UNIVERSITY OF EASTERN FINLAND

Algorithm demonstrator for Transducer Markup Language Transformations

Vladimir Ovsyukov
Master's Project
Department of Science and Forestry
University of Eastern Finland

May 2012

Prepared by:

Vladimir Ovsyukov

Accepted by:

Marko Hassinen

REVISION HISTORY

Revision no.	Date of Issue	Author(s)	Brief Description of Change
1.0	May, 2012	Vladimir Ovsyukov	1st draft.

Glossary

Term	**Definition**
XML	Extensible Markup Language
TML	Transducer Markup Language
XML Schema	Language to describe structure of a XML document in terms of constraints on the structure and content of documents of that type, using XML syntax
Algorithm	A sequence of finite instructions
Coordinate system	One of 3 coordinate systems TML utilizes defining methodology for positioning points within a spatial reference system
TML application (processor/controller)	A computing machine to understand, process and generate TML messages
TML process node	A process node presenting TML process on a network
TML transducer node	A node which presents transducer information on a network for transducer systems
TML transmitter	TML transducers" subdivision transforming input data into a remote action or phenomenon
API	Application Programming Interface

TABLE OF CONTENTS

1 Introduction

Up to this moment a plethora of applications for data transmissions between endpoints have been concerned. Many of whose are using XML-based proprietary/open-sourced protocols as transport. One of such is proposed by OpenGIS® Transducer Markup Language (TML), application and presentation level protocol for transmitting streaming or archived data, sensor data in a sensor system. The protocol contains descriptions of both the sensor data and the sensor system itself.

1.1 System Overview

The aim of this project is to implement one of sample algorithm demonstration applications for transducers of raw WAVE PCM (Pulse Code Modulation) audio files to TML (Transducer Markup Language) markup.

To complete this task we have to investigate binary format of wav audio files and TML specifications. The algorithm for accomplishing the task of transforming binary representation of audio file to TML, e.g. for binary delivering mechanisms, should be appointed.

1.1.1 WAVE audio file format

WAVE audio file is format of container targeted for containing of usually uncompressed audio sound. Here, the uncompressed sound is delivered after steps of digitization of quantized an analog signal. The principle of quantization and sampling for digital coding is demonstrated in Figure 1. There the red line is an originating analog signal.

Usual way to deliver digital formats in computer devises is to implement a self-contained file format comprising raw bytes' data representation along with somehow implemented mechanisms of data format (for i.e. how interpreting the raw bytes) appearance.

As in many other similar formats, wav files are being held in chunk-based form. Although, the specification for the format isn't strict, there is an overall chunk structure representation, implying at least obligatory presence of those chunks which are absolutely required in order to be able of proper reading of particular audio sound file.

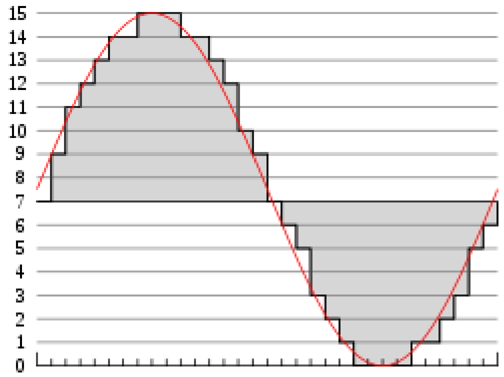


Figure 1. Sampling and quantization of a signal

Figure 2 shows basic chunks structure of wav file, where bigger outer chunk – RIFF chunk contains “fmt” and “data” sub chunks. There the all necessary information needed to properly parse the structure of an audio. It is described in “fmt” (stands for “format”) chunk. “Data” chunk is then contains sampled presentation of audio data to be transferred to a device’s sound card.

Chunks’ definition convention is such that every one of them is starting with 4- bytes ASCII characters name of the chunk, followed by 4 bytes size definition. To the end of chunk then other information can be held including arbitrary number of different fields’ definitions.

In Figure 2 the representation of “fmt” chunk is described together with sizes in byte:

- Subchunk1ID is name of chunk (4 bytes),
- Subchunk1ID size is size of it (4bytes),
- AudioFormat is a format of quantization of audio,
- NumChannels is a integer number of audio channels (1 for mono, 2 for stereo and so on),
- SampleRate – rate of appearance of samples in uniform time frame,
- ByteRate – amount of byte per second of audio,
- BlockAlign – quantity of bytes per one sample including all channels,
- BitsPerSample – amount of bit in one sample.

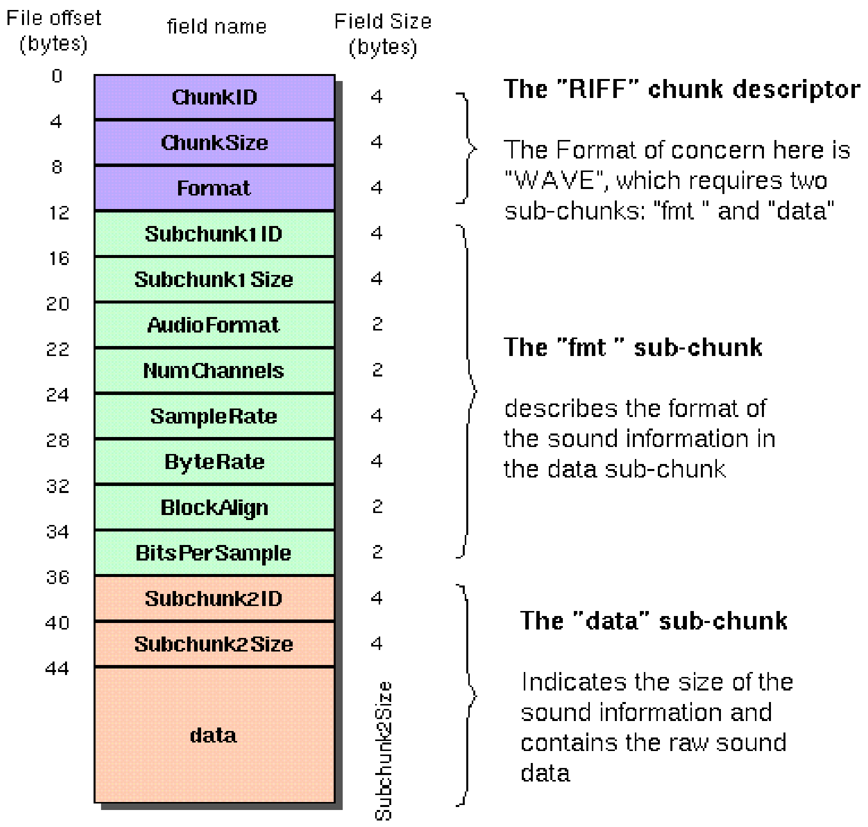


Figure 2. Canonical WAV file format

The stated above structure of "format" sub chunk can be extended with other fields.

1.1.2 Format of Transducer Markup Language' applications

The applications of TML based deliverables can be presented in a form of TML's "processes", with descriptive values definitions by means of "Cluster descriptions".

As it is being claimed in TML's specification:

"Transducer Markup Language (TML) is a language for capturing and characterizing not only data from transducers, but information necessary for the processing and understanding of that data by the eventual recipient of the transducer data. Both sensors and transmitters can be captured and characterized within TML, leading to the use of the term «transducer" rather than "sensor". TML handles not only static but also streaming transducer data. TML permits the data stream to handle live transducer data both being added to the stream and being deleted from the stream."

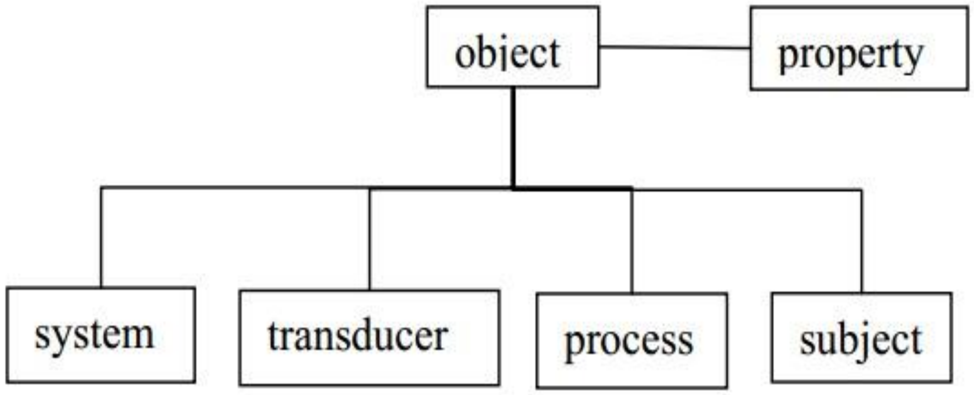


Figure 3. TML's conceptual components schema

2 Requirements

2.1 Project Requirements

Project requirements consist of designing and sketching implementation of algorithm for WAV PCM files transformations. They are to be transformed into TML based format file. Source file required to be Microsoft's PCM format and of mono/stereo type. Other files are skipped for simplicity.

Implementation:
· For implementation C# programming language is used;
· Microsoft's .Net platform of version 4.0, LINQ to XML API;
· Windows Presentation Foundation (WPF) is used for UI creation;
Documentation:
· Working implementation (source code);
· Final Report
· User Manual;
· Testing report;

2.2 Functional Requirements

Req. #	Requirement	Description
1	Wav file to be accepted	Interface for reading of audio wav PCM (version Microsoft wav PCM format) is presented
2	Audio is properly read	The chosen wav file structure is fully parsed into serialization classes
3	Transformation to TML by command	Serialized structure is transformed into TML like format and presented in readable form
4	Clear of interface and debug tooling	Presentation of TML form with possibilities to reload of a new file and investigate of their structure in binary form

3 Overall Architecture

The project was implemented for Microsoft's .Net platform with C# programming language, LING to XML API classes for working with XML and WPF for UI drawing. As an IDE Visual Studio 2010/11 Beta were chosen. The algorithm of manual serialization has been adhered.

The following picture demonstrates the overall classes' presentation in the project

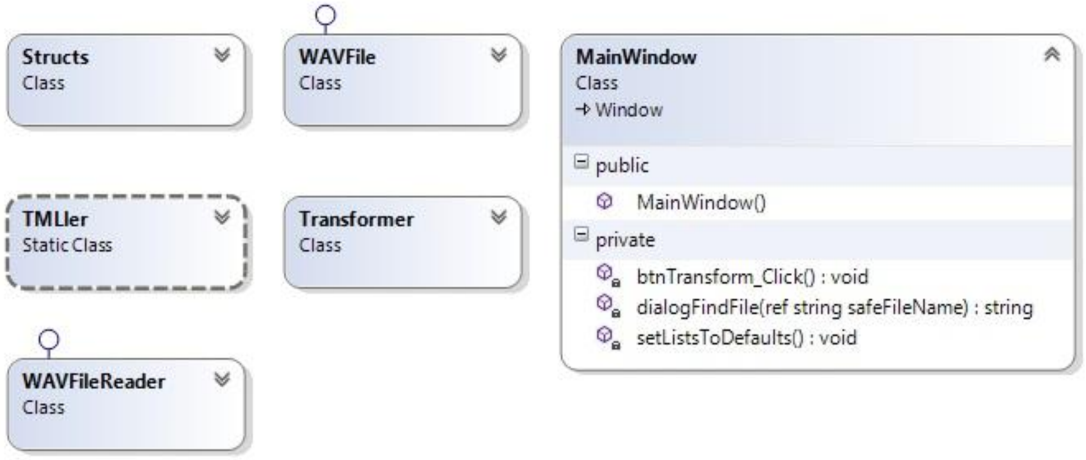


Figure 4. Overall classes' structure

And Figure 5 shows static class responsible for forming up XML document

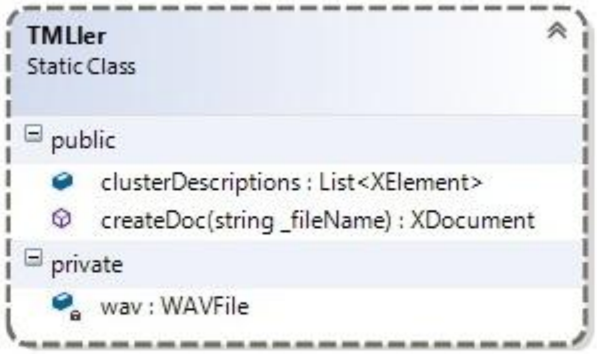
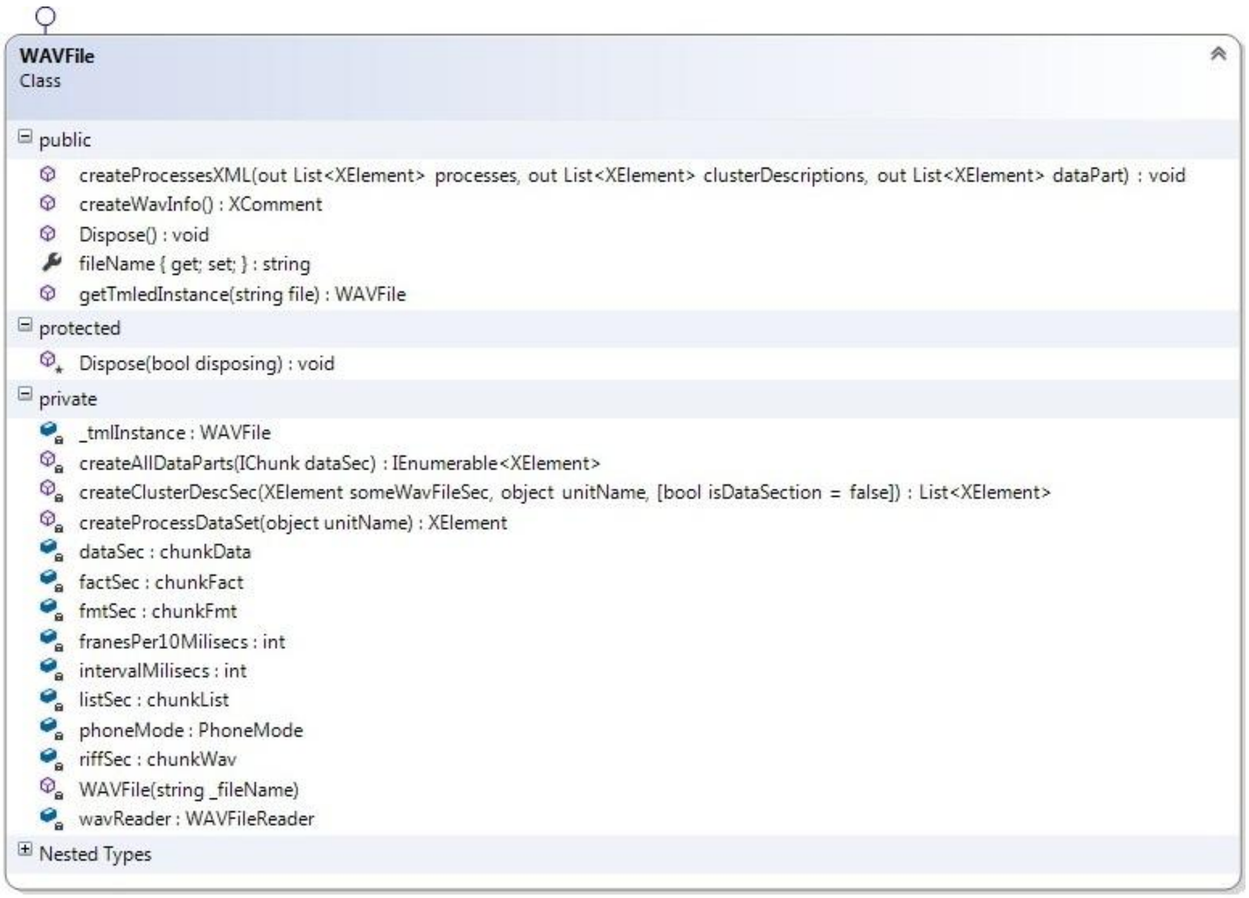

TMLler
Static Class
public
clusterDescriptions : List<XElement>
createDoc(string _fileName) : XDocument
private
wav : WAVFile

Figure 5. TMLler Class diagram

WavFile class implements the logic of serialization into TML. "c**reateProcessesXML**" method outputs the basic set of TML document parts – description of TML's processes, clusters' descriptions of the processes, raw data presentation:



WAVFile
Class

public

```
createProcessesXML(out List<XElement> processes, out List<XElement> clusterDescriptions, out List<XElement> dataPart) : void
createWavInfo() : XComment
Dispose() : void
fileName { get; set; } : string
getTmledInstance(string file) : WAVFile
```

protected

```
Dispose(bool disposing) : void
```

private

```
_tmlInstance : WAVFile
createAllDataParts(IChunk dataSec) : IEnumerable<XElement>
createClusterDescSec(XElement someWavFileSec, object unitName, [bool isDataSection = false]) : List<XElement>
createProcessDataSet(object unitName) : XElement
dataSec : chunkData
factSec : chunkFact
fmtSec : chunkFmt
franesPer10Milisecs : int
intervalMilisecs : int
listSec : chunkList
phoneMode : PhoneMode
riffSec : chunkWav
WAVFile(string _fileName)
wavReader : WAVFileReader
```

Nested Types

Figure 6. WAVFile Class diagramm

4 User Manual

On the Figure 7 graphical user interface can be seen, which present main window in minimalistic mode with “**TML**” button enabled to open up a dialog window choosing the wav file:

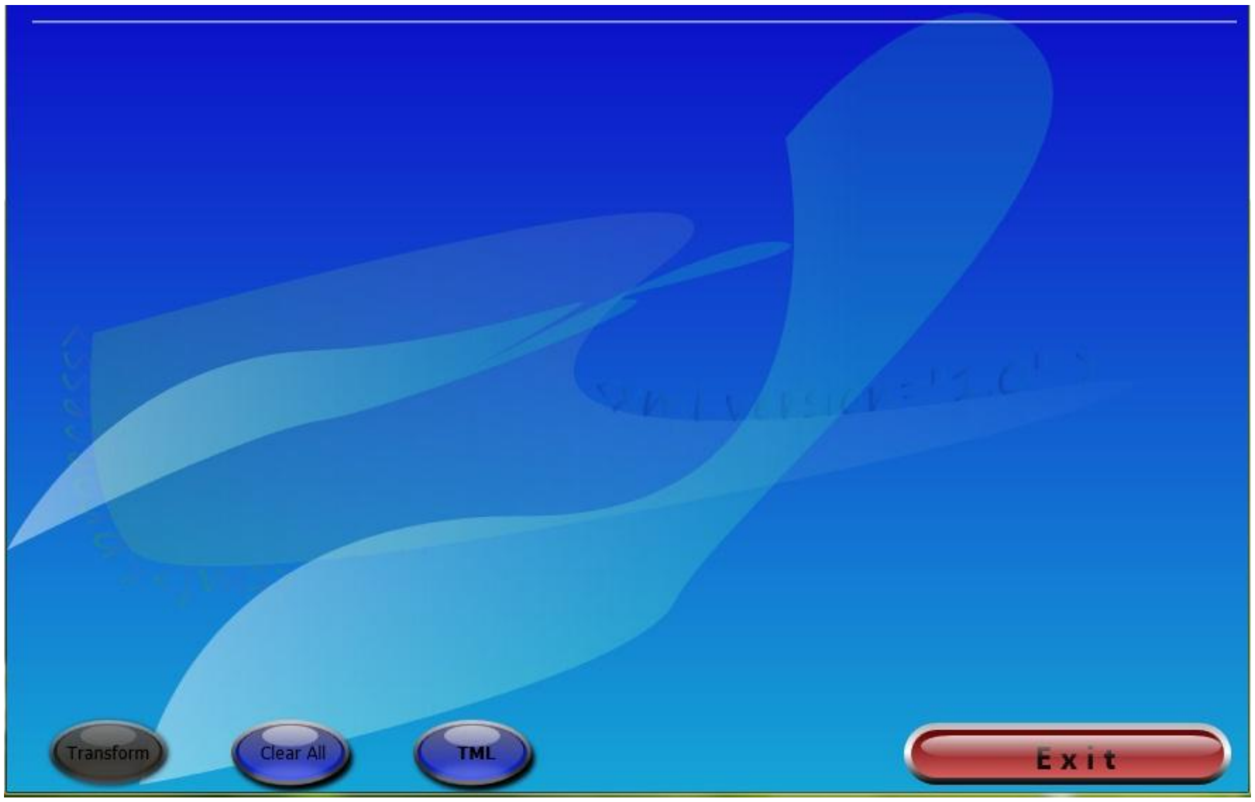
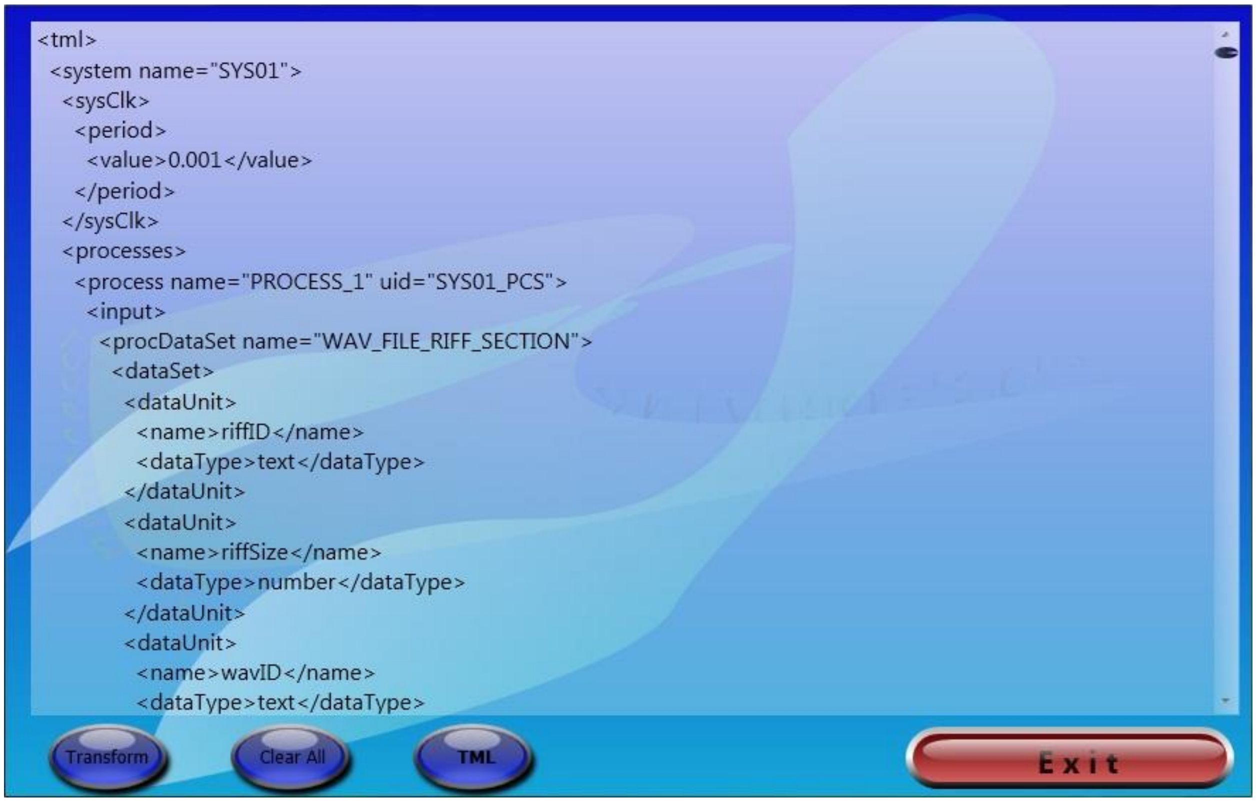


Figure 7. Main screen view



```
<tml>
 <system name="SYS01">
  <sysClk>
   <period>
    <value>0.001</value>
   </period>
  </sysClk>
  <processes>
   <process name="PROCESS_1" uid="SYS01_PCS">
    <input>
     <procDataSet name="WAV_FILE_RIFF_SECTION">
      <dataSet>
       <dataUnit>
        <name>riffID</name>
        <dataType>text</dataType>
       </dataUnit>
       <dataUnit>
        <name>riffSize</name>
        <dataType>number</dataType>
       </dataUnit>
       <dataUnit>
        <name>wavID</name>
        <dataType>text</dataType>
```

Figure 7. Main screen with scrollbox populated with TML view

Additionally, TML structure is being sent to text processing application (i.e. MS WordPad) of Operating System.

"**Tranform**" button is not available and only needed for debugging scenarios.

6 Testing report

This chapter describes the application testing of Algorithm demonstrator for Transducer Markup Language Transformations project. It contains list of all tests conducted and their results.

The test is based on following items:

1. Testing for – stands for application item to be tested,
2. How tested – stands for the item testing manner,
3. Expected results – stands for testing result expected,
4. Actual results – stands for the result obtained during testing procedure,
5. Passed? – stands for pointing whether the certain test was successfully passed or not.

The system test fails if the expected results do not coincide with the actual results. In the other case the test is considered to be passed.

Our task is to make tests for checking the ability of loading file into program and getting the proper results of application's working cycles, general test of UI.

Test name: *Loading executable assembly and opening WAV file.*

Testing for	**How Tested**	**Expected Result**	**Actual Result**	**Passed?**
Loading executable assembly and opening WAV file				
Load a Wav file	Click the button "**TML**"	TML based document returned	Assembly is loaded and on submitting Wav file application returns TML based document	Yes

Test name: Testing overall interface design of the project

Testing for	How Tested	Expected Result	Actual Result	Passed?
Verifying proper work of all butons in application				
Button “**Exit**” is pressed to exit application	After application loaded, the “**Exit”** button allows shutting it down	Application is shut down	Application is shut down	Yes
Button “**Clear**” is clearing listBoxs in main window	Press button	Filed listBox controls of main window is cleared	Filed listBox controls of main window is cleared	Yes
Long proceeding operations are not freezing the UI	Button “**TML**” is pressed and big file selected	Application stays responsive	Application freezes	No

Summary: General UI appears to be workable, although way for improving general usability features of the application are laying in better implementing application’s threading system, which can be for example reached by switching to TPL library.