Record screenshots and create PDF files

Egg, June 12, 2024: *Artificial intelligence (AI) seems to be everywhere at the moment. Microsoft is currently gaining some notoriety with the new Recall service announced for Windows 11, which is intended to become the user’s memory on their computer, so to speak. Screen shots are taken every x seconds. This data is processed in the background in an AI “fair” way so that later searches are possible. Capt2PDF is a new tool for Linux. Unlike Microsoft Recall, Capt2PDF works on the local desktop and the user retains control of the data.*

That’s why Capt2PDF is now available for the desktop

There are two basic approaches to recording screen content. OBS (Open Broadcasting System) is currently the most commonly used. This involves creating movie files (including sound) of screen activity. Text and speech recognition can be used to create searchable PDF files.

Movie files created with OBS require a lot of space, as approx. 30 images are recorded per second. The space required per minute is in the region of several dozen MBytes. The subsequent text and speech recognition require a lot of power, and there is currently no tool that does this job automatically.

The classic approach is an alternative. Screen copies can be created on any Linux desktop with ‘PrintScreen’ (top third right button). If the aim is to document some subsequent processes (e.g. in support), the undertaking soon becomes labor-intensive. The individual image files must be converted into one file. In most cases, a PDF file is created. To make the content searchable, text recognition must be started etc. etc.

Microsoft Recall or AI as a "role model"?

In itself, Microsoft Recall addresses precisely this concern. However, the automatic recording of content is nothing new. What is new is simply that the creation of screen copies is automated in real time so that the data can be searched.

The fact that the cloud should/must be used in some way for this remains a mystery. Obviously, it is more about manufacturers wanting to "copy" as much data as possible.

Unfortunately, this is not only the case with Microsoft and Windows. Apple has just announced that its products will be equipped with AI functions across the board. According to the "article" on the Apple presentation of 10.6.24, a **report (in German) appeared on Swiss television (SRF),** see also the **document created with Capt2PDF (in German) here.** Quote from it:

The functions have been deeply embedded in the operating systems for iPhone, Mac and iPad, emphasized software boss Craig Federighi. This means that Apple's AI models have access to the information users need in order to be useful to them. Many of the models run directly on the devices, emphasized Federighi. If necessary, the cloud is also connected – but with an encrypted connection. The Apple software decides on a case-by-case basis whether a task should be performed locally or via the cloud.

For all those who do not see **"red"** here, the following should be noted.

1. An operating system does not "only" have access to necessary information, it always has access to all data. How else would it be possible to save a file, for example? However, this does not mean that an AI has to work.

2. If an operating system cannot be used, it makes no sense. However, no AI is necessary to be able to work, either locally or in the cloud.

3. No software decides on a case-by-case basis whether to outsource personal data to the cloud. With any software, it is clear from the outset what happens where and how. How stupid are users taken for? How “limited” are the media when such madness is published in this way?

Apart from the fact that the above example is intended to show that a lot of nonsense is being done under the term AI, this article can also be used to show how easy Capt2PDF is to use.

The source code of Capt2PDF can be found below. The program can be started on a terminal with **perl capt2pdf 5** (recording with an interval of 5 seconds). The recording is stopped with **perl capt2pdf 0.** It makes sense to use a key combination for these two commands.

On the AV multimedia desktop, you can use the key combination **Ctrl+PrintScreen to start the recording** or **Shift+Ctrl+PrintScreen to end the recording.** In between, the screen is recorded at intervals of 5 seconds. After exiting the program, the finished searchable PDF file is displayed on the desktop after a few seconds. Simple, isn’t it?

Does Capt2PDF need an AI or a cloud?

Not in the slightest. Capt2PDF works 100% locally, any reasonably modern CPU is capable of doing the job. Capt2PDF **(Zip file with sources here)** currently comprises around 260 lines, which are published here. If you are not quite so technically minded or not interested in it, just read on to the next paragraph.

```
#!/usr/bin/perl
# capt2pdf (c) v0.2 - 2024-06-12 by Archivista GmbH, Urs
Pfister, GPLv2
```

```
# program to capture every x seconds a screenshot, check for
at least 1%
# difference in the screens (compare from ImageMagick) and
if so, set
# resolution (convert from ImageMagick) and send it to
tesseract for
# background ocr (incl. balance checker for 1/2 of cpus),
needed programs:
# scrot|spectacle|gnome-
screenshot,compare,convert,pdftk,tesseract,zenity

use strict;
use constant PFAD0 => "/home/archivista/data";
use constant PFAD1 => PFAD0."/screens";
use constant PFAD => `echo -n \$HOME`.'/screens'; # home
folder / logs etc
if (-e PFAD0) { # arrange paths if it is
AVMultimedia/ArchivistaBox
  mkdir PFAD1 if !-e PFAD1;
  doit("ln -s ".PFAD1." ".PFAD) if !-e PFAD
} else { # all other we take screens folder in home
directory
  mkdir PFAD if !-e PFAD;
}
use constant RUNS => PFAD."/capt2pdf.wrk"; # running (has
screen pixels)
use constant OCRLANGS => PFAD."/capt2pdf.ocr"; # languages
for tesseract
use constant X11CAPT => "/usr/bin/scrot"; # X11 screen
caputre program
use constant X11OPT => ""; # options (last one needs to be
file flag)
use constant WAYCAPT => "/usr/bin/spectacle"; # WAYLAND
screen capture program
use constant WAYOPT => "-f -b -n -o"; # options, last one -o
for file name
use constant WAYCAPT2 => "/usr/bin/gnome-screenshot"; #
WAYLAND gnome
use constant WAYOPT2 => "-f"; # options, last one -o for
file name
use constant CONVERT => "/usr/bin/convert"; # set dpi
(resolution)
use constant COMPARE => "/usr/bin/compare"; # comparing
```

```
images (ImageMagick)
use constant PDFTK => "/usr/bin/pdftk"; # program to
compbine pdf pages together
use constant TESSERACT => "/usr/bin/tesseract"; # ocr
recognition
use constant TESSOCR => "eng+deu"; # default languages for
tesseract
use constant RESDPI => "300x300"; # set image res (if no at
all, set '')
use constant XRANDR => "/usr/bin/xrandr"; # get resolution
of screen (1st one)
use constant PDFVIEW => "/usr/bin/qpdfview"; # open file
with pdf viewer
use constant MSG => "/usr/bin/zenity"; # send message with
zenity
use constant NOTE => "/usr/bin/notify-send"; # send message
with notify-send
use constant TIT => "Capt2PDF"; # title for application
use constant START => "start..."; # start message
use constant STOP => "stop..."; # stop message
use constant PERL => "/usr/bin/perl"; # get resolution of
screen (1st one)
use constant D0 => " 2>/dev/null"; # suppress error messages
my $mode = shift; # 0=stopit, 2-60=startit, file=capture
(called from prg)
my $lang = shift; # language string for tesseract
my $job = $0;
die "$0 mode [lang] => 0=stop,2-60=capt.time,deu+eng=ocr
lang" if $mode eq "";
logit("$0 called with $mode");
if ($mode eq "0" || $mode eq "") {
  stopit(); # stop capturing and create pdf file
} elsif ($mode>=2 && $mode<=60) {
  startit($mode,$lang); # start capturing (must be between 2
and 60 seconds)
} elsif (-e "$mode") {
  capture($mode); # a new page hase arrived, so process it
}

sub doit { # process a system call and log results
  my ($cmd) = @_;
  my $res=system($cmd);
  logit("$res=>$cmd");
  return $res;
}

sub logit { # log file to get infos about what was done
  my ($msg) = @_;
  open(FOUT,">>".PFAD."/av.log");
  print FOUT "$msg\n";
  close(FOUT);
}

sub getpixels { # get pixel of your screen (if no xrandr =>
then fullhd)
  my $screenx = 1980; my $screeny = 1080;
  my $xrandr = XRANDR;
  if (-e $xrandr) {
    my $xrandr = `$xrandr | grep '*'`;
    $xrandr =~ /([0-9]+)(x)([0-9]+)/;
    if ($1>0 && $2 eq "x" && $3>0) {
      $screenx = $1;
      $screeny = $3;
    }
  }
  my $maxpixels = $screenx*$screeny;
  logit("resolution of screen:$screenx:$screeny");
  return $maxpixels;
}

sub writefile { # write simple text file
  my ($file,$cont) = @_;
  open(FOUT,">$file");
  print FOUT "$cont";
  close(FOUT);
}

sub stopit { # stop capturing and create pdf file
  my $pfad = PFAD; my $ocr = -1; my $capt = 0; my $first=1;
  doit("rm ".RUNS) if -e RUNS;
  noteit(STOP,5);
  for(my $c=2;$c<10;$c++) { # wait for termination of ocr
    $ocr = countJobs(); # default is tesseract
    my $conv = countJobs(CONVERT);
    my $comp = countJobs(COMPARE);
    if (iswayland() eq "") {
      $capt = countJobs(X11CAPT);
    } else {
      $capt = countJobs(WAYCAPT) if -e WAYCAPT;
      $capt = countJobs(WAYCAPT2) if -e WAYCAPT2;
    }
    logit("ocr jobs:$ocr--conv:$conv--comp:$comp--
capt:$capt");
    if ($ocr==0 && $conv==0 && $comp==0 && $capt==0) {
      last if $first==0; $first=0;
    }
    sleep $c;
  }
  if ($ocr>0) {
    logit("we stop waiting, $ocr ocrjobs left");
  } else {
    logit("ocr processing has ended");
  }
  my @files = <$pfad/av-*.pdf>;
  my $file = $files[0];
  if (-e $file) {
    $file =~ s/(av-)/screen-/g; # unique name so it is not
killed next time
    my $prg = PDFTK." ".join(" ",@files)." output
$pfad/all.pdf";
    doit($prg);
    if (-e "$pfad/all.pdf") {
      doit("rm -f $pfad/av-*");
      doit("mv $pfad/all.pdf $file");
    }
  }
  doit(PDFVIEW." $file".DO." &") if -e PDFVIEW && PDFVIEW ne
"" && -e $file;
  doit("rm ".OCRLANGS) if -e OCRLANGS;
}

sub startit { # start capturing of sceenshots
  my ($wait,$lang) = @_;
  my $pfad = PFAD;
  $lang = ocrlangs($lang);
  if (!-e RUNS && !-e OCRLANGS) {
    noteit(START,2);
    my $pixels = getpixels();
    writefile(RUNS,$pixels);
    writefile(OCRLANGS,$lang);
    doit("rm -f /tmp/av*.png");
    doit("rm -f /tmp/av*.gif");
    doit("rm -f $pfad/av-*");
  } else {
    logit("$0 already started");
  }
  my $file = "/tmp/av-current.png";
  $wait--; # 1 second pause between ending capturing and
post processing
  my $cmd = X11CAPT." ".X11OPT;
  if (iswayland() ne "") {
    $cmd = WAYCAPT." ".WAYOPT if -e WAYCAPT;
    $cmd = WAYCAPT2." ".WAYOPT2 if -e WAYCAPT2;
  }
  my $prg = "sleep 1;".PERL." $0 $file".D0;
  doit("while [ -e ".RUNS." ];do sleep $wait;$cmd $file
".D0.";$prg;done &");
}

sub ocrlangs { # check and combine ocr languages for
tesseract
  my ($lang) = @_;
  my $langs = "";
  my @langs = split(/\,/,$lang);
  foreach my $lng (@langs) {
    if ($lng eq "deu" || $lng eq "frk" || $lng eq "fra" ||
$lng eq "ita" ||
      $lng eq "spa" || $lng eq "nld" || $lng eq "eng" ||
$lng eq "deu-frak") {
      $langs .= "+" if $langs ne "";
      $langs .= $lng;
    }
  }
  $langs = TESSOCR if $langs eq "";
  return $langs;
}

sub capture { # process captured page (incl. check if we
have differences)
  my ($file2) = @_;
  my $file = "/tmp/av-".timestamp().".png";
  doit("rm -f $file") if -e $file;
  doit("mv $file2 $file") if -e $file2 && !-e $file;
  my $pfad = PFAD;
  my @lines = <"/tmp/av-*.png">;
  my $current = pop @lines;
  my $last = pop @lines;
  if ($last ne "" && $file eq $current) {
    logit("curr:$current");
    logit("last:$last");
    my $check = $last.".gif";
    my $cmp = COMPARE;
    my $pixeldiff= int `$cmp -metric AE -fuzz 5% $last $file
$check 2>&1`;
    my $pixeldiff1 = $pixeldiff*100; # scale 1% to 100% for
comapring
    my $chk = "cat ".RUNS.D0;
    my $maxpixels = `$chk`;
    if ($pixeldiff1>$maxpixels) {
      logit("SAVENEW with $pixeldiff");
      captureOCR($last);
    } else {
      logit("NEARTO with $pixeldiff");
    }
    doit("rm $last") if -e "$last";
    doit("rm $check") if -e "$check";
  }
}

sub captureOCR { # create a pdf page from current screen
shot
  my ($file) = @_;
  my $pfad = PFAD; my $langs = ""; my $chk = OCRLANGS;
  $langs = `cat $chk` if -e "$chk";
  $langs = TESSOCR if $langs eq "";
  my $cpus = `cat /proc/cpuinfo | grep processor | wc -l`;
  my $ocrjobs = countJobs();
  if ($ocrjobs*2<=$cpus) {
    my @parts = split(/\//,$file);
    my $fname = pop @parts;
    @parts = split(/\./,$fname);
    my $ext = pop @parts;
    my $fbase = join('.',@parts);
    if (!-e "$pfad/$fbase.pdf") {
      if (RESDPI ne "") {
        my $cmd = CONVERT." $file -density ".RESDPI." -units
pixelsperinch ".
          "$pfad/$fbase.jpg";
        doit($cmd);
        $file = "$pfad/$fbase.jpg" if -e "$pfad/$fbase.jpg";
      }
      my $cmd = TESSERACT." -l $langs $file $pfad/$fbase pdf
".D0." &";
      doit($cmd);
    }
  }
}

sub countJobs { # give back the number of tesseract sessions
  my ($job) = @_;
  $job = TESSERACT if $job eq "";
  my $jobs = `ps ax | grep $job | grep -v grep | wc -l`;
  chomp $jobs;
  $jobs=0 if $jobs eq "";
  return $jobs;
}

sub timestamp { # give back current time stamp
  my @t = localtime( time() );
  my $Y = $t[5] + 1900;
  my $M = sprintf( "%02d", $t[4]+1 );
  my $D = sprintf( "%02d", $t[3] );
  my $h = sprintf( "%02d", $t[2] );
  my $m = sprintf( "%02d", $t[1] );
  my $s = sprintf( "%02d", $t[0] );
  return $Y."-".$M."-".$D."_".$h."-".$m."-".$s;
}

sub iswayland { # check if it is wayland display manager
  return `echo -n \$XDG_SESSION_TYPE`;
}

sub noteit {
  my ($msg,$wait) = @_;
  if (-e NOTE) {
    $wait=$wait*1000;
    doit(NOTE." ".TIT." $msg -t $wait".D0);
  } elsif (-e MSG) {
    doit(MSG." --timeout=$wait --info --title=".TIT." --
text=$msg".D0);
  }
}
```

The program was developed for AVMultimedia and ArchivistaBox. However, it should also run on most other Linux desktops. Under X11 the program **scrot** is used for screen captures, in Wayland it is **spectacle** or **gnome-screenshot.**

It should be mentioned at this point that the programs **compare, convert, tesseract and pdftk must be available** in addition to the capture tool. Capt2PDF starts a one-liner in the bash, which creates screen copies until Capt2PDF is terminated. After a screenshot has been created, Capt2PDF is started, specifying the current image, and from there the text recognition (OCR) is tesseract in the background.

With the ImageMagick utility program **compare**, the aim is to find out whether the current screenshot differs from the last one. Only then does it make sense to create a searchable PDF file from it. Currently, 1% of the pixels on the screen must change, otherwise the copy is discarded. This means that no new screens or PDF pages are created for pure mouse movements or updated status displays.

If there are more than 1% changed pixels, **tesseract** is called. Before text recognition is started, the desired resolution (dpi) is stored in the image file using **convert.** 300 dpi produces good results for text recognition (OCR) on all sides. On a 4K screen with 3860×2160 pixels, plus/minus approximately an A4 page in landscape format with 300dpi is available (3508×2480 pixels for A4).

The pages are created directly when the pages are recorded. Capt2PDF checks the number of CPU cores to ensure that slow CPUs (processors) are not “overloaded”. If more than half of the CPUs are in use for **tesseract,** no more text recognition takes place until sufficient resources are available again. Capt2PDF becomes “somewhat” forgetful here. It should be noted, however, that this only occurs with very old processors; as a rule, **tesseract** needs approx. 2 seconds for one page.

The corresponding PDF files are stored in the home directory (e.g. **/home/arcihvista)** under **screens** as individual pages. When Capt2PDF is closed, **pdftk** is used to create the complete PDF file. This process takes barely more than five to ten seconds, even for hundreds of pages. Finally, the searchable PDF file is displayed using the **qpdfview** viewer.

A short commercial at this point: Anyone who requires further evaluation or integral searchability of multiple PDF files will find ArchivistaBox a good workhorse. If you don’t want to use a fully-fledged document management system (DMS), you can of course also merge several searchable PDF files created in this way into one file. The corresponding call is simple:

```
pdftk file1.pdf file2.pdf output allfiles.pdf
```

Capt2PDF is of course open source (GPLv2). It was developed for the Linux distribution AVMultimedia (and therefore also for the ArchivistaBox) and is available there from version 2024/VI via the key combinations Ctrl+PrintScreen or Shift+Ctrl+PrintScreen. With a current size of approx. 260 lines of code, it is a handy tool for conveniently and easily capturing content when researching on the Internet, for example. It should be noted that content recorded in this way may not be placed on the Internet at will (more on this below).

Capt2PDF or AI: Pandorra's box

The example of the **(almost) unedited Apple press release used here, which ends up on the main page of Swiss television (Capt2PDF copy here),** fits well with the current AI issue, because it can be used as an example to show the contradictions that are currently arising. It should be clear that a private individual is allowed to save such an article. But is an operating system allowed to store the information created in this way or the information obtained from it in the cloud?

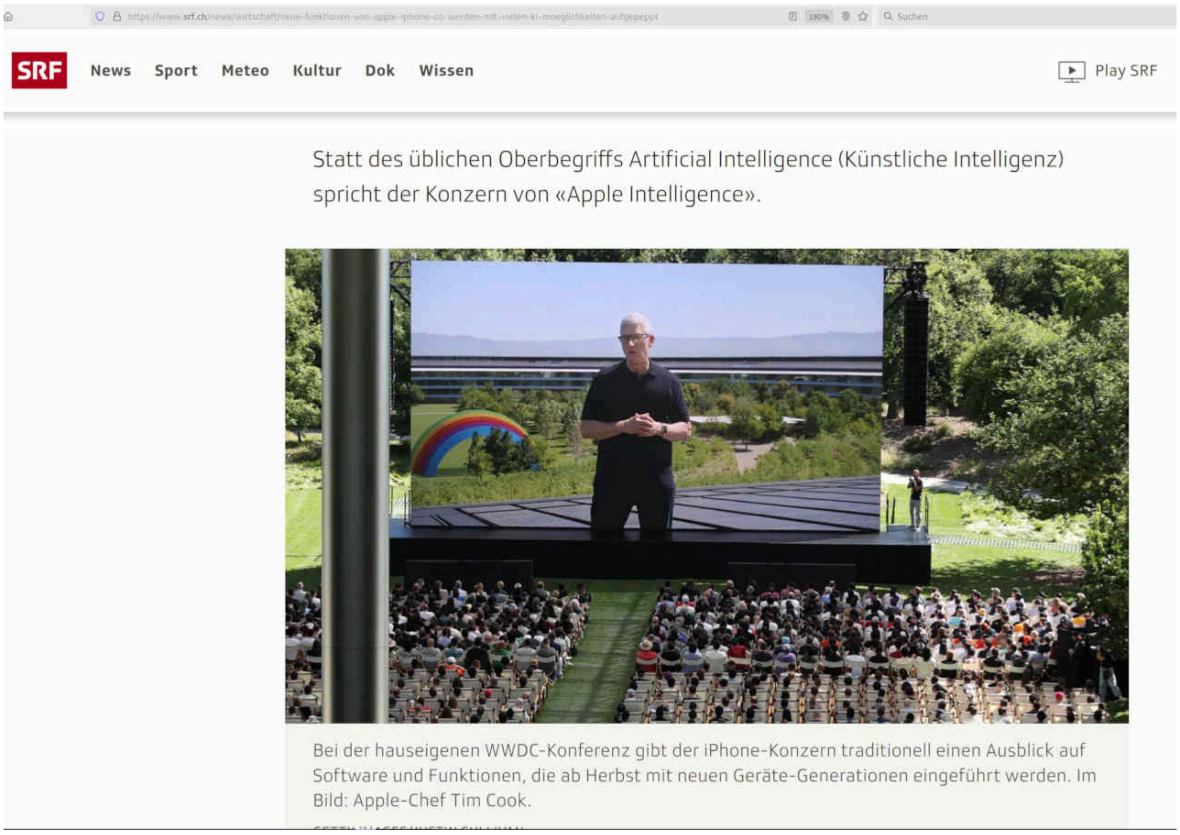
Where would AI be if copyright-protected content wasn't used billions of times to train AI? How can authors defend themselves against their content being used by third parties? With the Capt2PDF tool presented here, all work is done on a local computer. What happens when these files leave the local computer?

For example, does the **copy created here with Capt2PDF** contain copyrighted content? What happens if an operating system creates such copies without being asked and they become publicly accessible? On the subject of copyright, please refer to **anwalt.de (in German)** and the judgment cited there (quote):

A work is generally only protected by copyright if it has a certain level of creativity – i.e. it must not be something commonplace – and contains a personal intellectual achievement. In this respect, press releases are regarded as so-called "small coin", i.e. as a less sophisticated text which, however, just reaches the lowest limit of copyright protection (see also LG Hamburg, judgment of January 31, 2007, Ref.: 308 O 793/06).

In the **case of the (excerpt) of the SRF article published here,** the situation appears clear in this context. Even if it exceeds the copyright threshold, it is a criticism of the media's handling of AI and a solution with Capt2PDF in order to show what is going wrong and how it can be done better.

The Pandorra's box is that, for example, recording or creating searchable content is currently not trivial for users. The tech connoisseurs will claim that this is exactly why AI exists. The public perception (especially in the media) is that AI is essential.That's why you should take a closer look at such "articles". If articles are not labeled with full names, with phrases like 'Siri is getting even smarter', it could almost give rise to the suspicion that AI was at work here. It is fitting that the images come from GettyImages. In the first image, the text 'Siri' is mixed with 'AI' (definitely AI-level quality) and the second image reads: "Image: Apple CEO Tim Cook".


https://www.srf.ch/news/wirtschaft/neue-funktionen-von-apple-iphone-co-werden-mit-vielen-ki-moeglichkeiten-aufgepeppt
190%
Suchen
SRF
News
Sport
Meteo
Kultur
Dok
Wissen
Play SRF

Statt des üblichen Oberbegriffs Artificial Intelligence (Künstliche Intelligenz) spricht der Konzern von «Apple Intelligence».

Bei der hauseigenen WWDC-Konferenz gibt der iPhone-Konzern traditionell einen Ausblick auf Software und Funktionen, die ab Herbst mit neuen Geräte-Generationen eingeführt werden. Im Bild: Apple-Chef Tim Cook.

It would probably be correct to write, the boss does appear in a movie on a sensationally large display. Even if the article had copyright qualities, an (almost) complete publication seems appropriate here because it can show how blurred information is handled in connection with AI. Without tools such as Capt2PDF, however, it is quite time-consuming to demonstrate this.

But what is the situation when manufacturers of operating systems more or less copy users' data by the billions without being asked? The question may be asked **(in line with the PDF file published here),** what is the truth? What if this content is not quite as it is presented (not uncommon in advertising)? Or what happens if the data is subsequently altered?

The Apple boss has an amusing rainbow with a certain statement. What if the AI in certain states has something against rainbows? Will the cloud AI present me with a "neutral" water jet in these states? Or will unpleasant information be filtered out in real time as it is displayed?

What chances do normal users still have today? In the face of tech companies that are intruding into our daily lives, in the face of the media, which might perhaps report a little more critically? In the face of state authorities, where it takes a very long time for data protection and competition law violations to be punished, if at all.

Open source, if it is used locally, is a good, probably the best, alternative here. The Linux distribution AVMultimedia is designed precisely for this purpose, Capt2PDF is available here from version 2024/VI in such a way that the user alone decides whether

Capt2PDF is used or not. It is a useful program for recording screen content in such a way that Pandorra’s box does not have to be opened. In other words, completely without AI and cloud, but simple, transparent (100% open source) and locally and in real time on your own Linux desktop.