Behaviorally Accurate Simulator for Multifunction Printers and Scanners

Alexander Pevzner
(pzz@apevzner.com)

Title image source and licensing information:
https://commons.wikimedia.org/wiki/File:BESM-6_ACPY.jpg

Other images are generated with Craiyon and Kandinsky AIs.

Alexander Pevzner

- OpenPrinting member since 2020
- Write system software in C and Go
- Author of the ipp-usb and sane-airscan packages, used everywhere
- @alexpevzner at GitHub
- Now work with team that moves the entire 20-million city from Windows to Linux

Agenda

- Why we need MFP simulator?
- What is behaviorally accurate simulation?
- Scope of this project
- MFP is a complex thing. How to make models simple?
- Helper tools
- Current state
- Side projects

Alexander Pevzner (pzz@apevzner.com)

Why We Need an MFP Simulator

- Printers are large, heavy, and expensive.
- Maintaining a representative collection is difficult, even for corporations.
- Printing and scanning software is complex.
- Development, troubleshooting, and support require reproducibility.
- Accurate simulation would be a solution.

Alexander Pevzner (pzz@apevzner.com)

Scope of This Project

- Standard printing protocol (IPP).
- Standard scanning protocols (eSCL and WSD).
- IPP over USB simulation.
- DNS-SD and WS-Discovery advertising.
- Probably, semi-accurate implementation of legacy printing protocols, for completeness.
- Proprietary protocols not implemented and not planned.

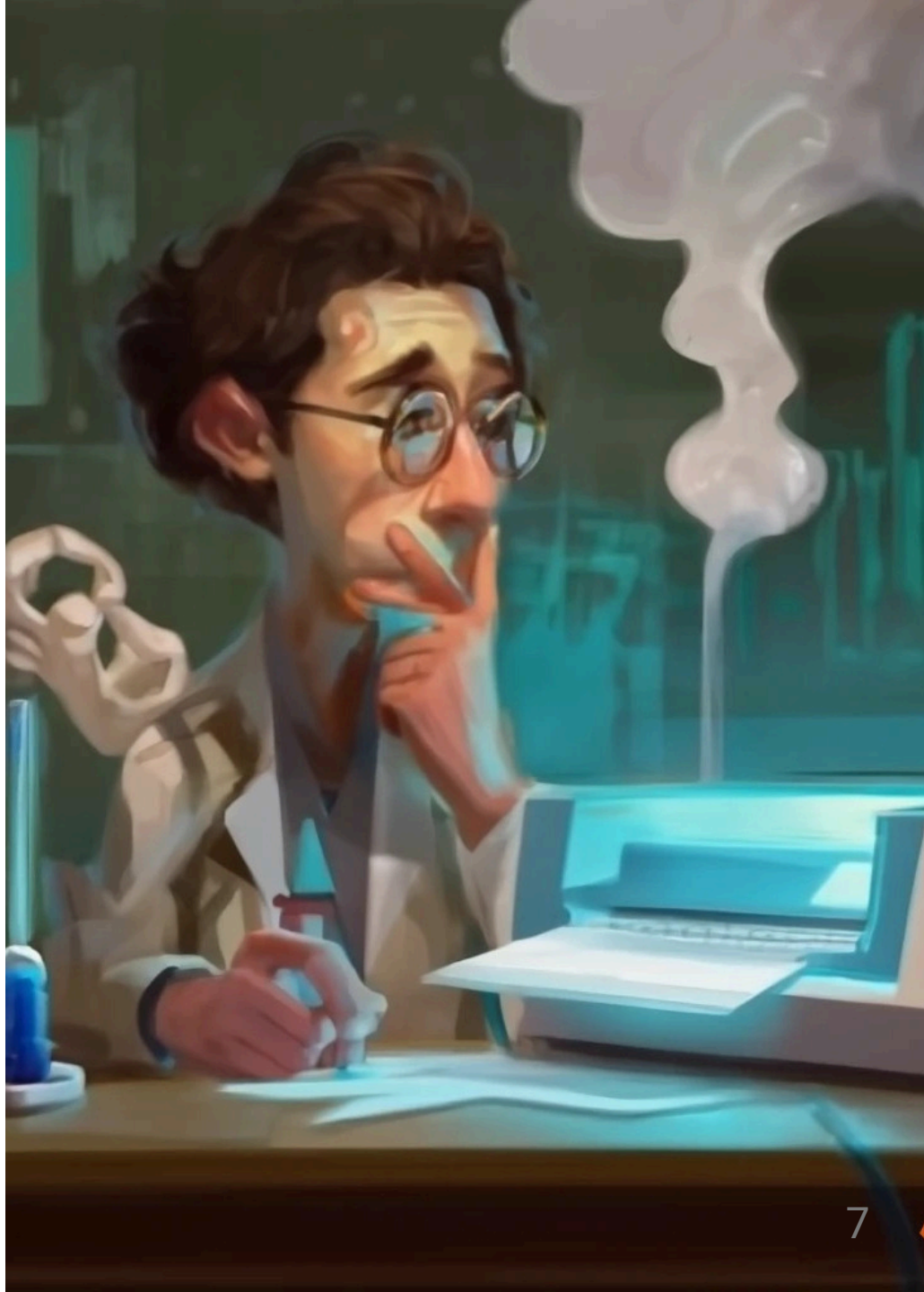
Alexander Pevzner (pzz@apevzner.com)

What is Behaviorally Accurate?

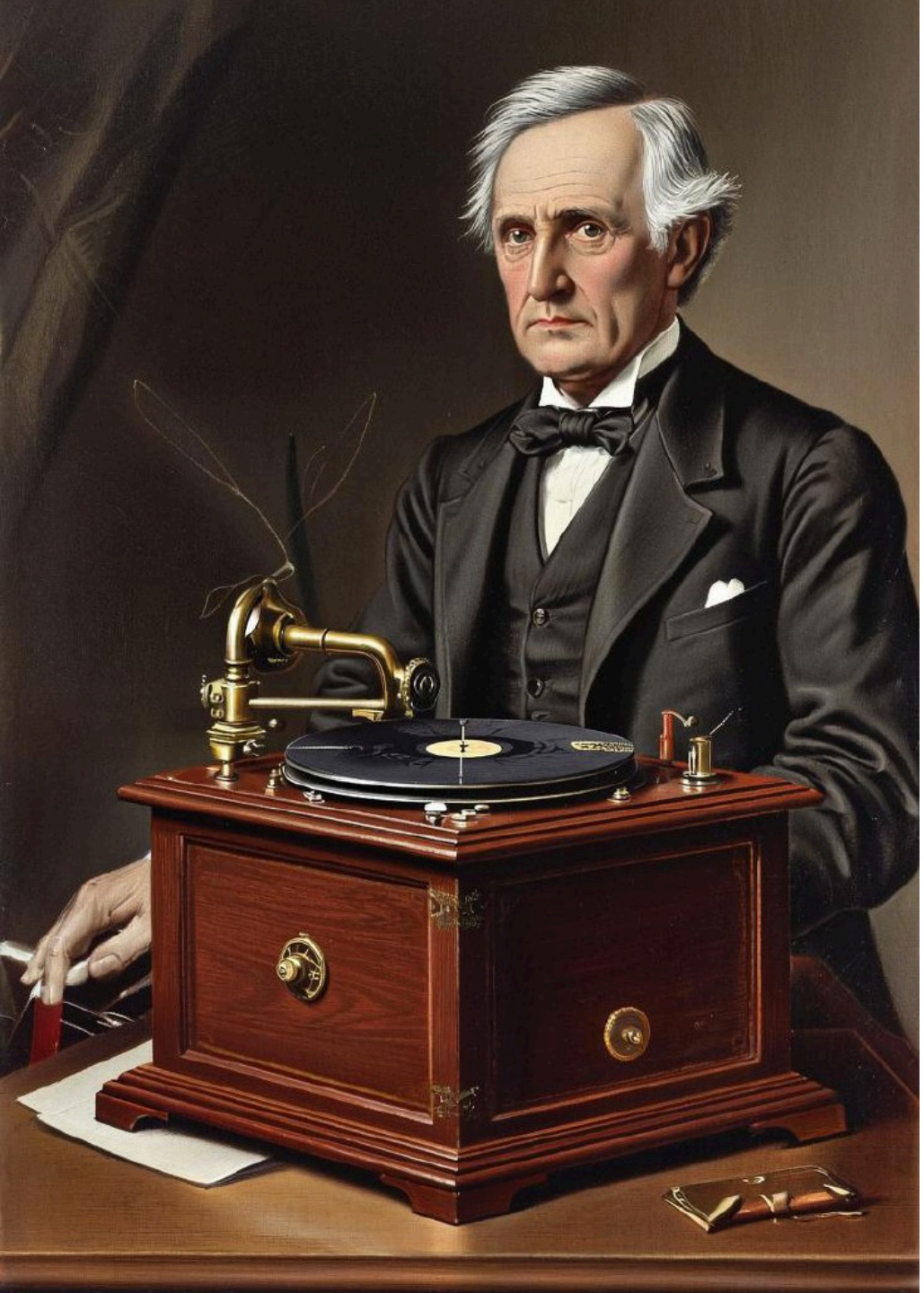
- For the standard protocols, behavior is defined by the specification and printer attributes/scanner capabilities.
- But this is not enough. Real hardware often deviate from these specifications.
- Model must define not only device parameters but the details of the actual device behavior, including bugs.

How accurate can be our models?

- The actual limiting factor is our detailed knowledge of the particular model.
- But 100% accuracy is not required. It is enough to reproduce essential details.
- In many cases it is enough just to reproduce the problem.



Alexander Pevzner (pzz@apevzner.com)



Creation of models

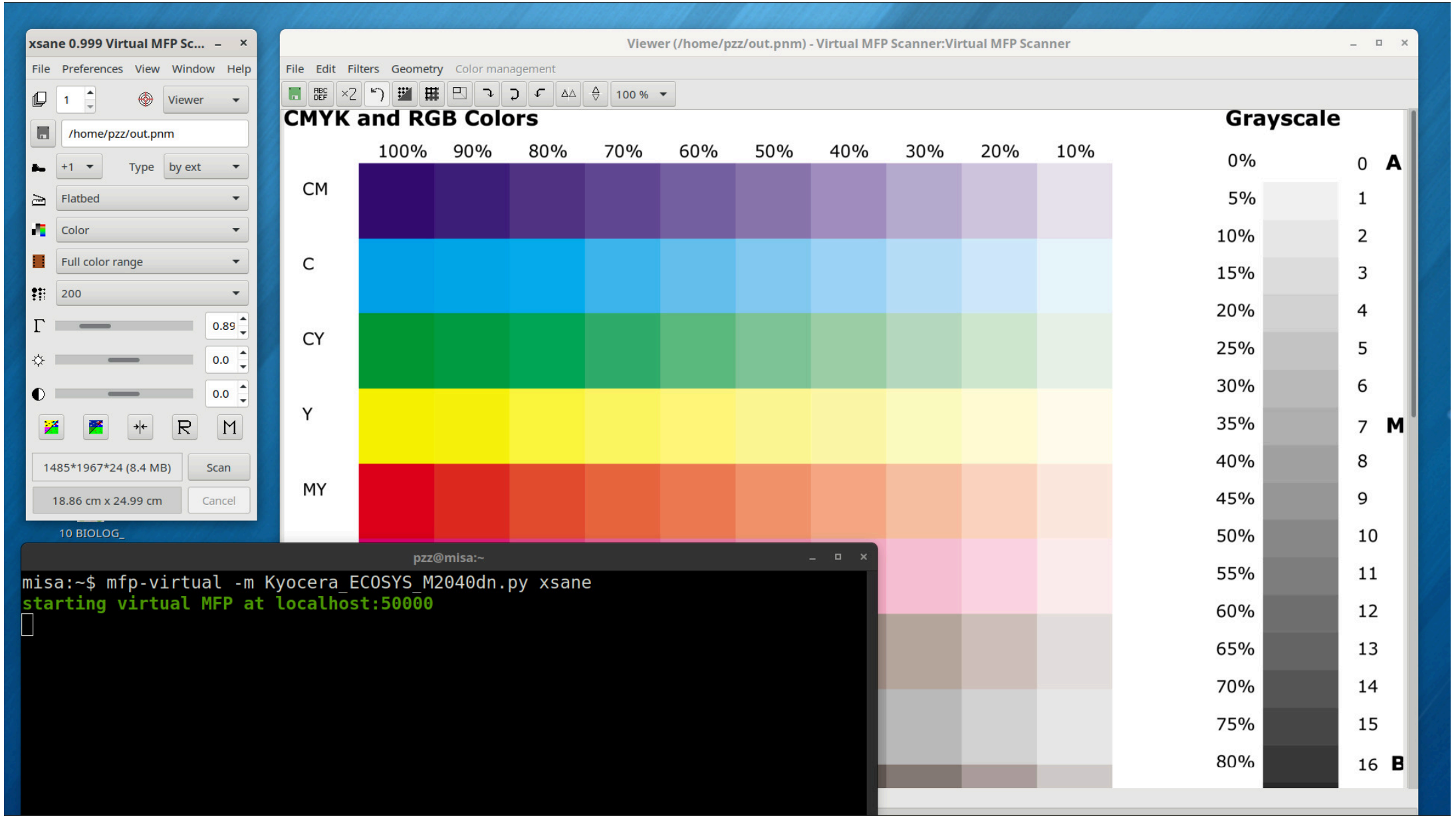
- A base model is a "recording" of a device's core capabilities — a simple collection of its printer attributes and scanner capabilities,
  made without behavioral details.
- The `mfp-model` tool records these baseline models automatically from real hardware.
- The `mfp-virtual` tool playbacks a model to emulate the original hardware.

Live example

This is the fragment of the `Kyocera ECOSYS M2040dn` eSCL scanner model, automatically generated with the `mfp-model` tool.

```
# eSCL scanner parameters:
escl.caps = {
  'Version': '2.62',
  'MakeAndModel': 'Kyocera ECOSYS M2040dn',
  'SerialNumber': 'VCF9192281',
  'Uuid': UUID('4509a320-00a0-008f-00b6-002507510eca'),
  'AdminUri': 'https://KM7B6A91.local/airprint',
  'IconUri': 'https://KM7B6A91.local/printer-icon/machine_128.png',
  'Platen': {
    'PlatenInputCaps': {
      'MinWidth': 118,
      'MaxWidth': 2551,
      'MinHeight': 118,
      'MaxHeight': 3508,
      'SupportedIntents': ['Document', 'TextAndGraphic', 'Photo', 'Preview'],
      'SettingProfiles': [
        {
          'ColorModes': ['BlackAndWhite1', 'Grayscale8', 'RGB24'],
          'DocumentFormats': ['image/jpeg', 'application/pdf'],
          'SupportedResolutions': [
            {
              'DiscreteResolutions': [
                {'XResolution': 200, 'YResolution': 100},
                {'XResolution': 200, 'YResolution': 200},
                {'XResolution': 200, 'YResolution': 400},
                {'XResolution': 300, 'YResolution': 300},
                {'XResolution': 400, 'YResolution': 400},
                {'XResolution': 600, 'YResolution': 600}
              ]
            }
          ]
        }
      ],
      'FeedDirections': ['ShortEdgeFeed', 'LongEdgeFeed']
    }
  },
  'Adf': {
    'AdfSimplexInputCaps': {
      'MinWidth': 591,
      'MaxWidth': 2551,
```

Alexander Pevzner (pzz@apevzner.com)


xsane 0.999 Virtual MFP Sc...
File Preferences View Window Help
Viewer
/home/pzz/out.pnm
Type by ext
Flatbed
Color
Full color range
200
0.89
0.0
0.0
1485*1967*24 (8.4 MB)
Scan
18.86 cm x 24.99 cm
Cancel
Viewer (/home/pzz/out.pnm) - Virtual MFP Scanner:Virtual MFP Scanner
File Edit Filters Geometry Color management
100 %
CMYK and RGB Colors
100% 90% 80% 70% 60% 50% 40% 30% 20% 10%
CM
C
CY
Y
MY
Grayscale
0% 5% 10% 15% 20% 25% 30% 35% 40% 45% 50% 55% 60% 65% 70% 75% 80%
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
A
M
B
pzz@misa:~
misa:~$ mfp-virtual -m Kyocera_ECOSYS_M2040dn.py xsane
starting virtual MFP at localhost:50000

Adding behavior details

- Using auto-recorded model, we can reproduce the idealized MFP behavior: defined by model's parameters and implemented according to specifications.
- Now lets add some hardware-specific behavior details.

The model language

- The model itself is a Python script.
- Printer attributes and scanner capabilities are defined as Python dictionaries.
- A set of Python hooks is provided to modify the model's behavior.
- All hooks are optional; write only the required ones.
- The simulator itself is written in Go but contains an embedded Python interpreter.

Alexander Pevzner (pzz@apevzner.com)

The practical case

- One of the scanners I had to implement the workaround for offers the JPEG/PNG image support.
- `sane-airscan` always prefers PNG if available, because it is lossless.
- However, this device actually sends a JPEG, even when PNG is requested.
- This caused decode errors, forcing me to add automatic format detection to `sane-airscan`.

- The `escl_onScanJobsRequest` hook in the model file can modify the eSCL scan request (represented as Python dictionary).
- We can see from the `sane-airscan` log that the received image format changed from PNG to JPEG.
- As simple as that; only few lines on Python required.

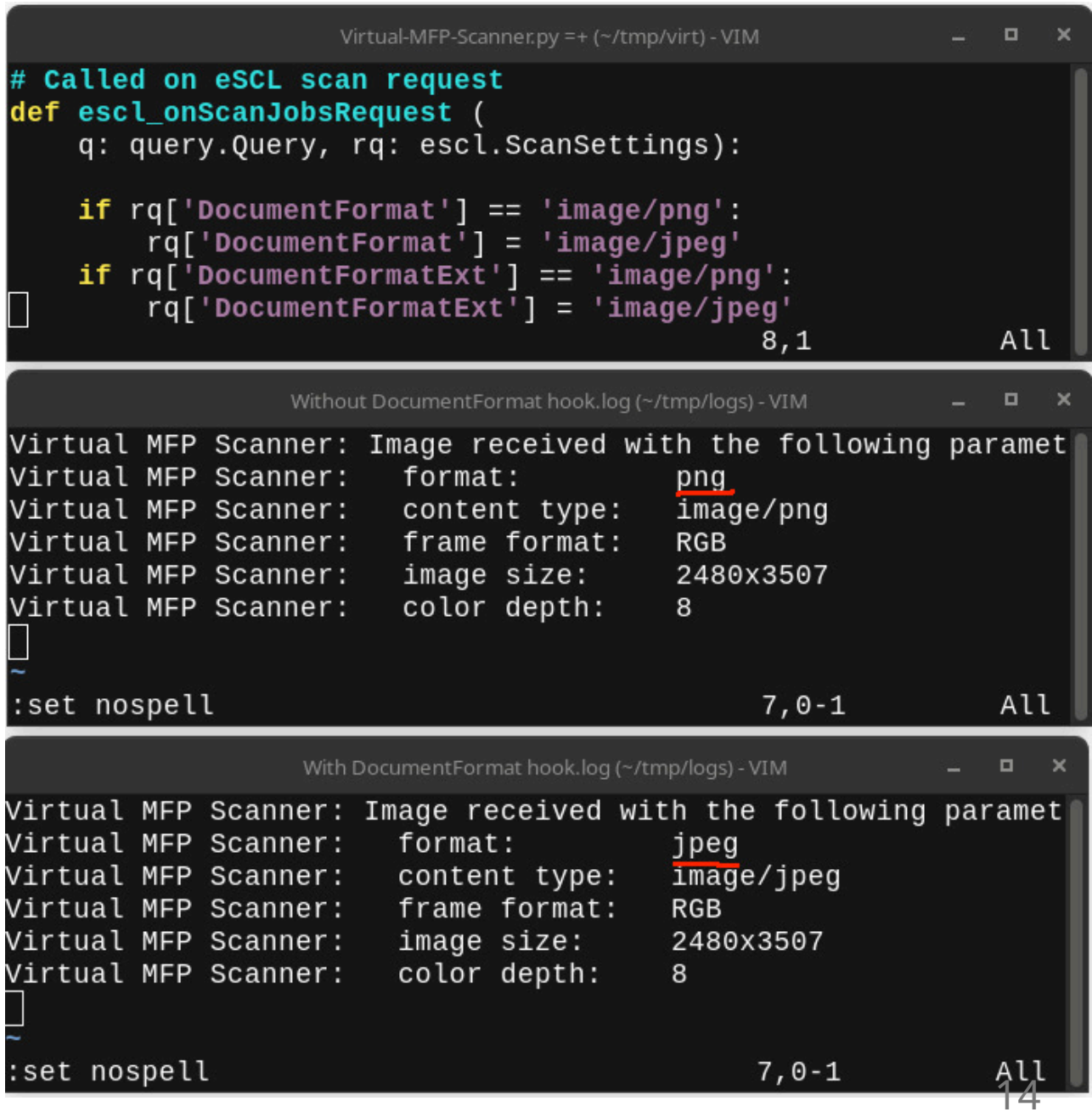
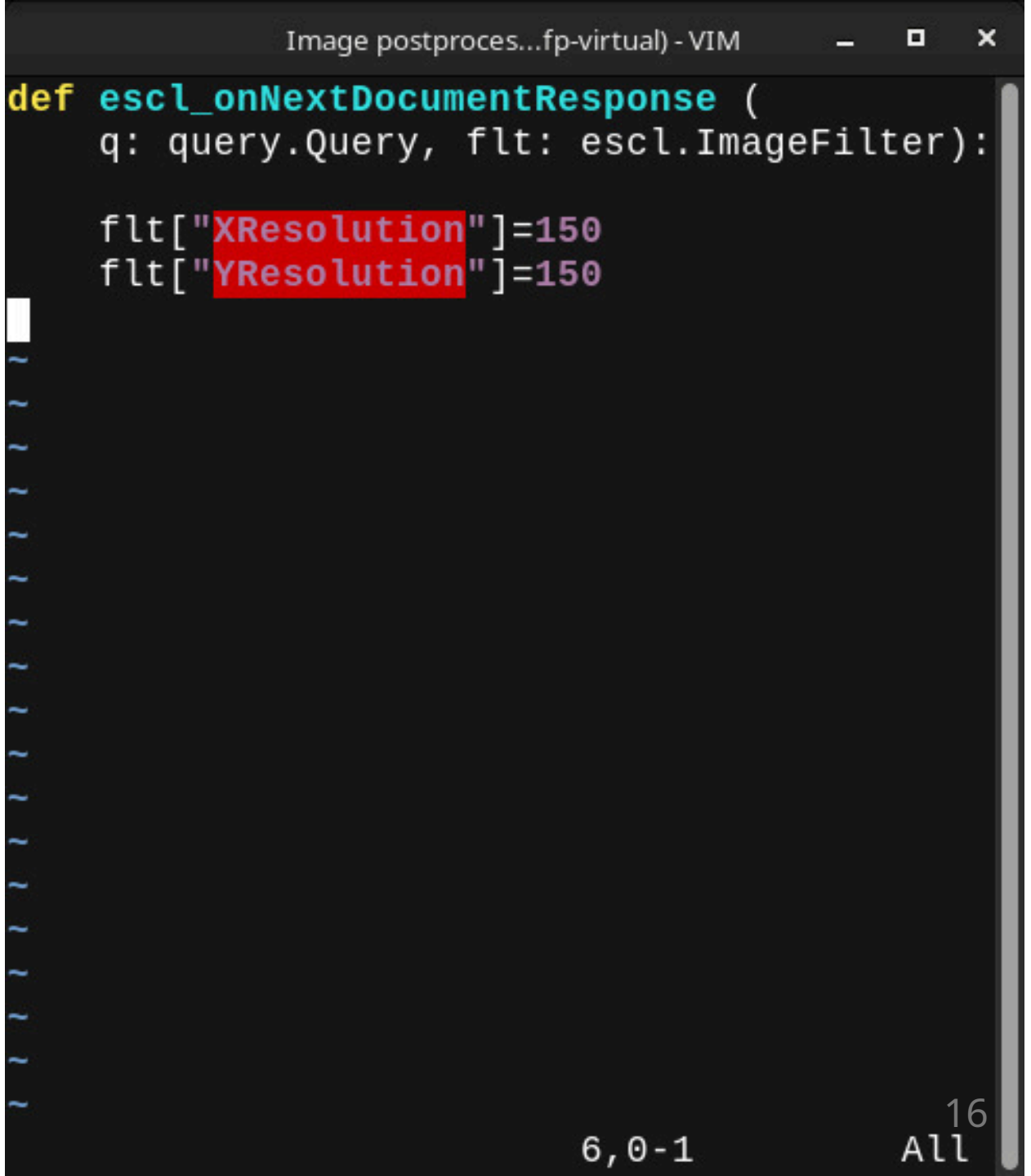


Image filtering

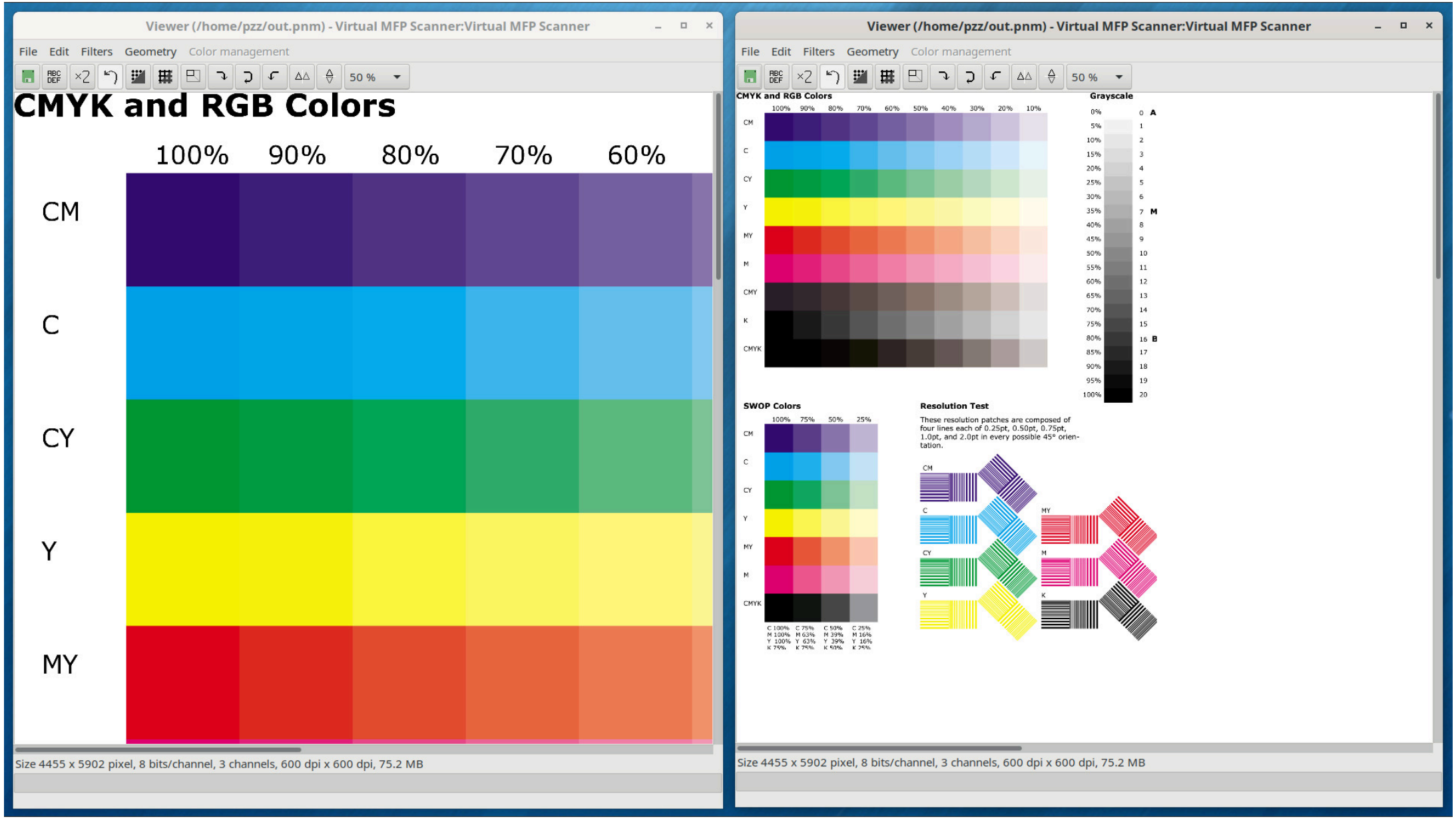
- We also have image filtering pipeline, integrated into the simulator.
- Scanned image can be resampled to change resolution, cropped, color mode can be changed etc.
- This is useful to emulate firmware bugs and to test our drivers.

Image filtering: try it

- `escl_onNextDocumentResponse` called when image is ready.
- It can set image post-processing parameters.
- We request image resampling from the original 600x600 to 150x150 DPI.
- As simple as that.
- Next slide shows it live.



Alexander Pevzner (pzz@apevzner.com)

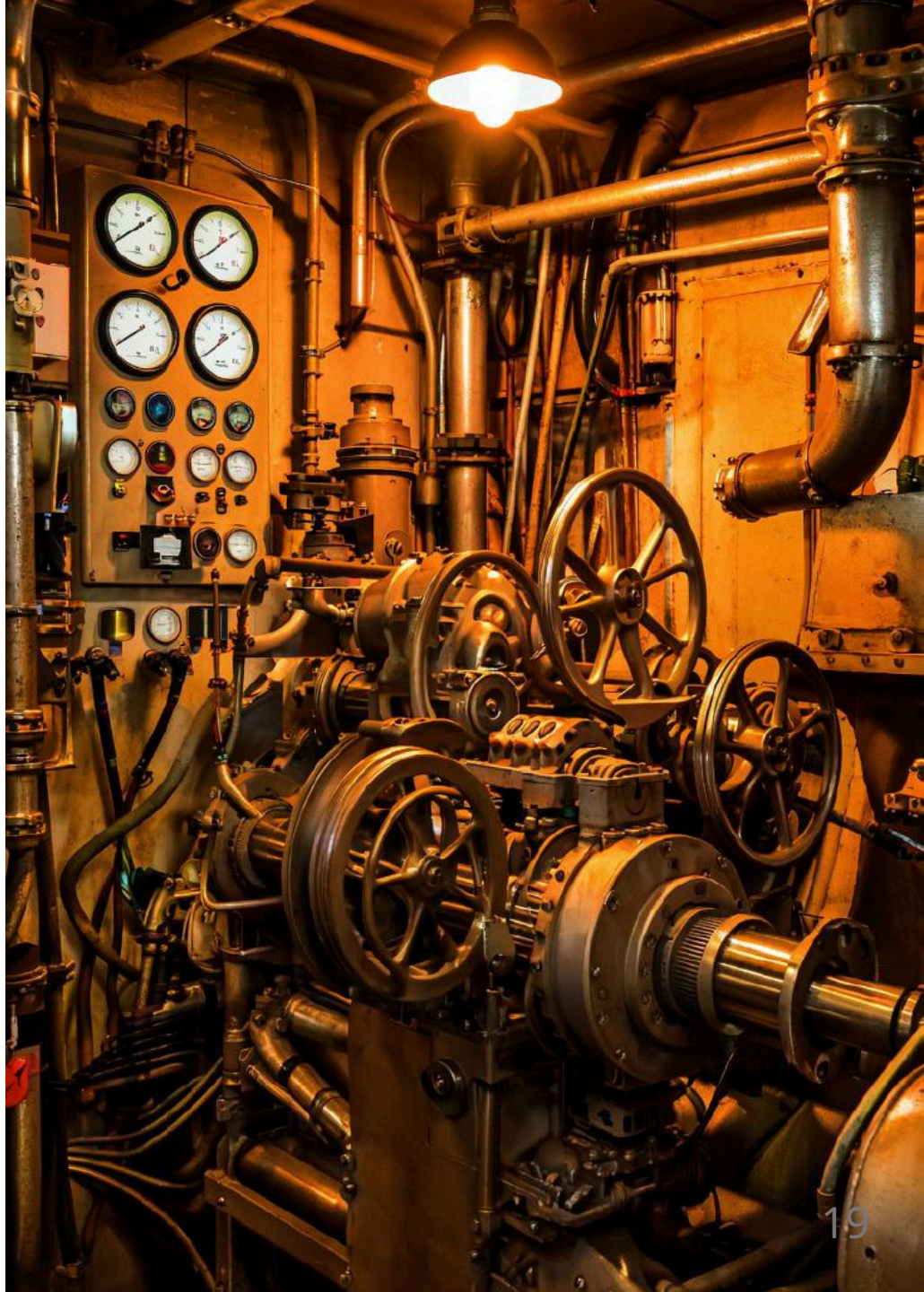

Viewer (/home/pzz/out.pnm) - Virtual MFP Scanner:Virtual MFP Scanner
File Edit Filters Geometry Color management
50 %
CMYK and RGB Colors
100% 90% 80% 70% 60%
CM
C
CY
Y
MY
Size 4455 x 5902 pixel, 8 bits/channel, 3 channels, 600 dpi x 600 dpi, 75.2 MB
Viewer (/home/pzz/out.pnm) - Virtual MFP Scanner:Virtual MFP Scanner
File Edit Filters Geometry Color management
50 %
CMYK and RGB Colors
Grayscale
SWOP Colors
Resolution Test
These resolution patches are composed of four lines each of 0.25pt, 0.50pt, 0.75pt, 1.0pt, and 2.0pt in every possible 45° orientation.
Size 4455 x 5902 pixel, 8 bits/channel, 3 channels, 600 dpi x 600 dpi, 75.2 MB

Here's what we have

- A simulator core that provides an "ideal," fully standards-compliant base simulation of a printer or scanner.
- Model parameters that define the device in protocol-specific terms.
- Simple Python hooks that can modify any aspect of the device's behavior.
- Useful helper tool for models creation.
- This architecture makes our simulator both simple to use and powerful.

Under the Hood

- The core simulator is a set of Go libraries that provide a generic and comprehensive implementation of several protocols (such as IPP, eSCL, WSD etc).
- Applications add a command-line interface (CLI) to access this functionality.
- This codebase can serve as a foundation for other projects, not necessarily limited to emulation



The Proxy Mode

- Another useful component of this project is the `mfp-proxy`.
- It implements an IPP/eSCL/WSD proxy.
- Transit traffic can be captured (sniffed).
- Device models can be applied to the real, proxied devices, effectively modifying their characteristics or behavior.

Side Projects

- This is a large project, with about 47K lines of Go code and 26K lines of tests.
- During development, several interesting sub-projects were created which may eventually have a life of their own.
- Here, I will briefly outline the most interesting of them.

Go Avahi Bindings (cgo)

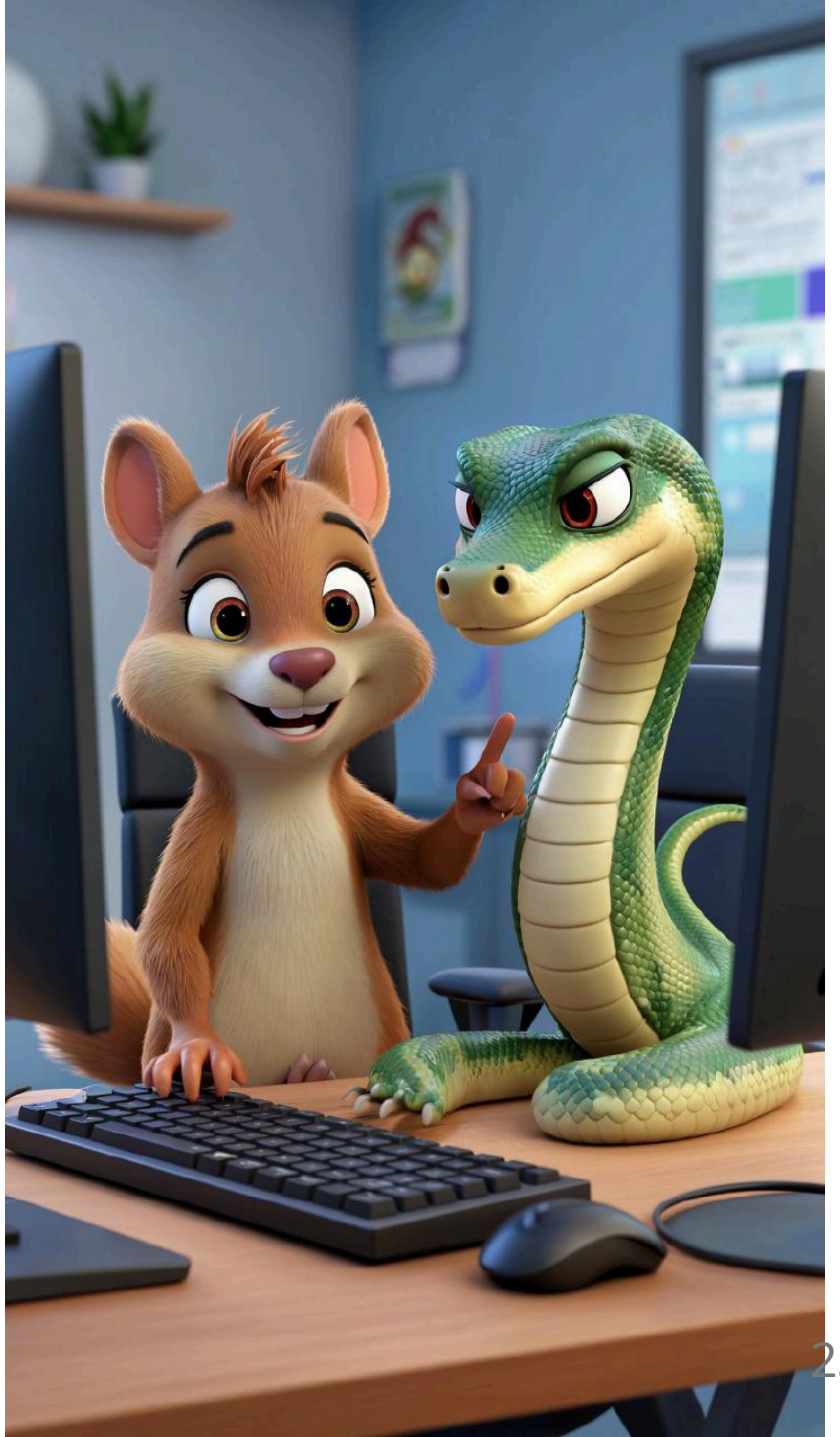
Complete, idiomatic Go bindings for the Avahi client library.

- As close to C API as possible.
- Idiomatic Go: Event handling via channels, not callbacks.
- Comprehensive documentation with many nuances. Useful even for C programmers.
- Moved into the separate project.

Go Binding for CPython

Distinguishing features:

- Links against the generic libpython3.so, not a version-specific libpython3.NN.so.
- Uses only CPython stable API.
- Tolerant to minor Python version upgrades without requiring a rebuild.
- Automatic garbage collection of Python objects on the Go side.
- Currently part of main project (at the /cpython directory).

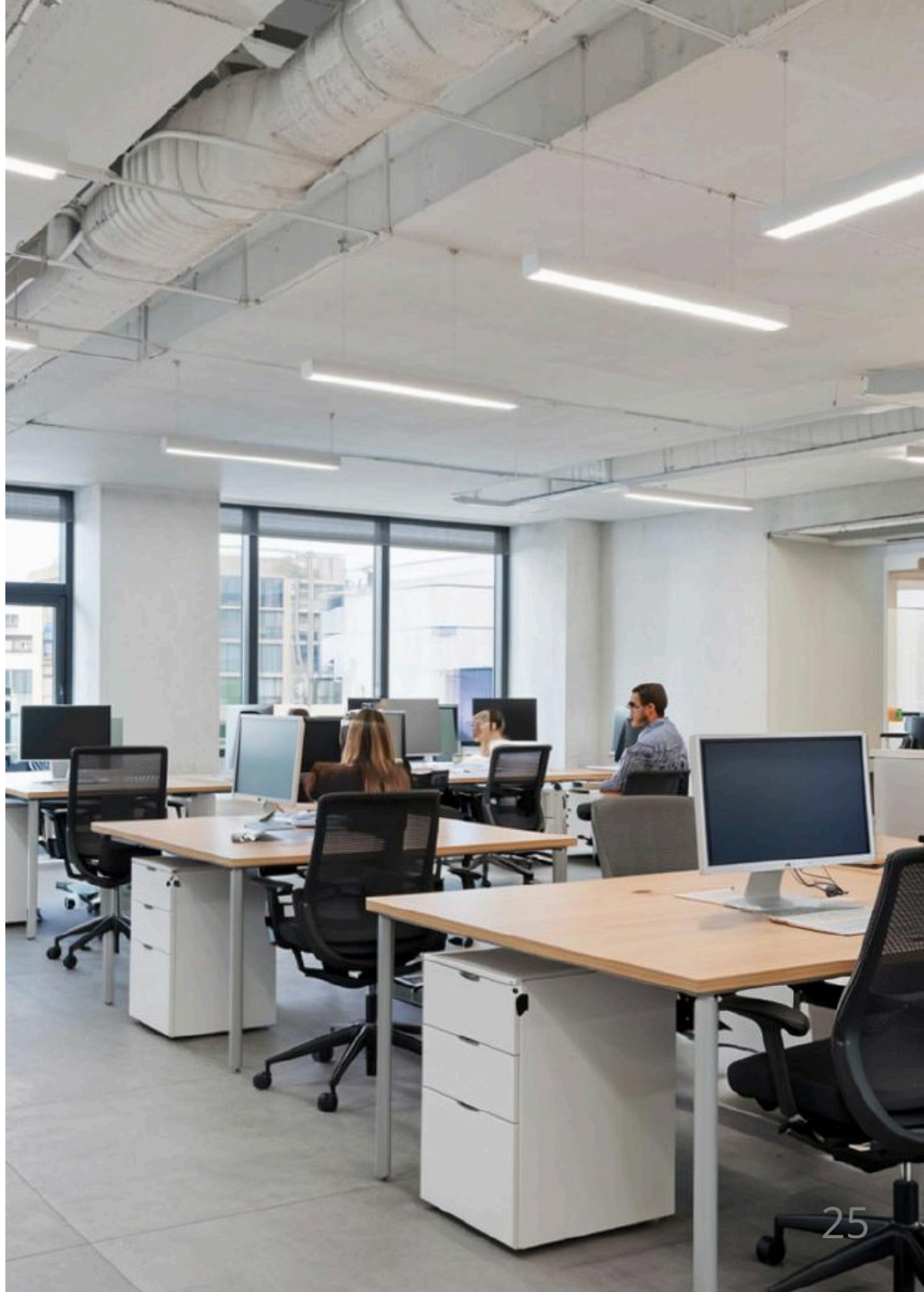


Current State

- This project is work in progress.
- Most Complete: eSCL scanner simulator.
- In Development: WSD scanner simulator (by Yogesh Singla, yogesh1801 at GitHub)
- Partially Done: IPP support.
- Proof-of-Concept: IPP-over-USB simulation (needs integration).
- Not Started: DNS-SD & WSD advertising.

Future Plans

- Project already used for the `sane-airscan` development and printing troubleshooting.
- I hope to finish major parts till the end of this year.
- Testing framework for running automated tests on simulated hardware.
- Integration with the image evaluation framework made by Sanskar Yaduka.



Alexander Pevzner (pzz@apevzner.com)

That's All for Now

Thank you for your time and attention!

I'm happy to answer any questions.

Alexander Pevzner (pzz@apevzner.com)

Contact Us

- The project page: https://github.com/OpenPrinting/go-mfp
- Author's e-mail: pzz@apevzner.com
- Author's Telegram: https://t.me/a_pevzner
- OpenPrinting Telegram channel: https://t.me/+RizBbjXz4uU2ZWM1