COSI – The Common OCR Service Interface

Rev 4 (2 apr 2016) – © Sylvain Giroudon, 2007-2016

What is COSI

COSI stands for « Common OCR Service Interface ». COSI is an open standard targeted at using different OCR (Optical Character Recognition) programs as external agents in a unified manner. COSI allows to integrate an OCR tool into various top-level applications using a client-server approach.

COSI has been successfully implemented and experimented on well-known open source OCR applications:

– GOCR (http://jocr.sourceforge.net);

– Tesseract-OCR (http://code.google.com/p/tesseract-ocr).

– GNU Ocrad (http://www.gnu.org/software/ocrad).

COSI implementations are available from the GitHub repository, at github.com/testfarm/cosi.

Where does COSI come from ?

The primary goal of COSI is to implement OCR capability for the "*TestFarm Virtual User*" software platform, a test automation tool that detects and tracks graphical objects displayed by an application. Please refer to the web site http://www.testfarm.org for further details about this tool.

COSI Overview

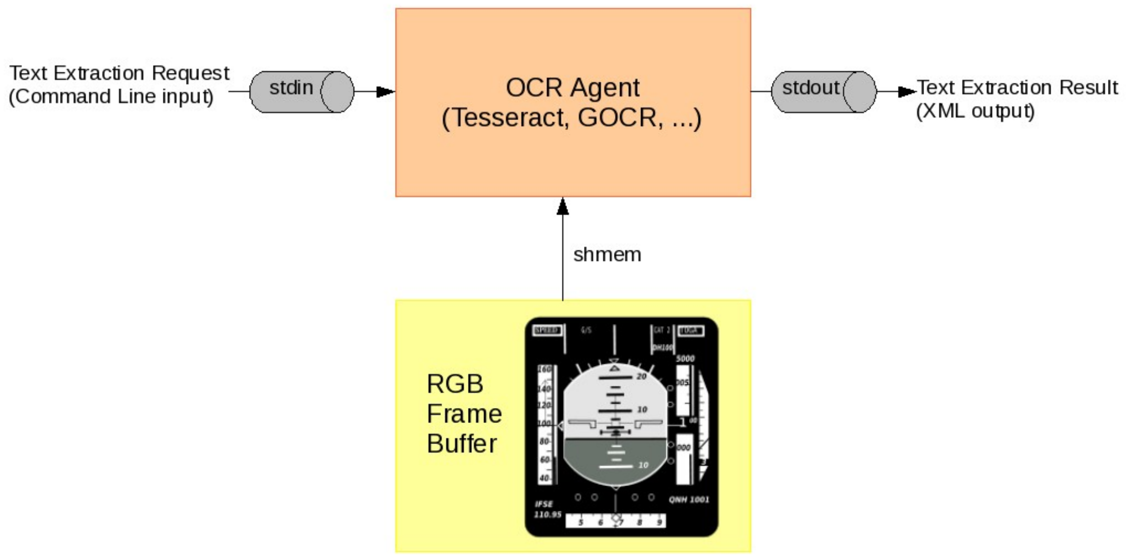
The main principles that characterize COSI are:

– We use use **existing OCR tools** as local services to extract text from a graphical frame buffer, rather than from a static image file. An OCR tool that supports COSI is then called the "OCR server".

– The OCR server takes its graphical input from a frame buffer mapped in a shared memory segment. This **shared frame buffer** is written by the client and read by the OCR server.

– The OCR server accepts requests from the client through its standard input. A request is a simple **command line**, which may include some options such as the coordinates of the graphical window from which the text is extracted (a.k.a Region of Interest).

– When the requests is processed, the OCR server writes the result in **XML format** to its standard output. The OCR result contains the extracted text and its graphical localisation in the frame buffer, both on a per-character and per-line basis.



COSI Interfaces Definition

Frame Buffer

The input image processed by the OCR agent is mapped as a shared memory area. This allows the client program to export the input image without writing any intermediate image file (which is often a resource consuming process in term of CPU and i/o operations). The shared memory (shmem) frame buffer contains some image format information, followed by the RGB pixels.

Offset	***Format***	***Field name***	***Content***
0	unsigned long (32 bits)	`width`	Frame width in pixels (number of pixels per row)
4	unsigned long (32 bits)	`height`	Frame height in pixels (number of rows)
8	unsigned long (32 bits)	`bpp`	Bytes per pixel. Incrementing a frame buffer pointer (X,Y) by this value makes it point to (X+1,Y)
12	unsigned long (32 bits)	`rowstride`	Bytes per row. Incrementing a frame buffer pointer (X,Y) by this value makes it point to (X,Y+1).
16	array of bytes	`buf[]`	Array of pixels. The size depends of the frame geometry fields above (typically `height * rowstride`).

A C declaration of the shmem frame buffer would then give:

```
struct COSI_frame_buffer {
  unsigned long width;       /* Number of pixels per row */
  unsigned long height;      /* Number of rows */
  unsigned long bpp;         /* Bytes per pixel */
  unsigned long rowstride;   /* Bytes per row */
  unsigned char buf[1];      /* Actual size  is .height * .rowstride */
};
```

In a POSIX operating system (Unix, Linux, BSD and others), a shared memory buffer is designated by a numerical key called a “shmid”. The shmid should be given by the client to the OCR agent as an input parameter. The client and the OCR agent should alloc and map the shared frame buffer using the POSIX shared memory system calls (`shmget`, `shmat`, etc.).

Requests input

Once the frame buffer is mapped, the OCR agent is ready to accept processing requests from its standard input. An input request is a textual command line, composed of attributes, and terminated by a newline character (Line Feed). Each attribute follows the XML-like *attr-name*="*attr-value*" format. Quotes may be ommited if *attr-value* does not contain blank characters (space, tab, etc.).

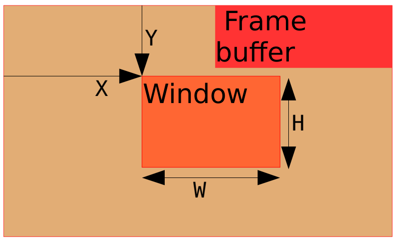
The request format synopsis is:

```
[id=request-id]  [geometry=window-geometry]  [options...]
```

► The request can be identified using the "`id=`*request-id*" attribute. This identifier is an arbitrary text word used as a traceability marker between the request and the output result.

► The request may contain a "`geometry=`*window-geometry*" attribute, which is a *Region Of Interest* asking the OCR agent to perform the text extraction from within this window rather than from the full frame buffer area.

The *window-geometry* specification complies to the X-Window (X11) geometry format. This format allows to define the window position and size in a single character string *W*x*H*+*X*+*Y*. *W*x*H* represents the size of the window, and +*X*+*Y* represents the position.



► The other "*options*..." attributes are optional and depend on the OCR agent capabilities and features. Unsupported attributes should be silently ignored by the OCR agent.

Results output

After OCR process is completed, the result are output in XML format. This format shows the extracted text on both a per-character and per-line basis, and includes the localisation information of the character/line objects.

The synopsis of the XML output format follows:

```
<document id="request-id" geometry="window-geometry" options...>
  <page>
    <line geometry="line-geometry">
      <box geometry="box-geometry" value="char" />
      ...
      <space geometry="line-geometry" />
      ...
    </line>
    ...
 </page>
</document>
```

The whole output data are encapsulated into the top-level element `<document>`. The attributes of this element are directly taken from the input request: request identifier, window (ROI) geometry, and other agent-specific options.

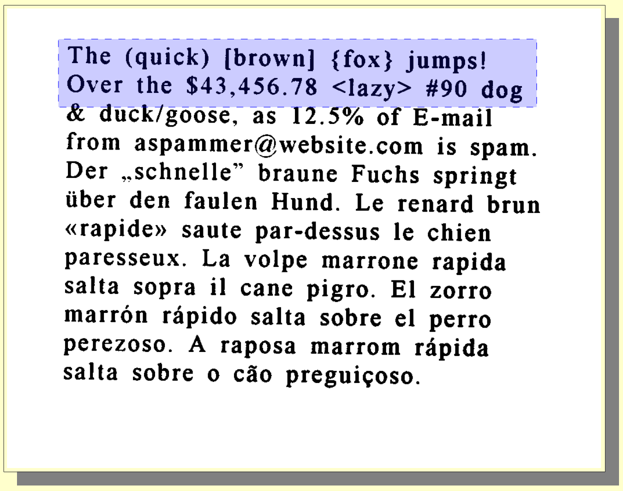
Document pages are delimited by elements `<page>`. This element is for future (and well, it also keeps some coherence in the page layout hierarchy) : only one page per document is supported by COSI for now.

Each line extracted from the document is stored in a `<line>` element. The attribute `geometry=`"*H*x*W*+*X*+*Y*" indicates where the line is located.

Inside each `<line>` element are listed the extracted characters. An “real” character is embedded into a `<box>` element, with its geometry and ASCII symbol respectively stored as attributes `geometry` and `value`. A space is not considered as a normal character, but is shown as a `<space>` element, with its geometry as an attribute.

The example below shows an output from the *Tesseract* OCR agent. It extract a piece of text from the sample image `eurotext.tif` (included in the *Tesseract* source tree for testing purpose):

```
<document id="TEXT" geometry="805x117+93+53">
<page>
 <line geometry="721x52+11+12"       >
  <box geometry="28x32+11+12" value="T" />
  <box geometry="22x34+42+12" value="h" />
  <box geometry="19x25+67+21" value="e" />
  <space geometry="25x44+86+12" />
  <box geometry="14x40+111+15" value="(" />
  <box geometry="22x34+128+22" value="q" />
  <box geometry="22x24+153+24" value="u" />
  <box geometry="11x33+179+14" value="i" />
  <box geometry="20x26+194+22" value="c" />
  <box geometry="22x33+216+14" value="k" />
  <box geometry="15x38+241+16" value=")" />
  <space geometry="26x44+256+14" />
  <box geometry="13x41+282+17" value="[" />
  <box geometry="23x34+297+17" value="b" />
  <box geometry="18x24+322+25" value="r" />
  <box geometry="23x26+340+25" value="o" />
  <box geometry="31x25+365+26" value="w" />
  <box geometry="23x25+399+25" value="n" />
  <box geometry="13x41+424+17" value="]" />
  <space geometry="28x43+437+17" />
  <box geometry="16x40+465+20" value="{" />
  <box geometry="21x34+486+18" value="f" />
  <box geometry="23x26+503+27" value="o" />
  <box geometry="23x23+528+29" value="x" />
  <box geometry="16x40+556+19" value="}" />
  <space geometry="21x46+572+18" />
  <box geometry="16x42+593+22" value="j" />
  <box geometry="24x26+612+31" value="u" />
  <box geometry="35x25+638+30" value="m" />
  <box geometry="24x33+674+30" value="p" />
  <box geometry="18x26+700+31" value="s" />
  <box geometry="9x35+723+21" value="!" />
 </line>
 <line geometry="786x54+10+61"       >
  <box geometry="32x34+10+61" value="O" />
  <box geometry="22x24+44+71" value="v" />
  <box geometry="20x25+68+71" value="e" />
  <box geometry="16x24+91+71" value="r" />
  <space geometry="24x37+107+61" />
  <box geometry="13x30+131+68" value="t" />
  <box geometry="23x33+147+64" value="h" />
  <box geometry="20x25+172+72" value="e" />
  <space geometry="25x42+192+63" />
  <box geometry="20x38+217+63" value="$" />
  <box geometry="21x33+241+66" value="4" />
  <box geometry="18x33+267+66" value="3" />
  <box geometry="9x13+292+92" value="," />
  <box geometry="21x32+306+67" value="4" />
  <box geometry="20x33+331+67" value="5" />
  <box geometry="22x32+355+67" value="6" />
  <box geometry="8x8+382+92" value="." />
  <box geometry="21x32+396+67" value="7" />
  <box geometry="21x33+421+67" value="8" />
  <space geometry="26x49+442+63" />
  <box geometry="24x24+468+74" value="<" />
  <box geometry="11x34+496+68" value="l" />
  <box geometry="20x26+511+77" value="a" />
  <box geometry="20x24+534+79" value="z" />
  <box geometry="22x33+555+79" value="y" />
  <box geometry="25x24+580+73" value=">" />
  <space geometry="24x38+605+68" />
  <box geometry="22x35+629+71" value="#" />
  <box geometry="20x33+655+72" value="9" />
  <box geometry="22x32+678+73" value="0" />
  <space geometry="25x44+700+71" />
  <box geometry="22x33+725+74" value="d" />
  <box geometry="21x25+750+82" value="o" />
  <box geometry="23x34+773+81" value="g" />
 </line>
</page>
</document>
```


The (quick) [brown] {fox} jumps!
Over the $43,456.78 <lazy> #90 dog
& duck/goose, as 12.5% of E-mail
from aspammer@website.com is spam.
Der „schnelle" braune Fuchs springt
über den faulen Hund. Le renard brun
«rapide» saute par-dessus le chien
paresseux. La volpe marrone rapida
salta sopra il cane pigro. El zorro
marrón rápido salta sobre el perro
perezoso. A raposa marrom rápida
salta sobre o cão preguiçoso.

Bringing COSI support to existing OCR agents

GOCR

COSI support is available for GOCR 0.50. It is brought by 2 source patch files, to be applied to the original GOCR development tree :

– `xml_output.patch` extends the original XML output capabilities of GOCR to the COSI format. It adds a command option "`-F` ***num***" (like in XML Filter) to enable COSI-specific attributes and elements. By default, the original GOCR format (i.e. non-COSI compliant is enabled).

– `server.patch` adds COSI server capability to GOCR, by adding support for getting an image from a shared memory frame buffer, and accepting requests from the standard input. This mode is enabled with a new command option "`-S` ***shmid***".

To invoke GOCR as a COSI server from a shared-memory frame buffer id ***shmid***, the command is:

```
$ gocr -f XML -F 10 -S shmid
```

Tesseract-OCR

COSI support is available for Tesseract 3.03 and later. It is implemented as a program that uses the Tesseract library. This program is named "`tesseract-srv`".

To invoke Tesseract as a COSI server from a shared-memory frame buffer id ***shmid***, the command is:

```
$ tesseract-srv shmid
```

GNU Ocrad

COSI support is available for Ocrad 0.25. It is brought by 2 source patch files, to be applied to the original Ocrad development tree :

– `xml_output.patch` brings XML output capabilities to Ocrad, complying to the COSI format. XML output can be enabled with the command option "`--xml`". It is written to the result export file (option "`-x` ***exportfile***").

– `server.patch` adds COSI server capability to Ocrad, by adding support for getting an image from a shared memory frame buffer, and accepting requests from the standard input. This mode is enabled with a new command option "`-S` ***shmid***".

To invoke GNU Ocrad as a COSI server from a shared-memory frame buffer id ***shmid***, the command is:

```
$ ocrad -x - -Fxml -S shmid
```