Documentation of the Subtitling Conversion Framework (SCF) - 21/02/2017

Contents

About the SCF PDF documentation

Introduction

The subtitle conversion framework (SCF) is a set of different modules that allows a user to convert between different file based subtitle formats. The original driver and still the main purpose of the SCF is to convert a subtitle file that conforms to the binary subtitle format EBU STL (Tech 3264) in one XML document that conforms to one of the EBU-TT profiles (EBU-TT Part 1 - Tech 3350 or EBU-TT-D Tech 3380).

The main design idea is to keep the different conversion module as separate as possible so that in a processing chain one module implementation can be replaced by an alternative implementation.

In addition the SCF follows clearly defined and documented requirements. The documentation of the requirements are published together with the source code. Where possible the requirements that are implemented are covered by automated software tests.

Although there is no reason to not use the SCF in operation its goal is primarily to provide a reference framework that shows in a transparent way how to implement new subtitle standards (especially EBU-TT) and how to integrate them in an existing work flows. The source code can be used by other tool providers to implement their own conversion methods. They can just take it as a reference or they could just reuse some of the SCF code.

The current state of the SCF is a beta release. Comments from the community that we received since the first publication of scf have been considered. Any comments regarding the current version are welcome and should be made by opening an issue on github or sending an email to *open.source@irt.de*

Overview of the processing chain

The main conversion chain follows these processing steps:

- The start point is an binary STL which is decoded and serialized in an XML representation of the STL file. This has the following advantages: XML processing tools can be used from that point onwards the XML representation of the STL file is human readable standard XML validation technologies can be used.
- The STL XML representation is then transformed into an EBU-TT file that conforms to EBU-TT Part 1. The guidance for the transformation is the EBU document EBU Tech 3360. Like an EBU-STL file the purpose of an EBU-TT Part 1 document that was created from an STL file based on the Tech 3360 guideline is thought to be used in exchange and production but not for distribution.
- To get a EBU-TT that can be used for distribution over IP based networks a transformation to EBU-TT-D (EBU Tech 3380) is necessary. The module for this conversion step expects an EBU-TT file that follows the Tech 3360 guideline and creates one possible form for an EBU-TT-D file.

General prerequisites

- For the XSLT modules an XSLT processor that is conformant to XSLT 1.0 is needed. You could use for example a saxon xslt processor from version 6.5.5.
- To validate an STLXML with the STLXML W3C XML Schema a XML Schema 1.0 parser is required. You could use for example the Xerces XML parser and validator.
- For the conversion of an EBU STL file into STLXML (an XML based representation of the STL file) you need the python script stl2stlxml.py that . This python script requires a 2.7.x python runtime enviroment it will not run under python 3.0.

http://saxon.sourceforge.net/
http://xerces.apache.org/
https://github.com/Irt-Open-Source/stl2stlxml
https://www.python.org/downloads/

General Approach

Structure of requirements

The SCF implementation is based on the requirements for the different modules. The requirements are available as documentation of the different modules.

The structure of the requirements are as follows:

- **title:** a short title with the internal id of the requirement in brackets
- **description:** the requirement text - the specified text will be taken to test the implementation
- **area:** apart from more general requirements the requirements are categorized by modules (e.g. STLXML2EBUTT oder EBUTT2EBUTTD)
- **requirement review status:** this is the internal review status of the requirement itself (esp. of the requirement text)
- **status implementation:** this status indicates if the requirement is already met by the implementation the status codes are:
  - *outstanding* - the corresponding code has not been written yet or the requirement has been implemented but there are no test files for it
  - *waitingReview* - the code to implement the requirement has been written but with exception of the developer nobody has reviewed the code yet
  - *underReview* - the corresponding code is underReview and has not been accepted by the first reviewer yet
  - *reviewed* - the corresponding code has been reviewed and accepted by the first reviewer
  - *accepted* - the corresponding code has been accepted by the developer team and is ready to be published.
- **priority according to MoSCoW:** (the priority that is the base to decide when the feature will be implemented: m - MUST, S - Should, C - could, W - Won´t. For more information see *http://en.wikipedia.org/wiki/MoSCoW_method*))

Tests

The test files that are used as test input for a module are named according to the following pattern:

requirement-[id of requirement]-[number of test for this requirement].[file suffix]

Example: The first test file for the requirement 27 in an XML format is named "requirement-0027-001.xml".

If there are certain assertions written that can be automatically processed (e.g. by an schematron schema) than each assertion file has the corresponding file name of the test file (with file suffix of the assertion format). A schematron file that tests the output of a module that gets the test file "requirement-0027-001.xml" as input would be named as "requirement-0027-001.sch".