NETWORKED SCOPES AND MACOS

Scope

The discussion below relates specifically to the Keysight MSO-X 3034A oscilloscope, but I suspect it will apply just as well to other Keysight scopes and perhaps more generally. I'd be interested to hear about other instruments.

On the computer end, although I've written this with MacOS in mind, I expect it would work on Linux without modification.

Introduction

Lots of modern test instruments have an Ethernet jack on the back, which means we can connect them to the LAN and control them remotely. Obvious tasks include:

- remote control;
- data logging;
- automation.

For the MacOS user though, much of the online information seems to be Windows specific, which is a nuisance. Further, although software libraries exist to communicate with these devices they seem a bit unwieldy.

For someone using lots of different devices, or who needs a solution which is robust in the face of errors and bizarre happenings, these formal, well-tested systems make sense.

However, more casual approaches seem to work well on MacOS.

Given that most of this is text based, Perl seems a convenient choice if we want to automate things. Rather than using port 5024 though, it's better to use port 5025 which offers an interface without human-helpful prompts.

Grabbing a screen dump

As an example task, let's dump an image of the oscilloscope's screen. If you just want the code, grab it from GitHub.

The key, and most difficult, task is to find a suitable command. After some searching, I found :DISP:DATA? which is described on page 306.

To get an image of the screen in PNG format, all we have to send is:

```
DISP:DATA? PNG
```

The data returned are in IEEE-488.2 binary block data format.

It seems convenient to derive a new class from `IO::Socket::INET` to handle this:

```
package MSO;

use base qw(IO::Socket::INET);

sub get_ieee_binary_block
  {
    my $io = shift;

    my $header;
    read($io, $header, 2);

    # length of length
    my ($m) = ($header =~ /^#(\d)$/)
  or die "Unable to parse header A: $header, ";

    # length of data
    read($io, $header, $m);
    my ($n) = ($header =~ /^(\d+)$/)
  or die "Unable to parse header B: $header, ";

    # data
```

```
    my $data;
    read($io, $data, $n);

    # end-of-line
    $io->getline;

    return $data;
  }
```

Which we can call thus:

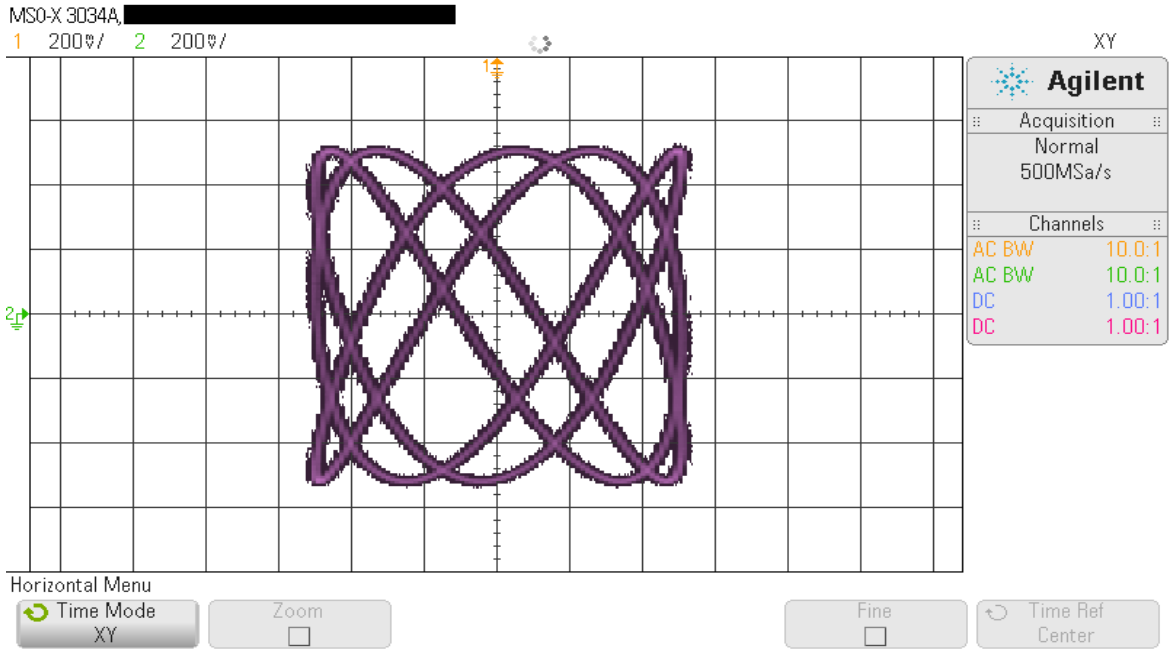
```
my $addr = 'mso.local';

my $mso = MSO->new(PeerAddr => $addr, PeerPort => 5025)
  or die "Unable to open MSO connection, ";

$mso->print("DISP:DATA?\n");
my $data = $mso->get_ieee_binary_block;

open(my $fh, '>', 'foo.png');
print {$fh} $data;
```

Here’s the result:



As you see, the code is pretty simple. There might be some sense in abstracting the binary block handling into a library, and perhaps wrapping `print` and `getline` to facilitate tracing, but I’ve not yet done that.

Source: http://www.mjoldfield.com/atelier/2015/06/scope-fun.html