Documentation of the Subtitling Conversion Framework (SCF) - Draft 19/12/2014

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 an early beta release. The idea is to release early and to get feedback from the community in an early development stage. This enable us to integrate comments before the different modules have a stable status.

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.

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
  - *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.xml".

SCF Modules

STLXML-XSD

The STLXML W3C XML Schema is thought as a help for the further processing of STLXML files. STLXML files that don't validate against the STLXML Schema may lead to exceptions or unexpected results from modules that take STLXML files as input.

Prerequisites

- an XML validating parser (e.g. Xerces) or an schema aware XML editor

DESCRIPTION

The STLXML XSD defines a structure for an XML representation of en EBU STL (see resources). The validation restrictions of the XSD should not lead to false negative messages (so any XML document that is not valid against the STLXML-XSD has not the necessary characteristics of an STLXML). On the other hand it can lead to false positive messages (a XML document can validate against the XML Schema but does not meet the expectations of a valid STLXML document).

RESOURCES

EBU STL (EBU Tech 3264)

STL2STLXML

The STL2STLXML module converts a binary file that conforms to EBU-STL (EBU Tech 3264) into an XML representation of EBU-STL. The module is written in python.

Prerequisites

- Python 2.7.x

USAGE

stl2stlxml.py [SOURCE-STL-FILE] -x [DESTINATION-XML-FILE]

[SOURCE-STL-FILE] *Path to the source EBU-STL file that shall be translated.*

Parameters

```
     -x, --xml [DESTINATION-XML-FILE] Output file for the XML representation
of the stl file.
     -p, --pretty  Outputs the XML File in pretty XML (with indention).
```

DESCRIPTION

Decodes the EBU-STL file and exports in a XML representation that can be used for further processing with XML technologies or for debugging purposes.

EXAMPLES

```
python stl2stlxml.py test.stl -x test.xml
```

RESOURCES

EBU STL (EBU Tech 3264)

STLXML2EBU-TT

The requirements of the STLXML2EBU-TT are implemented in the SCF by the XSLT 1.0 stylesheet STLXML2EBU-TT.xslt. The XSLT takes as input an STLXML file (a XML representation of a binary EBU STL file). The output is an EBU-TT file that conforms to the EBU-TT Part 1 specification (EBU Tech 3350). The conversion is based on the guideline provided in EBU Tech 3360 (see Resources).

Prerequisites

- an XSLT 1.0 processor with EXSLT support (e.g. SAXON 6.5.5 or higher). Note that the latest SAXON 6.5.5 and the latest SAXON release 9.6 have been tested for EXSLT support but versions in between may not support EXSLT.

USAGE

STLXML2EBU-TT.xslt has the following parameters:

```
- offsetInSeconds
     A positive integer that defines in seconds the time-offset that's used
 for the TCI and TCO elements (default is 0)

- timeBase
    Either the value  'smpte' or 'media'. It sets explicitly  the ttp:timeBase
 attribute (default is SMPTE).
```

DESCRIPTION

The goal of the conversion is to create an EBU-TT-Part 1 file that can be used in archive and exchange based on an STLXML file. The results can therefore be seen as an EBU-TT representation of an STL file.

Amongst others STLXML files with the following characteristics are *not* supported:

- the DFC (Disc Format Code) is set to another values then "STL25.01" (25 fps)
- the TCS of a TTI element is set to 0 (timecode that is not intended for use)
- the Extension Block Number has another value than "FF"
- the STLXML does not validate against the STLXML XSD

EXAMPLES

If you use the the saxon parser (version 9.5) you could perform a transformation as follows:

```
java -cp saxon9he.jar net.sf.saxon.Transform -s:testStl.xml -xsl:STLXML2EBU-
TT.xslt -o:ebutt-out.xml
```

RESOURCES

EBU STL (EBU Tech 3264)

MAPPING EBU STL TO EBU-TT SUBTITLE FILES (EBU Tech 3360)

EBU-TT2EBU-TT-D

The implementation of the EBU-TT2EBU-TT-D module is based on the use case of the distribution of subtitles to online services with EBU-TT-D subtitle files. The module takes as input an EBU-TT File that is conformant to the EBU-TT Part 1 spec (EBU Tech 3350) and on the basis of the STL to EBU-TT mapping guideline.

This Submodule provides an XSL transformation for converting EBU-TT files into EBU-TT-D files. The folder named 'tests' contains two folders named 'schema' and 'test_files'. Within the 'test_files' folder there're files to test the requirement given by their respective name.

The 'schema' folder contains the corresponding XML schema files to test the result of the testfile being transformed. This structure can be used with an automated testdriver, e.g. an XProc pipeline, to provide automated testing.

Prerequisites

- an XSLT 1.0 processor (e.g. SAXON 6.5.5 or higher)

Usage

The EBU-TT2EBU-TT-D.xslt provides the following parameter:

```
 - offsetInSeconds
   Defines the time-offset that's used for the TCI and TCO elements in seconds
```

DESCRIPTION

This transformation assumes that the input EBU-TT file is based on an EBU-STL file and the transformation followed the guidelines of EBU Tech 3360.

EXAMPLES

```
java -cp saxon9he.jar net.sf.saxon.Transform -s:ebutt-part1.xml -xsl:EBU-
TT2EBU-TT-D.xslt -o:ebutt-d-out.xml
```

RESOURCES

EBU STL (EBU Tech 3264)

MAPPING EBU STL TO EBU-TT SUBTITLE FILES (EBU Tech 3360)

EBU-TT-D SUBTITLING DISTRIBUTION FORMAT (EBU Tech 3380)

EBU-TT-D2EBU-TT-D-Basic-DE

The implementation of the EBU-TT-D2EBU-TT-D-Basic-DE module is based on the use case of the distribution of subtitles to online services that uses the constrained EBU-TT-D format defined for the ARD Mediathek Portals. It takes as input an EBU-TT-D File that is conformant to the EBU-TT-D spec (EBU Tech 3380).

Prerequisites

- an XSLT 1.0 processor (e.g. SAXON 6.5.5 or higher)

Usage

The EBU-TT2EBU-TT-D.xslt has no parameters.

DESCRIPTION

This transformation assumes that the input EBU-TT-D file was created through the use of the SCF modules starting with an STL file.

EXAMPLES

```
java -cp saxon9he.jar net.sf.saxon.Transform -s:ebutt-d.xml -xsl:EBU-TT-D2EBU-
TT-D-Basic-DE.xslt -o:ebutt-d-basic-de-out.xml
```

RESOURCES

EBU-TT-D SUBTITLING DISTRIBUTION FORMAT (EBU Tech 3380)

XML-Format for Distribution of Subtitles in the ARD Mediathek portals (EBU-TT-D-Basic-DE)