

Optical Character Recognition (OCR) and Scanning

Handling

OCR data is stored in Teedy database table `t_file` which contains the string column `fil_content_c`. In H2 the data is stored als plaintext string. In PostgreSQL the column is filled as datatype `::text`. A normal select returns number. The unencrypted OCR text data can be accessed from the large object by using some SQL statement like

```
select
fil_name_c,
convert_from(loread(lo_open(fil_content_c::int, 131072), 999999999), 'UTF8')
from t_file WHERE fil_deletedate_d IS NULL AND fil_content_c IS NOT NULL limit 1;
```

Teedy uses a built in process runner to start the binary `tesseract` with a language parameter. This works if "`tesseract`" is contained in `$PATH` (Linux) or `%PATH%` (Windows) environment variable.

Fixing faulty `fil_content_c` data the easy way

Some quick fix for issue described in <https://github.com/sismics/docs/issues/451>

```
SELECT fil_content_c FROM t_file
WHERE  LENGTH(fil_content_c) > 6
ORDER BY fil_createdate_d DESC;

UPDATE t_file SET fil_content_c = NULL
WHERE  LENGTH(fil_content_c) > 6;
```

Converting LOB data to plain text (was required at some point from updating Teedy 1.8 to Teedy 1.9)

```
/*
Show items which start with useless linefeeds. We need to correct those because otherwise we
cannot continue with following statements (casting "fil_content_c::int" will fail and other
issues)
```

Result may be empty

```
*/
SELECT
    fil_id_c,
    fil_name_c,
    fil_content_c
FROM t_file
WHERE
    fil_content_c LIKE E'%\n'
;

/*Trim beginning linefeeds (only) away*/
UPDATE t_file SET fil_content_c = TRIM(e'\n' FROM fil_content_c)
WHERE
    fil_content_c LIKE E'%\n'
;

/*
Show faulty data which would return "invalid byte sequence for encoding "UTF8": 0x00" or
similar.
First we build some function to check for valid UTF8 bytea because sometimes we have faulty
stuff inside DB
Result may be empty
*/
CREATE FUNCTION is_valid_utf8(bytea) RETURNS boolean
    LANGUAGE plpgsql AS
$$BEGIN
    PERFORM convert_from($1, 'UTF8');
    RETURN TRUE;
EXCEPTION
    WHEN character_not_in_repertoire THEN
        RAISE WARNING '%', SQLERRM;
        RETURN FALSE;
END;$$;
SELECT
    fil_id_c,
    fil_name_c,
    loread(lo_open(fil_content_c::int, CAST( x'20000' AS integer)), 999999999) AS BYTE_DATA,
    LENGTH(loread(lo_open(fil_content_c::int, CAST(x'20000' AS integer)), 999999999)) AS LEN
```

```

FROM t_file
WHERE
    fil_content_c IS NOT NULL AND
    fil_content_c != '' AND
    LENGTH(fil_content_c) <= 6 AND
    is_valid_utf8(fil_content_c::bytea) IS FALSE
;

/*We set NULL to all items with faulty UTF-8 encoding (if there were some from previous
statement)*/
UPDATE t_file SET fil_content_c = NULL
WHERE
    fil_content_c IS NOT NULL AND
    fil_content_c != '' AND
    LENGTH(fil_content_c) <= 6 AND
    is_valid_utf8(fil_content_c::bytea) IS FALSE
;

/*
Select OCR content which is in LOB format (Large Object) and valid UTF-8
*/
SELECT
    fil_id_c,
    fil_name_c,
    fil_content_c,
    fil_content_c::bytea, /*shows "invisible" data which does not trigger NULL or ''*/
    loread(lo_open(fil_content_c::int, CAST( x'20000' AS integer)), 999999999) AS BYTE_DATA,
    /*we use the encoding we used to create the database. See setup instructions. Usually
this is "UNICODE" or "UTF8"*/
    LENGTH(loread(lo_open(fil_content_c::int, CAST(x'20000' AS integer)), 999999999)) AS LEN,
    convert_from(loread(lo_open(fil_content_c::int, CAST(x'20000' AS integer)), 999999999),
'UNICODE') as "fil_content_c"
FROM t_file
WHERE
    fil_content_c IS NOT NULL AND
    fil_content_c != '' AND
    LENGTH(fil_content_c) <= 6 AND
    is_valid_utf8(fil_content_c::bytea) IS TRUE
ORDER BY LEN ASC

```

```

;

/*Convert LOB data into plain text. First we do it for a custom selected file with fil_id_c*/
UPDATE t_file SET fil_content_c = convert_from(loread(lo_open(fil_content_c::int, CAST(
x'20000' AS integer))), 999999999), 'UNICODE')::TEXT
WHERE
    fil_id_c = '13411bb0-12fd-4e25-b483-2e2d18b344ed'
;

/*Check the conversion value*/
SELECT
    fil_id_c,
    fil_name_c,
    fil_content_c
FROM t_file
WHERE
    fil_id_c = '13411bb0-12fd-4e25-b483-2e2d18b344ed'
;

/*
Now we do mass processing for LOB to plain text
DO NOT CONTINUE WITH OTHER STATEMENTS IF THIS ONE FAILS AND CHECK THE UPPER ONES AGAIN
*/
UPDATE t_file SET fil_content_c = convert_from(loread(lo_open(fil_content_c::int, CAST(
x'20000' AS integer))), 999999999), 'UNICODE')::TEXT
WHERE
    fil_content_c IS NOT NULL AND
    fil_content_c != '' AND
    LENGTH(fil_content_c) <= 6 AND
    is_valid_utf8(fil_content_c::bytea) IS TRUE
;

/*We fix again useless linefeeds by trimming*/
UPDATE t_file SET fil_content_c = TRIM(e'\n' FROM fil_content_c)
WHERE
    fil_content_c LIKE E'%\n'
;

/*

```

Now that we converted all the LOB stuff we do mass processing for remaining stuff with length lesser than 6 chars because those OCR values are just crap

WARNING: DO NOT RUN THIS BEFORE CONVERTING BECAUSE YOU WILL OVERWRITE. IF YOU DID YOU WILL NEED TO REPROCESS ALL DOCUMENTS!

```
*/  
UPDATE t_file SET fil_content_c = NULL  
WHERE  
    fil_content_c IS NOT NULL AND  
    fil_content_c != '' AND  
    LENGTH(fil_content_c) <= 6  
;  
  
/*Finally we check again the values visually*/  
SELECT  
    fil_id_c,  
    fil_name_c,  
    fil_content_c  
FROM t_file  
  
/*Finally re-run the indexing from background UI web interface or API to have a good search  
index again*/
```

Tesseract OCR command line binary

The installation of tesseract is simple. Note that for different operating system versions there are different tesseract versions. All tesseract versions work different in their speed and quality. We figured out that tesseract 3 on Ubuntu 16 works much faster than tesseract 4 on Ubuntu 18.

<https://github.com/tesseract-ocr/tesseract/wiki>

Installation

For Linux users:

```
#install regular version  
sudo apt install tesseract-ocr tesseract-ocr-deu #will install the most recent version  
belonging to your OS. So older system you might get older tesseract  
  
#install devel version. See https://launchpad.net/~alex-p/+archive/ubuntu/tesseract-ocr-devel
```

```
sudo add-apt-repository ppa:alex-p/tesseract-ocr-devel sudo apt-get update  
sudo apt install tesseract-ocr tesseract-ocr-deu #add your desired languages here
```

For Windows users:

<https://github.com/tesseract-ocr/tesseract/wiki/4.0-with-LSTM#4x-for-windows>

Critical optimization

<https://github.com/tesseract-ocr/tesseract/issues/2611>

Some users said that disabling multiprocessing in tesseract fixes speed problems. Therefore some environment flag should be set using export. See also [Environment Configuration](#)

```
export OMP_THREAD_LIMIT=1
```

Scanner Apps for Smartphones

There are a LOT of scanner apps in PlayStore. Most of them have nearly same naming. The following list is only a minimalistic overview of stuff around the web. Mainly we are looking for open source applications.

- [Genius Scan](#)
- [CamScanner](#)
- [Notebloc](#)
- [OpenNoteScanner](#)
- [SwiftScan](#)

Wishes

- automatic upload to or sending by mail
- problem: what if you use multiple instances of DMS? Then you will need multiple upload locations. All known app do not deal with that feature. With app cloning the scanner app could be multiplied so each Scanner app instance has its own configuration. Then the scanner app could send to the correct inbox per DMS instance