
            self.ro.kernel.writeln('%s%s : %s' % (varname, suffix, value))
```

RISC OS Pyromaniac

What is it like to work with? (7)

Exceptions can be trapped in a pythonic way:

```
def cmd_unset(self, args):
    """
    Syntax: *Unset <variable>
    """
    try:
        self.ro.kernel.api.os_setvarval_delete(args)
    except RISCOSError as exc:
        if exc.errnum != errors.ErrorNumber_VarCantFind:
            raise
        # Lack of a variable is not an error
```

RISC OS Pyromaniac

What good is it? (1)

- Writing my own software.
  - This presentation tool.
  - Other older components.
- Trying things out.
  - Sound system.
- Debugging other people's software!
  - https://asciinema.org/a/345766 shows an interactive session demonstrating that freeing the stack you're currently using may have bad effects.

RISC OS Pyromaniac

What good is it? (2)

```
Supervisor

*cobey:obey echosed
...
==== Begin exception report ====
Exception triggered: Exception 'Prefetch Abort'
  r0  = &a9a9a9a9, r1  = &a9a9a9a9, r2  = &00000000, r3  = &a9a9a9a9
  r4  = &00000191, r5  = &07001f28, r6  = &07008ac0, r7  = &00000001
  r8  = &07008aa8, r9  = &07007720, r10 = &07008d18, r11 = &a9a9a9a9
  r12 = &a9a9a9a9, sp  = &a9a9a9a9, lr  = &a9a9a9a9, pc  = &a9a9a9ac
  CPSR= &60000010 : USR-32 ARM fi ae qvCZn
Recently executed code:
    ---- Block &07006f4c, 1 instructions ----
     7006f4c: LDR     pc, &07007230             ; = &0382095c
    ---- Block &0382095c, 5 instructions ----
     382095c: {DA 'ROM', module 'SharedCLibrary'}
    Function: memset
     382095c: MOV     r12, sp                   ; Function: memset
     3820960: PUSH    {r0, r11, r12, lr, pc}
     3820964: SUB     r11, r12, #4
     3820968: SUBS    r2, r2, #4
     382096c: BMI     &038209E4
    ...
    ---- Block &038209a0, 4 instructions repeated 229 times ----
     38209a0: STMIA   r0!, {r1, r3, r12, lr}
     38209a4: STMIA   r0!, {r1, r3, r12, lr}
     38209a8: SUBS    r2, r2, #&20
     38209ac: BGE     &038209A0
    ...
    ---- Block &038209d4, 6 instructions ----
     38209d4: SUBS    r2, r2, #4
     38209d8: STRLT   r1, [r0], #4
     38209dc: STMGEIA r0!, {r1, r3}
     38209e0: SUBGE   r2, r2, #4
     38209e4: ADDS    r2, r2, #4
     38209e8: LDMDBEQ r11, {r0, r11, sp, pc}
==== End exception report ====

Error: Internal error: Abort on instruction fetch at &a9a9a9a8 (Error number &80000001)
*quit
```

RISC OS Pyromaniac

Problems...

- IRQs and timed events aren't handled well.
- Execution context is split between emulated system and Python code.
- Error handling is still a bit troublesome.
- Replaced writing device driver, with writing interface modules.

RISC OS Pyromaniac

Other technologies!

Tech:

- RISC OS Alphabets in Python Codecs - https://github.com/gerph/python-codecs-riscos
- Non-RISC OS editor syntax modes:
  - SublimeText syntax for RISC OS command files - https://github.com/gerph/sublimetext-riscoscommand-syntax
  - NanoRC syntaxes for some RISC OS file types - https://github.com/gerph/nanorc-riscos
- Tool for building hourglass modules - https://github.com/gerph/riscos-hourglass-maker
- Tests for RISC OS APIs, and a tool for testing tools - https://github.com/gerph/riscos-tests
- PRM-in-XML documentation system rework.
- Miscellaneous toolchain updates.
- Changelog management system - https://gitlab.gerph.org/gerph/changelog-management

RISC OS Pyromaniac

What does it run on?

- macOS (console, GTK, wxWidgets)
  - Also a dedicated application.
- Linux (console, GTK, wxWidgets)
  - Also within a docker container.
- Windows (console, wxWidgets) [native and under Wine, also docker wine-py]
  - Also a dedicated application.

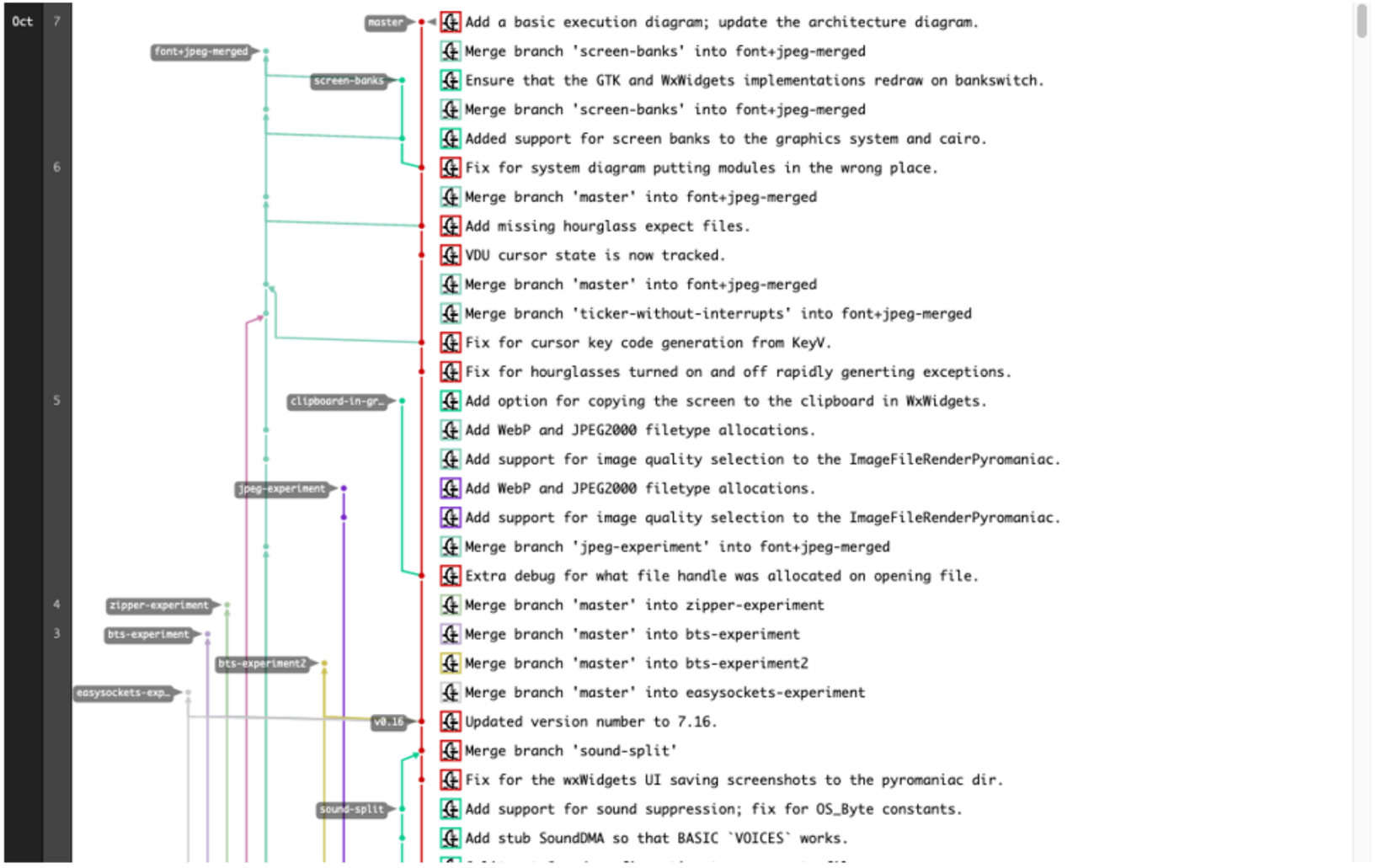
RISC OS Pyromaniac

“Releases”?

- Released once a month, just to myself.
  - October's version is 0.16.
  - Releases are a way to stop it being unusably 'half finished'.
  - Releases are a great incentive - I really have achieved a lot this month!
- Long lived development, for example...
  - Font Manager lived on a branch for about 6 months.
  - EasySockets is still on a branch.
  - PyromaniacGit is still be worked on.

RISC OS Pyromaniac

“Releases”?



5. Conclusion

Conclusion

Have I done what I set out to do?

Let's review what I saw as problems...

- Development on RISC OS is tedious
- RISC OS testing is awful
- RISC OS is awful for testing

Conclusion

Development on RISC OS is tedious

- Source control
- Cross compiling
- Managed development environments
- Automated testing
- Feature and regression testing
- Fleets of systems available

Conclusion

Development on RISC OS is tedious

- Source control - ***yup, using GitLab, PyromaniacGit***
- Cross compiling
- Managed development environments
- Automated testing
- Feature and regression testing
- Fleets of systems available

Conclusion

Development on RISC OS is tedious

- Source control - yup, using GitLab, PyromaniacGit
- Cross compiling - ***yup, Linux and macOS***
- Managed development environments
- Automated testing
- Feature and regression testing
- Fleets of systems available

Conclusion

Development on RISC OS is tedious

- Source control - yup, using GitLab, PyromaniacGit
- Cross compiling - yup, Linux and macOS
- Managed development environments - ***yup, docker, artifactory, applications***
- Automated testing
- Feature and regression testing
- Fleets of systems available

Conclusion

Development on RISC OS is tedious

- Source control - yup, using GitLab, PyromaniacGit
- Cross compiling - yup, Linux and macOS
- Managed development environments - yup, docker, artifactory, applications
- Automated testing - ***yup, build.riscos.online, and GitHub and GitLab builds***
- Feature and regression testing
- Fleets of systems available

Conclusion

Development on RISC OS is tedious

- Source control - yup, using GitLab, PyromaniacGit
- Cross compiling - yup, Linux and macOS
- Managed development environments - yup, docker, artifactory, applications
- Automated testing - yup, build.riscos.online, and GitHub and GitLab builds
- Feature and regression testing - ***yup, thousands of tests, some public***
- Fleets of systems available

Conclusion

Development on RISC OS is tedious

- Source control - yup, using GitLab, PyromaniacGit
- Cross compiling - yup, Linux and macOS
- Managed development environments - yup, docker, artifactory, applications
- Automated testing - yup, build.riscos.online, and GitHub and GitLab builds
- Feature and regression testing - yup, tests for the OS, and code coverage
- Fleets of systems available - ***well, no, not yet***

Conclusion

RISC OS Testing is awful

- RISC OS Pyromaniac has tests - about a thousand at present.
- Tests take about 18 minutes to run - and run on Linux and macOS in parallel.
- Code coverage hovers at around 65%.

Conclusion

RISC OS is awful for testing

- Clarification: RISC OS Classic is awful for testing.
- Heavily used as part of the development of the `present` tool.
- JFPatch itself is tested.
- BASIC module has some tests that run programs.

Conclusion

Could it be better?

- More APIs.
- Better handling of corner cases.
- Sprites (sigh).
- Back Trace Structures.
- Finish the pending branches - Windows, Zipper, EasySockets, Git, DCI4, ...
- Using it for actual testing - that was what it was for!
- So many other opportunities.

Conclusion

References

If you're wanting to know more, or review this talk, a site, https://pyromaniac.riscos.online/ has been created which contains support materials:

- Copies of these slides.
- Links to the technologies in these slides.
- Explanations of the CI examples.
- Development images and screenshots.
- Documentation from Pyromaniac (features, changelogs, configuration info).

There's also a demonstration site: http://shell.riscos.online/.

Conclusion

Am I happy?

You can make whatever judgements you like!

6. Questions

Info site: https://pyromaniac.riscos.online/
Shell: http://shell.riscos.online/