# DSRF Flat File Library and Conformance Tool

User manual

Last updated: May 2019
License: Open source - Apache 2
Git repo: https://github.com/ddexnet/dsrf



## Introduction

The *DSRF Flat File Parser and Conformance Tool* is a open source Library that allows you to parse and test DDEX DSR Flat files in conformance with DDEX DSR Flat File Standard v3.0.

## Installation

The Library can be installed on the following platforms:

- Mac OS X
- Linux
- Windows

## Requirements

Prior to installation you will need the following packages

- Python (preferably v2.7 although other versions might work).
- Google Protocol Buffer compiler.

To verify that both of these are installed and accessible, you should be able to run both of these commands from the command line without getting a "command not found":

```
$ python --version
Python 2.7.10

$ protoc --version
libprotoc 3.6.1
```

If you need to install python or Google Protocol Buffer see Annex below for detailed instructions.

## Step-by-step guide

1. Download and decompress the DSRF install package (dsrf_1_0_x.tar)
2. From a Terminal, run the following command

   **For Unix (Mac OS X or Linux):**

   ```
   $ sudo python setup.py install
   ```

   **For Windows:**

   Open your command line as Administrator (right-click on cmd.exe):

   ```
   $ python setup.py install
   ```

Upon successful installation, you should read the following message:

```
Installed /Library/Python/2.7/site-packages/dsrf-1.0.0-py2.7.egg
Processing dependencies for dsrf==1.0.1 1
Finished processing dependencies for dsrf==1.0.1
```

## Testing your installation

Run the following commands from a Terminal:

```
$ TEST_FILE=testdata/DSR_TEST_YouTube_AdSupport-music_2015-Q4_IS_1of1_20160121T150926.tsv
$ python run_dsrf.py $TEST_FILE --human_readable=True
```

*Note: in all the following examples, we will replace the filename by this variable. Make sure you assign it a value or replace the variable name by an existing filename.*

You should see the report displayed as human readable protocol buffer starting with:

```
type: HEAD
version: "dsrf/30"
file_number: 1
rows {...
```

If that's the case. Congratulations ! your DSRF parser is now ready to run.

# Directory Structure

Your newly created directory should look like this

```
dsrf/
├── conformance/
├── parsers/
├── processor/
├── proto/
├── revenue_example/
├── schemas/
├── testdata/
```

**conformance:**
Contains the script that verify that your file is conformant to the specified Profile in the Standard. The output is in the specified log file.

**parsers:**
Contains the scripts that parse the files and extract the blocks to create the proto buffers queue.

**processor:**

Contains the scripts that process the reports

**proto:**
Contains the schema of the protocol buffers. Those files should NOT be edited.

**revenue_example:**
Contains example of scripts to be executed by the DSRF parser. You may use these example to create your own scripts.

**schemas:**
Contains the XSD files that represent (a) the allowed values for specific fields in the Standard and (b) the schema of all the profiles. Those files should NOT be edited.

**testdata:**
Contains the sample reports and the test files.

## Running DSRF

The DSRF parser is executed with the following syntax:

```
$ python run_dsrf.py [-h] [--dsrf_xsd_file DSRF_XSD_FILE]
[--avs_xsd_file AVS_XSD_FILE] [--dsrf_version DSRF_VERSION]
[--log_file LOG_FILE] [--human_readable HUMAN_READABLE] files_list
[files_list ...] | python [script] [--profile_name PROFILE]
```

Explanation of the variables:

| | |
|---|---|
| **--help** | print this help message and exit |
| **--dsrf_xsd_file** | The dsrf xsd schema file. This file contains the profiles and the row types definition. The default value is : ‘schemas/3.0/sales-reporting-flat.xsd’ |
| **--avs_xsd_file** | The xsd avs schema file. This file contains the allowed value set to the fixed string cells. The default value is : ‘schemas/3.0/avs.xsd’ |
| **--dsrf_version** | The format version The default value is : ‘3.0’ |

| | |
|---|---|
| **--log_file** | This file will contain the library logs. The default value is *'$tmp/example.log'* where $tmp is your system's default tmp directory. You can also specify a custom log file location. |
| **--human_readable** | If True, write the block to the stdout in human readable form. The default value is 'False'. |
| **files_list** | List of the files to be processed. Filenames are separated by a space character. For multi-file reports, all the files must be listed. 3 Compressed files are supported. |
| **script** | Name of the script you want to execute. This script takes the parsed blocks as input. For conformance validation, use `conformance/conformance_processor.py`. |
| **--profile_name** | The name of the profile you want use for validation (if you're using the `conformance_processor.py` script above). Allowed values are dictated by the XSD (eg. "UgcProfile" and "BasicAudio"). |

## The conformance tool

To check whether a file is conformant with the DSRF Standard, run the following command:

```
$ python run_dsrf.py --log_file=example.log $TEST_FILE | python
conformance/conformance_processor.py --profile_name=UgcProfile
```

The output should be as follows:

```
[Block conformance] Blocks validated: X blocks(Y rows).
The conformance validation passed successfully! Validated X blocks (Y
rows).
And your log file should look like this
INFO:dsrf.parsers.dsrf_report_manager:Validating the report file names.
INFO:dsrf.parsers.dsrf_report_manager:Start parsing file number 1.
INFO:dsrf.parsers.dsrf_report_manager:Start parsing the HEAD block in file number 1.
INFO:dsrf.parsers.dsrf_report_manager:Start parsing block number 1 in file number 1.
INFO:dsrf.parsers.dsrf_report_manager:Start parsing block number 2 in file number 1.
INFO:dsrf.parsers.dsrf_report_manager:Start parsing block number 3 in file number 1.
INFO:dsrf.parsers.dsrf_report_manager:Start parsing the FOOT block in file number 1.
```

# Executing a custom script

The library includes a sample of a custom script, which you can execute as follows:

```
>$ python run_dsrf.py $TEST_FILE | python
revenue_example/revenue_processors.py PUB_3
```

The name of a RightsController (PUB_3 in this example) must be passed as an argument. The script will return the sum of the Revenue attributed to PUB_3 in each block of the report. You can use this script as a base to extract any value or aggregate values from the report.

# Appendix: Installing the google Protocol buffer compiler

The instructions below assumes that you have Python installed. If that is not the case, please follow the instructions at https://www.python.org/downloads.

## Instructions for Windows OS

- Download the protocol buffer compiler from https://github.com/google/protobuf/releases. Choose the Windows version : **protoc-3.0.0-beta-2-win32.zip**
  Unzip and copy the directory somewhere under you /ProgramFiles directory (eg. c:\Program Files\DSRF_Library\protoc-3.0.0-beta-2-win32\protoc.exe)
  You need to declare the Path to the compiler :
  Open your command line as Administrator (right-click on cmd.exe)

```
> set PATH=%PATH%;c:\Program
Files\DSRF_Library\protoc-3.0.0-beta-2-win32
```

- You will also need the Python runtime library from https://github.com/google/protobuf
  Download unzip master.zip
  Open your command line as Administrator (right-click on cmd.exe)
  Go to the \python directory

```
> python setup.py
```

To test your installation, from your command line you should get the following:

```
> python --version
python 2.7.xx
> protoc --version
Libprotoc 3.0.0
```

## Instructions for MAC OS X

- Download the protocol buffer compiler from https://github.com/google/protobuf/releases. Choose the OS X version : **protoc-3.0.0-beta-2-osx-x86_32.zip**
  Unzip and copy the protoc executable somewhere in your system PATH, eg:

```
$ sudo cp path/to/protoc /usr/local/bin/
```

- You will also need the Protobuf Python runtime library from https://github.com/google/protobuf
  Download and unzip master.zip

  Fro the python/ directory, run:

```
$ sudo python setup.py
```

- To test your installation, from your command line you should get the following:

```
$ python --version
python 2.7.xx
$ protoc --version
Libprotoc 3.0.0
```

## Instructions for LINUX

On Linux, you can install the protobuf compiler using the Advanced Package Tool:

```
$ sudo apt-get update
$ sudo apt-get install protobuf-compiler
```

To test your installation, from your command line you should get the following:

```
$ python --version
python 2.7.xx
$ protoc --version
Libprotoc 3.0.0
```