# Technical Documentation

**Andrea Esposito**

**Apr 21, 2020**

# TABLE OF CONTENTS

# Part I

# The Emotions Tool

CHAPTER

ONE

# QUICK START

## 1.1 Installation

To use the tool, download its compiled binary from the repository and execute it from a console.

**Important:** The tool has been tested on **Ubuntu Xenial 16.04**.

The Affdex SDK is only available on Windows and Ubuntu Xenial 16.04, so compatibility with other Operative Systems is not guaranteed.

### 1.1.1 From source with CMake

Clone and open the GitHub repository in a console, using the following commands:

```
git clone https://github.com/espositoandrea/Bachelor-Thesis.git
cd Bachelor-Thesis
```

Open the directory containing the tool's source code:

```
cd emotions
```

Finally, create and compile the CMakeProject:

```
mkdir bin
cd bin
cmake -G "CodeBlocks - Unix Makefiles" ..
make
```

## 1.2 Usage

**Important:** To use the tool you must have the Affdex SDK installed on your machine. Then, you have to add `/path/to/affdex-sdk/lib` to the variable `$LD_LIBRARY_PATH` (on Ubuntu).

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/affdex-sdk/lib/
```

The tool will then search, in its folder, for the folder `lib/affdex-sdk/data/` (that has to contain the data used by Affdex).

The tool can be used through CLI (or executed by another script).

```
./emotions [<option>...] IMAGE...
./emotions [<option>...] --file FILE
```

Where `IMAGE` is a *data URI*. The available options are:

**-h, --help** Get the help message

**-f FILE, --file FILE** The file containing the images to be analyzed (as a data URI)

CHAPTER

TWO

# CODE DOCUMENTATION

## 2.1 The Entry Point

The main file.

This file is the main file of the tool.

**Author** Andrea Esposito <github.com/espositoandrea>

### Functions

int **`main`** (int *argc*, char ***argv*)

The main entry point.

This is the entry point of the tool.

**Return** An *exit code* based on the execution.

**Parameters**

- `argc`: The length of `argv`.
- `argv`: The array of arguments given through the command line.

## 2.2 The CLI

*exit_codes* **`setup_options`** (int *argc*, char ***argv*, std::vector<std::string> &*images*)

Set up the tool's options and arguments.

This function is responsible of the CLI API of the tool. It sets and handles the available options and arguments.

**Return** An *exit code*:

- *exit_codes::OK* If the given arguments are valid and no errors occurred.
- *exit_codes::HALT* If the given arguments are valid but the argument combination stops the execution.
- *exit_codes::ARGUMENT_ERROR* If the given arguments are invalid
- *exit_codes::UNKNOWN_ARGUMENT_ERROR* If an unknown error occurred.

**Parameters**

- `argc`: The length of `argv`.
- `argv`: The array of arguments passed via CLI.
- `images`: A variable that will contain the images passed through the CLI API.

## 2.3 The Exit Codes

**enum exit_codes**

A collection of all the exit codes of the tool.

This enum contains all the (expected) exit codes of the tool.

*Values:*

**OK** = 0

The tool exited with no error completing its tasks.

**HALT** = 1

The tool exited with no error, but without completing its tasks.

**ARGUMENT_ERROR** = 2

The tool exited due to errors in the given arguments.

**UNKNOWN_ARGUMENT_ERROR** = 3

The tool exited due to unknown errors while parsing the arguments.

## 2.4 The Data URI

**class data_uri**

A utility class to handle data URIs.

This class represents a data URI. A data URI is defined by MDN as a string with the followind sintax: `data:[<mediatype>][;base64],<data>`.

### Public Functions

**data_uri**(**const** std::string &*s*)

The class constructor.

This constructor creates a *data_uri* from a string.

**Parameters**

- `s`: The string representing the data uri.

**Exceptions**

- *data_uri::string_not_uri*: if `s` is not a valid data URI.

std::string **get_type**() **const**

Get the media type.

This function returns the media type of the data URI.

**Return** The media type (<mediatype> in data:<mediatype>;*base64*,<data>).

std::string **get_data**() **const**
Get the data.

This function returns the data contained in the data URI.

**Return** The data (<data> in data:<mediatype>;*base64*,<data>).

std::string **get_uri**() **const**
Get the URI.

This function returns the entire URI as a string.

**Return** The URI as a string.

### Public Static Functions

bool **is_data_uri**(**const** std::string &*s*)
Check if a string is a data URI.

The function checks if a string is in the format data:<mediatype>;*base64*,<data>.

**Return** True if s is a data URI, false otherwise.

**Parameters**

- s: The string to be checked

**class string_not_uri** : **public** exception
An exception raised if a string is not an URI.

This exception is thrown if a string, assumed to be one, is not a data URI.

## 2.5 The Base64 Utilities

**namespace base64**
Namespace for dealing with *base64* strings.

This namespace contains utilities to deal with *base64* strings.

### Functions

std::string **encode**(**const** std::string &*s*)
Encode a string to *base64*.

This function encodes a string to a *base64* string.

**Return** The encoded string.

**Parameters**

- s: The string to be encoded.

std::string **encode** (unsigned char **const** *s, unsigned int *len*)
Encode a string to *base64*.

This function encodes a string to a *base64* string.

**Return** The encoded string.

**Parameters**

- `s`: The string to be encoded.
- `len`: The length of the string `s`.

std::string **decode** (std::string **const** &*s*)
Decode a *base64* string.

This function decodes a *base64* string to a binary string.

**Return** The decoded string.

**Parameters**

- `s`: The string to be decoded.

# Part II

# The Browser Extension

CHAPTER

THREE

# THE BROWSER EXTENSION

# Part III

# The Server

CHAPTER

FOUR

# INTRODUCTION

## 4.1 Folder Structure

The `server/` folder contains all the source code of the developed server. Its structure is the following (all described folders are subfolders of `server/`).

**`views/`** This folder contains all the views developed for the server.

- **`views/layouts/`** This folder contains the layouts used to define the views.

**`survey/`** This folder contains all the required data for the survey.

**`assets/`** This folder contains all the static files that will be served without any modification.

- **`assets/images/`** A folder that contains all the images and illustrations used.
- **`assets/js/`** A folder that contains all the external JavaScript files (needed by the extension).
- **`assets/style/`** A folder that contains all the stylesheets of the server (written in SASS).

CHAPTER

FIVE

# THE DATA PROCESSOR

**class DataProcessor**()

The data processor.

This class processes the collected data, by adding various features (that can be extracted by the existing fields).

DataProcessor.**_analyzeEmotions**(*data*)

Extract the emotions from the image field.

**Arguments**

- **data** (*Object*) – The data to work on.

**Returns** **Promise.<Array.<Object>>** – A promise that will be resolved once the analysis is completed. The returned parameter contains the modified data.

DataProcessor.**_minimumAcceptedValue**

Get the minimum accepted value for the emotions' fields. If the registered value is less than the returned value, the emotions' object can be safely discarded.

DataProcessor.**_roundValue**(*val*)

Round a value to two decimal places.

**Arguments**

- **val** (*number*) – The value to be rounded.

**Returns** **number** – The value rounded to two decimal places.

DataProcessor.**process**(*data*)

Process the data. This modify the passed data injecting various features in them.

**Arguments**

- **data** (*Object|Array.<Object>*) – The data to be processed.

**Returns** **Promise.<Array.<Object>>** – A promise that will be resolved once all the data have been processed. The promise's parameter holds the modified data.

CHAPTER

SIX

# THE SURVEY

The survey is generated using the exported object defined in the module `survey-data.js`. Here is documented the structure of that object.

**class Survey**()
The survey configuration object.

**Arguments**

- **introduction** (*string*) – The introduction to the survey. Treated as raw HTML.
- **sections** (*Array.<Section>*) – The survey's sections.

**class Section**()
A section of the survey.

**Arguments**

- **title** (*string*) – The section's title.
- **questions** (*Array.<Question>*) – The section's questions.

**class Question**()
A question of the survey. @extends BasicQuestion.

**Arguments**

- **type** (*string*) – The type of question.
- **rules** (*Object*) – Various additional rules. Can be any HTML attribute accepted by the current input type.
- **placeholder** (*string*) – The input placeholder.
- **choices** (*Array.<string>*) – A list of choices. Used only if type is 'choice'.
- **question** (*string|Array.<BasicQuestion>*) – If it's a string, the same as BasicQuestion.question. If an array of BasicQuestion, a list of questions used if type is 'likert'.

**class BasicQuestion**()
A basic question of the survey. This class contains all the required field of a question.

**Arguments**

- **question** (*string*) – The question that will be asked to the user.
- **name** (*string*) – The name of the GET/POST parameter.

- **required** (*boolean*) – Wether or not the input is required.

# INDEX