TECHNISCHE UNIVERSITÄT
CHEMNITZ

Development of an Automation Test Setup for Navigation Data Processing

Master Thesis

for

the fulfillment of the academic degree

M.Sc. in Automotive Software Engineering

Faculty of Computer Science

Department of Computer Engineering

Submitted by: Dhruvjit Vilas Bhonsle, 29/12/1989

Supervisor: Prof. Dr. W. Hardt

Prof. Mirko Lippmann

Basti Anil Shenoy (Bosch GmbH)

Ralf Feil (Bosch GmbH)

David Ayemle (Bosch GmbH)

Dhruvjit Vilas Bhonsle (dhruvjit-vilas.bhonsle@s2012.tu-chemnitz.de)

Development of an Automation Test Setup for Navigation Data Processing

Master Thesis, Technische Universität Chemnitz, December 2015

Acknowledgement

This master thesis is an outcome of direct and indirect support from large group of people. Firstly, I would heartily convey my thanks to my mentors Prof. Dr. Wolfarm Hardt and Prof. Mirko Lippmann who gave important guidance and ideas during my presentations and before even I began my work.

I am also very much thankful to Dr. Ariane Heller who greatly supported me with organizing my thesis. Her prompt feedbacks were very helpful from beginning to an end of my thesis. It was due to guidance of Prof. Dr. Hardt, I was able to undertake my thesis work at Bosch, Abstatt in department of electronics engineering.

My thesis was well guided under the supervision of my mentor Mr. Basti Anil Shenoy. Under any circumstances it would not have been possible to finish my thesis work in given time period without his nonstop efforts. I would also like to thank Mr. Ralf Feil and Mr. David Ayemle for their technical inputs throughout my term. I am very grateful to my mentors at company for giving me chance to contribute my efforts for such an innovative project.

Also, I am very much thankful to my friends, especially Himanshu, Ketan and Mrinal for encouraging and helping me to achieve my goals. It was because of them I was able to plan my priorities right. Lastly, I want to express my love and gratitude towards my family members who continuously made efforts for boosting my work-life balance.

Thank you,

Dhruvjit

Abstract

With the development of Advanced Driving Assistance Systems (ADAS) vehicles have undergone better experience in field of safety, better driving and enhanced vehicle systems. Today these systems are one of the fastest growing in automotive domain. Physical parameters like map data, vehicle position and speed are crucial for the advancement of functionalities implemented for ADAS.

All the navigation map databases are stored in proprietary format. So for the ADAS application to access this data an appropriate interface has to be defined. This is the main aim of Advance Driver Assistant Systems Interface Specifications (ADASIS) consortium. This new specification allows a coordinated effort of more than one industry to improve comfort and fuel efficiency.

My research during the entire duration of my master thesis mainly focuses on two stages namely XML Comparator and CAN stream generation stages from ADASIS Test Environment that was developed in our company. In this test environment ADASIS Reconstructor of our company is tested against the parameters of Reference Reconstructor provided by ADASIS consortium. The main aim of this environment is to develop a Reconstructor which will adhere to all the specifications given in ADASIS Reconstructor.

My implementation in this master thesis focuses on two stages of test environment setup which are XML Comparison and CAN Stream Generation Tool respectively. Prior to my working, these stages lacked in-depth research and usability features for further working.

Table of Contents

List of Figures

List of Tables

Listings

1. Introduction

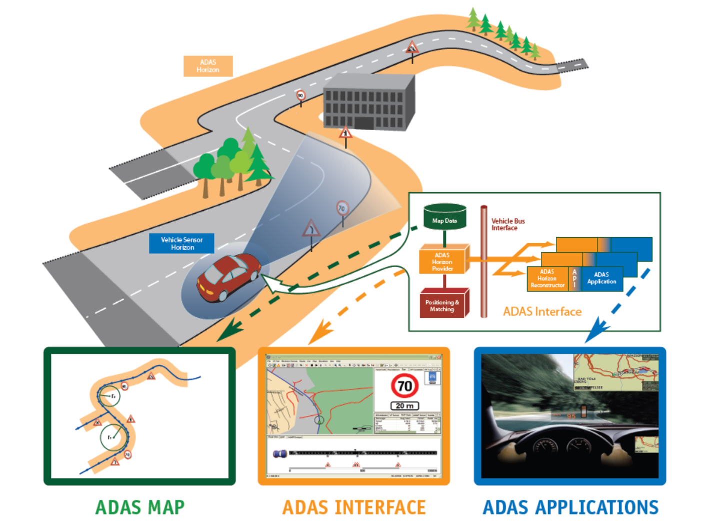


Figure 1.1: ADASIS Interface [1]

Advance Driver Assistant Systems Interface Specifications (ADASIS) is a standard protocol for compact representation and exchange of navigation information (digital map, position and speed). It provides a limited preview of road geometry (crossing, curve, slope…) around vehicle called the "ADAS Horizon" or "e-horizon".

The horizon preview is made available to ADAS application via standard interfaces. The preview is then used by applications as predictive sensor data to enhance their functionality.

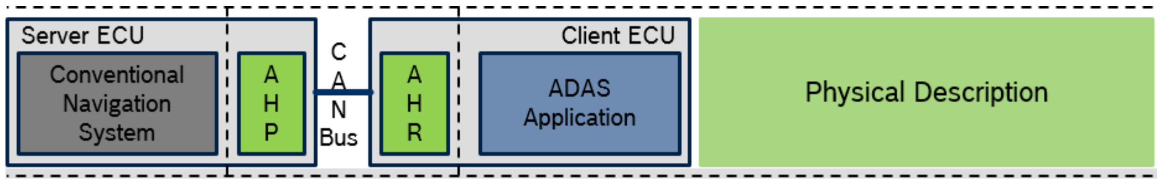


Figure 1.2: ADASIS Interface Description [1]

In Figure 1.2 Server ECU provides the horizon information to the client ECU and information is exchanged between ADAS horizon provider (AHP) and ADAS horizon reconstructor (AHR) via CAN bus. One thing to note here is server ECU is continuously updated with latest horizon information from the server. This setup is installed together in the car

1.1. Motivation

After standard objectives of ADASIS consortium were announced we wanted to develop our own implementation of reconstructor which would follow the guidelines of ADASIS reconstructor. The main entities that are part of ADAS applications are:

ADAS Horizon Provider – This maintains the ADAS Horizon

ADAS Protocol – This defines how ADAS Horizon will be sent from ADAS provider

ADAS Application – It is a client application that receives the ADAS Protocol messages then reconstructs and uses the ADAS Horizon [1]

Main motivation behind test environment is to verify the conformance of Bosch Reconstructor against Reference Reconstructor which is provided by ADASIS consortium. Moreover, to nullify the differences or errors if generated once the test setup was deployed there was need to come up with efficient XML data comparison tool because the information of horizon is specifically stored in XML format.

Similarly, for the CAN stream generation stage which is part of our test environment setup there was a need to develop robust tool that will allow developers to use user interface which will take manual input of test points and output CAN data.

Since horizon attributes are crucial for classifying data which will be discussed ahead there was a great requirement of research to develop these useful tools that will work on applying different set of rules for providing better usability to developers.

1.2. Objective

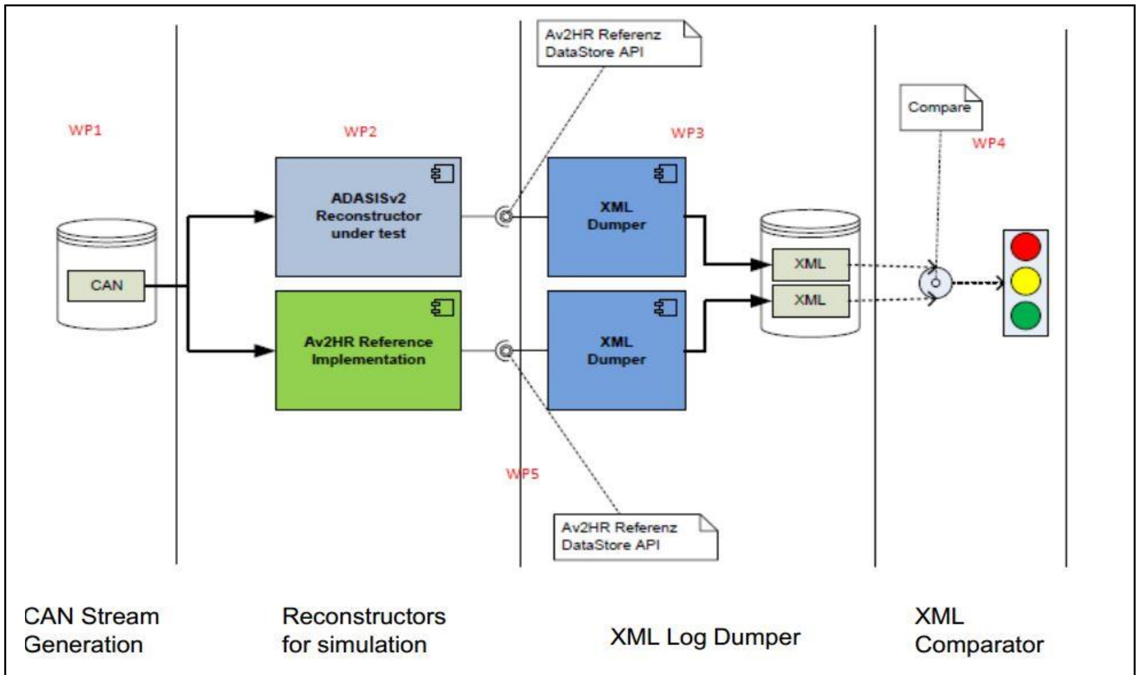

Av2HR Referenz DataStore API
Compare
WP1
WP2
WP3
WP4
ADASISv2 Reconstructor under test
XML Dumper
CAN
XML
XML
Av2HR Reference Implementation
XML Dumper
WP5
Av2HR Referenz DataStore API
CAN Stream Generation
Reconstructors for simulation
XML Log Dumper
XML Comparator

Figure 1.3: ADASIS Test Setup [2]

Test environment setup is divided into above given five work packages. The main objective of this thesis is to mainly undertake work package-1 and if time permits then focus on work package-4. Main implementation consists of deploying different functions and features for software tool based on specific research rules related with these two work packages.

First implementation stage consists of developing a working XML Comparison tool which is located in work package-4. This is current top priority of our organization and hence it was taken first into consideration.

Work package-1 is CAN stream generation stage where CAN data has to be generated using the CAN interface tool which will be developed in second implementation stage as it is not top priority task. CAN scripts are available for CAN data stream generation and there only lacks a solid user interface to handle them.

1.3. Organization of Thesis

This thesis is divided into nine chapters and each one with sub sections. Chapter one mainly deals with motivation and objectives.

In chapter two main focus is given on state of the art which comprises of technology available for four sub section – CAN Script Generator, Reconstructor, XML dumper and lastly XML Comparison Tool.

Chapter three sheds some light over the ADASIS v2 Protocol which is crucial to understand before mentioning concept of XML comparison. It consists of description of digital map database, Path profiles and messages.

Chapter four explains about research for development of XML comparator which covers necessity for this task and my approach to fulfill it. Similarly, Chapter five focuses on concepts for CAN script generator tool.

Chapter six shows implementation for XML comparator and CAN script generator. This covers explanation for implementation of algorithms and logics. Finally in Chapter seven working results are presented.

Lastly, to conclude in Chapter 8 summary is described which states conclusion, Challenges and Limitation. With last Chapter 9 I have marked the end of my thesis.

1.4. Type and Contribution

This master thesis is a combined mixture of “Research and Implementations” of software tools to gain desired output. After understanding the prevalent technology and ongoing requirement of Xml Comparator and CAN script generator. My contribution to thesis was:

- Develop working XML comparator
    - Research Comparison rules
    - Take order change into effect
    - Highlight errors
    - Saving results to XML file

- Develop working model of CAN stream generation
    - Research means to get data as input for defined range
    - GUI to receive input and output from user

2. State of the Art

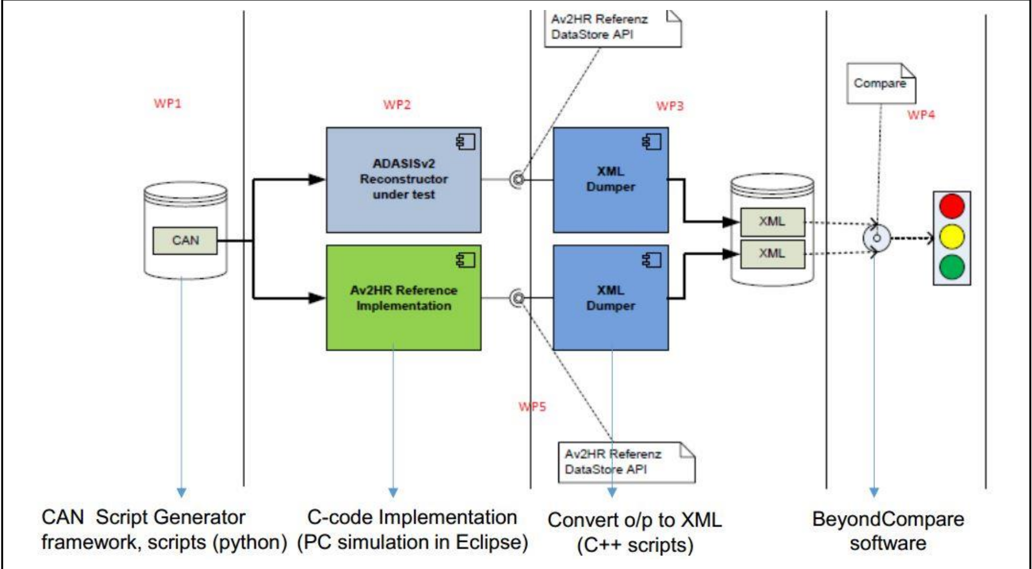

Av2HR Referenz DataStore API
Compare
WP1
WP2
WP3
WP4
CAN
ADASISv2 Reconstructor under test
Av2HR Reference Implementation
XML Dumper
XML Dumper
XML
XML
WP5
Av2HR Referenz DataStore API
CAN Script Generator framework, scripts (python)
C-code Implementation (PC simulation in Eclipse)
Convert o/p to XML (C++ scripts)
BeyondCompare software

Figure 2: ADASIS test setup state of the art [2]

The image shown in Figure 2 is the current situation of environment under test. The entire system is developed to put ADASISv2 Reconstructor (Bosch or Test Reconstructor) under test. This is done by comparing the difference between Reference Reconstructor and Bosch Reconstructor. Finally these differences can be known and eliminated from the Test Reconstructor to make it follow the functioning of Reference Reconstructor.

Our test setup is divided into five work packages and for every stage there is a degree of development associated with it which is explained as below:

- [WP1] CAN test stream generation: Basic Python based framework and scripts are available
- [WP2] Simulation setup for executing the 2 reconstructors: Eclipse based simulation projects are available
- [WP3] XML log dumper: C++ based solution is available
- [WP4] XML log comparison: Currently manual file comparison done using BeyondCompare tool
- [WP5] Integration of WP1 – WP4 into a single test automation setup: Currently no solution available.

The state of the art is further explained here covering all the stages shown in figure 2 in detail. Explanation from work package 1 to work package 5 is presented.

2.1. CAN Script Generator

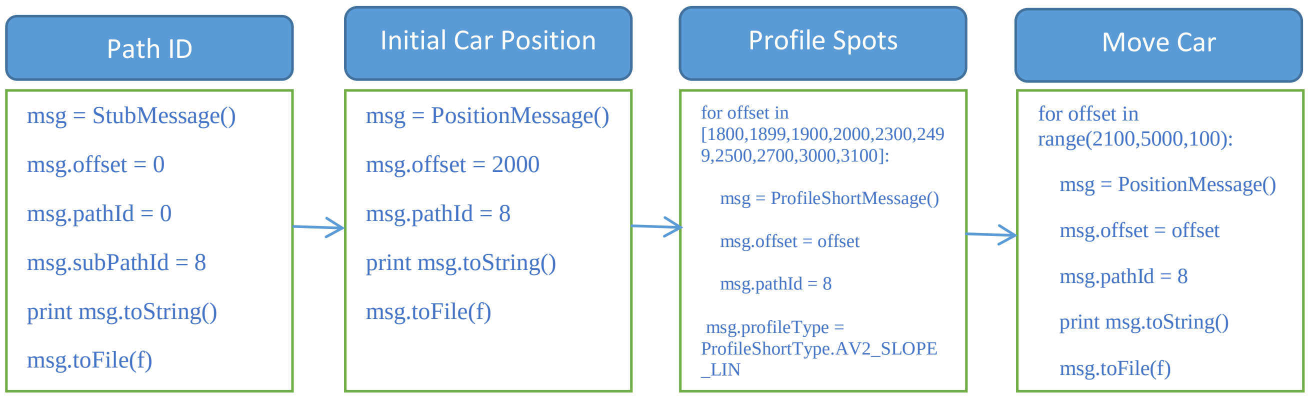
This is a python based framework which is used to generate CAN trace files. Test cases specified in the Test specification are implemented as Python scripts using the framework. The result of the script execution is suitable CAN file used as input for the Re-constructors.

Create a horizon on a single path with multiple profile spots within the horizon limits. Then move forward to see whether the entries behind are removed from the horizon. The removal of entries are done here because as the vehicle moves forward the entries from the past location are supposed to be removed as they hold no significance.

One of such script works on following parameters:

- single path: path ID = 8
- initial car position: 2000
- initial horizon: [1900,2500]
- profile type: Slope Linear
- profile spots: [1800,1899,1900,2000,2300,2499,2500,2700,3000,3100]
- move car: 2100, 2200, ….., 5000

Above data depicts different points that are needed for data deletion which is in reference to past details. The script which is making use of above details is able to generate CAN trace file. A short view of this script is explained below:



```
msg = StubMessage()
msg.offset = 0
msg.pathId = 0
msg.subPathId = 8
print msg.toString()
msg.toFile(f)
```

```
msg = PositionMessage()
msg.offset = 2000
msg.pathId = 8
print msg.toString()
msg.toFile(f)
```

```
for offset in [1800,1899,1900,2000,2300,2499,2500,2700,3000,3100]:
    msg = ProfileShortMessage()
    msg.offset = offset
    msg.pathId = 8
    msg.profileType = ProfileShortType.AV2_SLOPE_LIN
```

```
for offset in range(2100,5000,100):
    msg = PositionMessage()
    msg.offset = offset
    msg.pathId = 8
    print msg.toString()
    msg.toFile(f)
```

Figure 2.1 CAN stream generation script

2.1.1. CAN Trace View

When script runs as explained in Figure 2.1 it generates a CAN trace file which is in ASCII format as shown below. This file is later on fed to the Reconstructor which is covered in section 2.2.

```
0.000000 1  121        Rx   d 8 60 00 00 20 00 00 00 00  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 27 d0 08 00 00 00 00 00  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 87 08 08 20 00 00 2b ff  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 87 6b 48 20 00 00 2b ff  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 87 6c 88 20 00 00 2b ff  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 87 d0 c8 20 00 00 2b ff  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 88 fc 08 20 00 00 2b ff  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 89 c3 48 20 00 00 2b ff  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 89 c4 88 20 00 00 2b ff  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 8a 8c c8 20 00 00 2b ff  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 8b b8 08 20 00 00 2b ff  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 8c 1c 48 20 00 00 2b ff  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 28 34 48 00 00 00 00 00  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 28 98 88 00 00 00 00 00  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 28 fc c8 00 00 00 00 00  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 29 60 08 00 00 00 00 00  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 29 c4 48 00 00 00 00 00  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 2a 28 88 00 00 00 00 00  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 2a 8c c8 00 00 00 00 00  Length = 236000 BitCount = 122
0.000000 1  121        Rx   d 8 2a f0 08 00 00 00 00 00  Length = 236000 BitCount = 122
```

Figure 2.2: CAN Trace View

2.1.2. Main Use

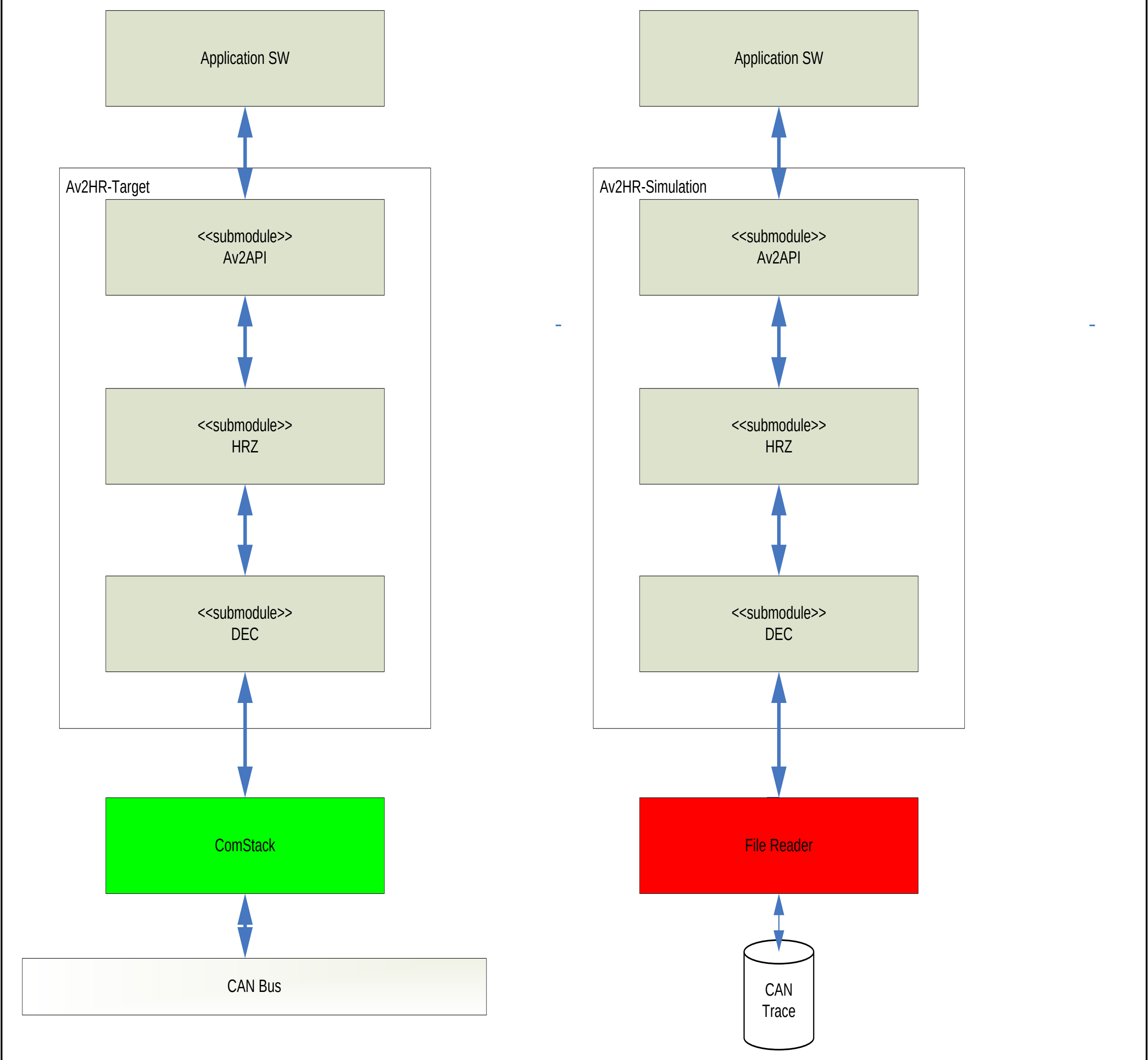


Figure 2.3: CAN stream generator use [2]

As shown in above figure that CAN stream generator is used for feeding the input of test setup with CAN data from trace file as opposed to the real time environment where data is fed from original CAN bus. Since the test setup is simulated on the computer workspace, CAN trace plays an important role in feeding setup with CAN trace. The real system and our simulated systems is same except only the CAN stream generation block.

2.2. Reconstructors

In the test setup which is shown in figure 2 there are two types of reconstructors used, firstly, the reconstructor which is put under test (test or Av2HR Bosch reconstructor) and other one is the reference reconstructor provided by ADASIS consortium (Av2HR-Ref Reference reconstructor)

An ADASISv2 Bosch reconstructor is “compliant” if at any given time a complete query of the eHorizon tree using the Av2HR DataStore-API matches the eHorizon tree of the Av2HR reference reconstructor while been feed with the identical ADASISv2 CAN stream and configured with the same trailing horizon length.

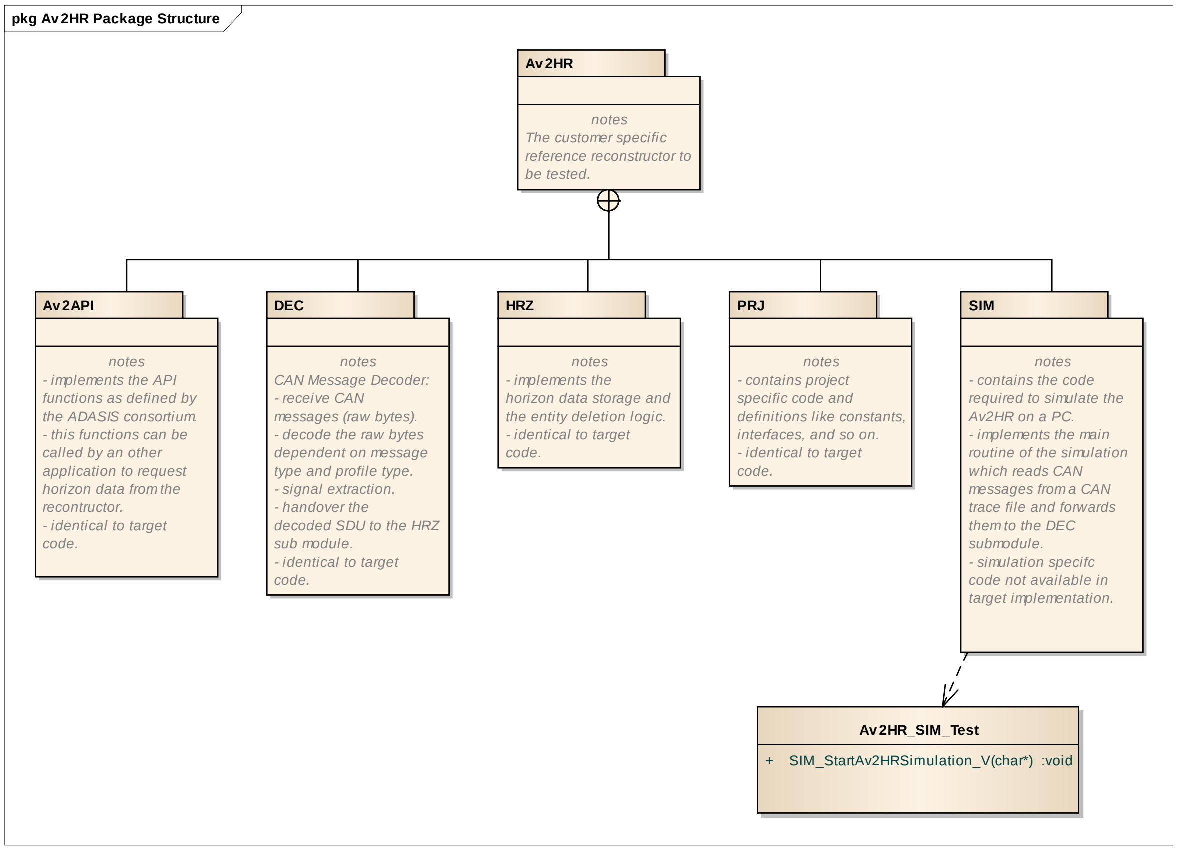

pkg Av2HR Package Structure
Av2HR
notes
The customer specific reference reconstructor to be tested.
Av2API
notes
- implements the API functions as defined by the ADASIS consortium.
- this functions can be called by an other application to request horizon data from the recontructor.
- identical to target code.
DEC
notes
CAN Message Decoder:
- receive CAN messages (raw bytes).
- decode the raw bytes dependent on message type and profile type.
- signal extraction.
- handover the decoded SDU to the HRZ sub module.
- identical to target code.
HRZ
notes
- implements the horizon data storage and the entity deletion logic.
- identical to target code.
PRJ
notes
- contains project specific code and definitions like constants, interfaces, and so on.
- identical to target code.
SIM
notes
- contains the code required to simulate the Av2HR on a PC.
- implements the main routine of the simulation which reads CAN messages from a CAN trace file and forwards them to the DEC submodule.
- simulation specifc code not available in target implementation.
Av2HR_SIM_Test
+ SIM_StartAv2HRSimulation_V(char*) :void

Figure 2.4: Package Structure of Reconstructors [2]

Above chart describes the package structure of reconstructors. All the packages listed above are same for test and reference reconstructor.

2.2.1. API for Information retrieval

To retrieve any information from the reconstructors we have to make use of specific DataStore API. To follow the ADASIS guidelines any compliant reconstructor has to provide following DataStore API:

```
av2hr_e Av2HR_getPosition (uint8_t index, av2hr_position_t* position)
av2hr_e Av2HR_getStub (av2hr_pathid_t pathId, av2hr_offset_t offset, uint8_t index, av2hr_stub_t* stub)
av2hr_e Av2HR_getProfileRange (uint16_t type, av2hr_pathid_t pathId, av2hr_offset_t* x0, av2hr_offset_t* x1)
av2hr_e Av2HR_getProfile (uint16_t type, av2hr_pathid_t pathId, av2hr_offset_t offset, av2hr_profiledesc_t* pd)
av2hr_e Av2HR_getLocation (uint16_t* countryCode, uint16_t* regionCode)
av2hr_e Av2HR_getDrivingSide (bool_t* drivingSideRight)
av2hr_e Av2HR_getSpeedUnit (bool_t* speedUnitMPH)
av2hr_e Av2HR_getProtocolVersion (uint8_t* major, uint8_t* minor, uint8_t* subminor)
av2hr_e Av2HR_getHardwareVersion (uint16_t* hardwareVersion)
av2hr_e Av2HR_getMapProvider (av2hr_mapprovider_e* mapProvider)
av2hr_e Av2HR_getMapVersion (uint16_t* mapVersionYear, uint8_t* mapVersionQuarter)
```

Figure 2.5: DataStore APIs

Similarly any compliant reconstructor also has to provide Event API which is mentioned:

```
av2hr_e Av2HR_onNewPosition (const av2hr_position_t* p)
av2hr_e Av2HR_onNewSegment (const av2hr_segment_t* p)
av2hr_e Av2HR_onNewStub (const av2hr_stub_t* p)
av2hr_e Av2HR_onNewProfile (const av2hr_profile_t* p)
av2hr_e Av2HR_onNewMetaData (const av2hr_metadata_t* p)
av2hr_e Av2HR_onMissingMessage (const av2hr_msgtype_e msgType)
```

Figure 2.6: Event API

2.2.2. Main Loop of Simulation

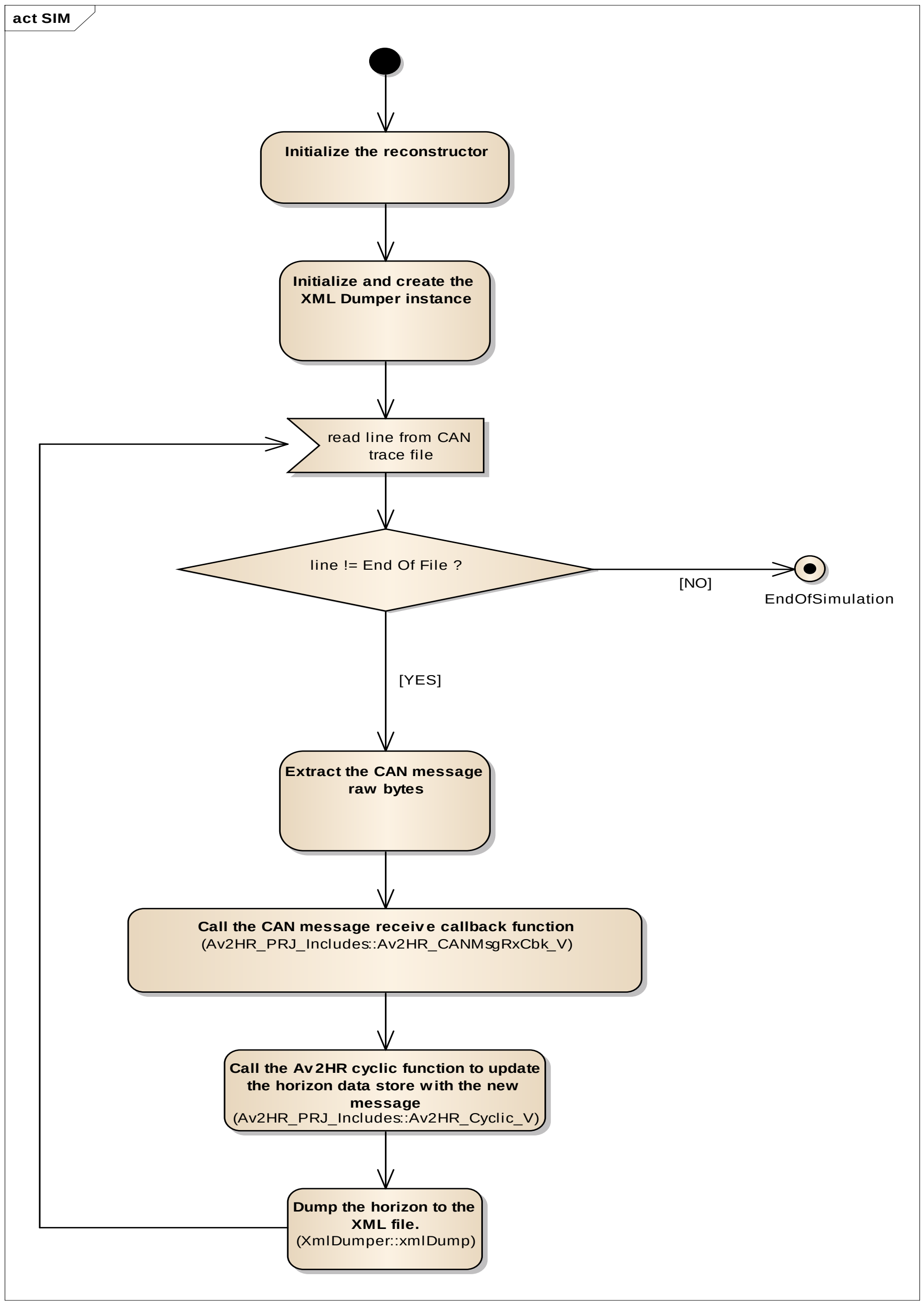

act SIM
Initialize the reconstructor
Initialize and create the XML Dumper instance
read line from CAN trace file
line != End Of File ?
[NO]
EndOfSimulation
[YES]
Extract the CAN message raw bytes
Call the CAN message receive callback function (Av2HR_PRJ_Includes::Av2HR_CANMsgRxCbk_V)
Call the Av2HR cyclic function to update the horizon data store with the new message (Av2HR_PRJ_Includes::Av2HR_Cyclic_V)
Dump the horizon to the XML file. (XmlDumper::xmlDump)

Figure 2.7 Main simulation loop [2]

In the main loop of simulation we can see that the simulation reads from a CAN trace file, extracts the raw data bytes and afterwards directly calls Av2HR functions of the original target reconstructor. At the end of the process we will have fully dumped XML file which will have all the information of horizon.

2.3. XML Dumper

The XML Dumper is querying the eHorizon tree using the Av2HR DataStore API after each new CAN packet that has arrived and processed by the reconstructor. The XML-Dumper has been integrated into the Av2HR-Reference simulation. It is called in the main loop of the simulation each time a CAN message has been processed.

The task of the XML-Dumper is to request the complete horizon from an Av2HR reconstructor and to dump this horizon into an XML file. To request the horizon data from the Av2HR it uses the standard ADASIS Av2 API which is implemented by the reconstructor. The data received by the reconstructor is mapped to the XML format which has been specified by the ADASIS consortium. Finally the XML result is written to an XML file.

The following sections will give some details about the XML-Dumper architecture and its implementation.

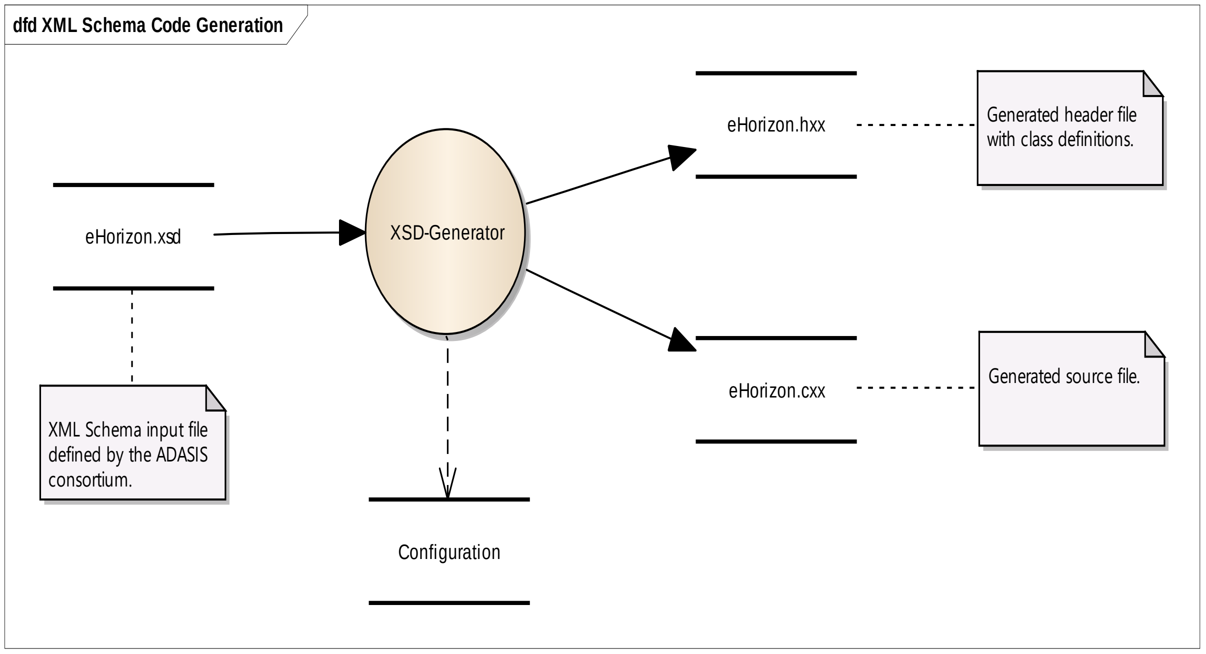

dfd XML Schema Code Generation
eHorizon.xsd
XSD-Generator
eHorizon.hxx
eHorizon.cxx
Configuration
Generated header file with class definitions.
Generated source file.
XML Schema input file defined by the ADASIS consortium.

Figure 2.8: XML Code Generation [2]

The format and contents of the XML file / dump is defined in a XML DTD by the ADASIS consortium. This XML schema file is the input file of a so called XML schema to C++ data binding compiler. In this project the open source compiler CodeSynthesis XSD is used for this purpose (XSD-Generator). It generates C++ classes which realize and cover all the XML functionality needed to generate the XML dump. The following excerpt from the CodeSynthesis website explains what is does and what the benefits are:

“CodeSynthesis XSD is an open-source, cross-platform W3C XML Schema to C++ data binding compiler. Provided with an XML instance specification (XML Schema), it generates C++ classes that represent the given vocabulary as well as parsing and serialization code. You can then access the data stored in XML using types and functions that semantically correspond to your application domain rather than dealing with the intricacies of reading and writing XML.”

Classes shown in Figure 2.9 implement an abstract and easy to use XML interface.

2.3.1. Deployment

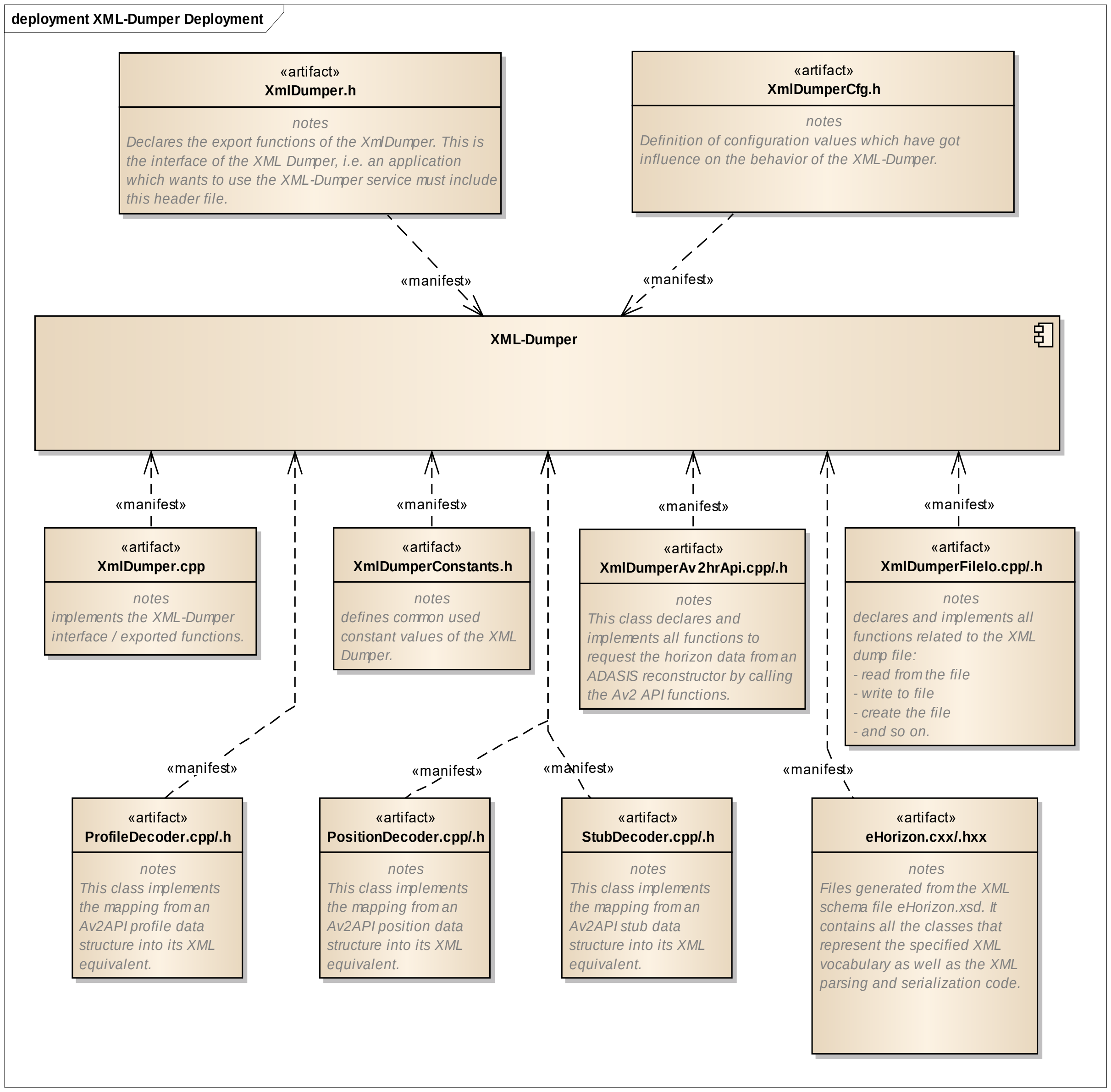

deployment XML-Dumper Deployment
«artifact»
XmlDumper.h
notes
Declares the export functions of the XmlDumper. This is the interface of the XML Dumper, i.e. an application which wants to use the XML-Dumper service must include this header file.
«artifact»
XmlDumperCfg.h
notes
Definition of configuration values which have got influence on the behavior of the XML-Dumper.
«manifest»
«manifest»
XML-Dumper
«manifest»
«manifest»
«manifest»
«manifest»
«artifact»
XmlDumper.cpp
notes
implements the XML-Dumper interface / exported functions.
«artifact»
XmlDumperConstants.h
notes
defines common used constant values of the XML Dumper.
«artifact»
XmlDumperAv2hrApi.cpp/.h
notes
This class declares and implements all functions to request the horizon data from an ADASIS reconstructor by calling the Av2 API functions.
«artifact»
XmlDumperFileIo.cpp/.h
notes
declares and implements all functions related to the XML dump file:
- read from the file
- write to file
- create the file
- and so on.
«manifest»
«manifest»
«manifest»
«manifest»
«artifact»
ProfileDecoder.cpp/.h
notes
This class implements the mapping from an Av2API profile data structure into its XML equivalent.
«artifact»
PositionDecoder.cpp/.h
notes
This class implements the mapping from an Av2API position data structure into its XML equivalent.
«artifact»
StubDecoder.cpp/.h
notes
This class implements the mapping from an Av2API stub data structure into its XML equivalent.
«artifact»
eHorizon.cxx/.hxx
notes
Files generated from the XML schema file eHorizon.xsd. It contains all the classes that represent the specified XML vocabulary as well as the XML parsing and serialization code.

Figure 2.9: Deployment XML dumper [2]

Above diagram shows all files of the XML Dumper and describes its functionality. The functionality is deployed programmatically using different classes. The description of all classes inside files is also available in the individual blocks.

2.3.2. Application Interface

The following section describes how XML-Dumper functionality can be integrated into client Av2HR reconstructor.

The XML dumper artifact is a static library that can be used by another application to request an Av2 horizon using the Av2API functions and dump it to an XML file. The XML dumper internally uses Xerces-C for all XML handling. For this reason the Xerces Library is embed in the XML-Dumper library.

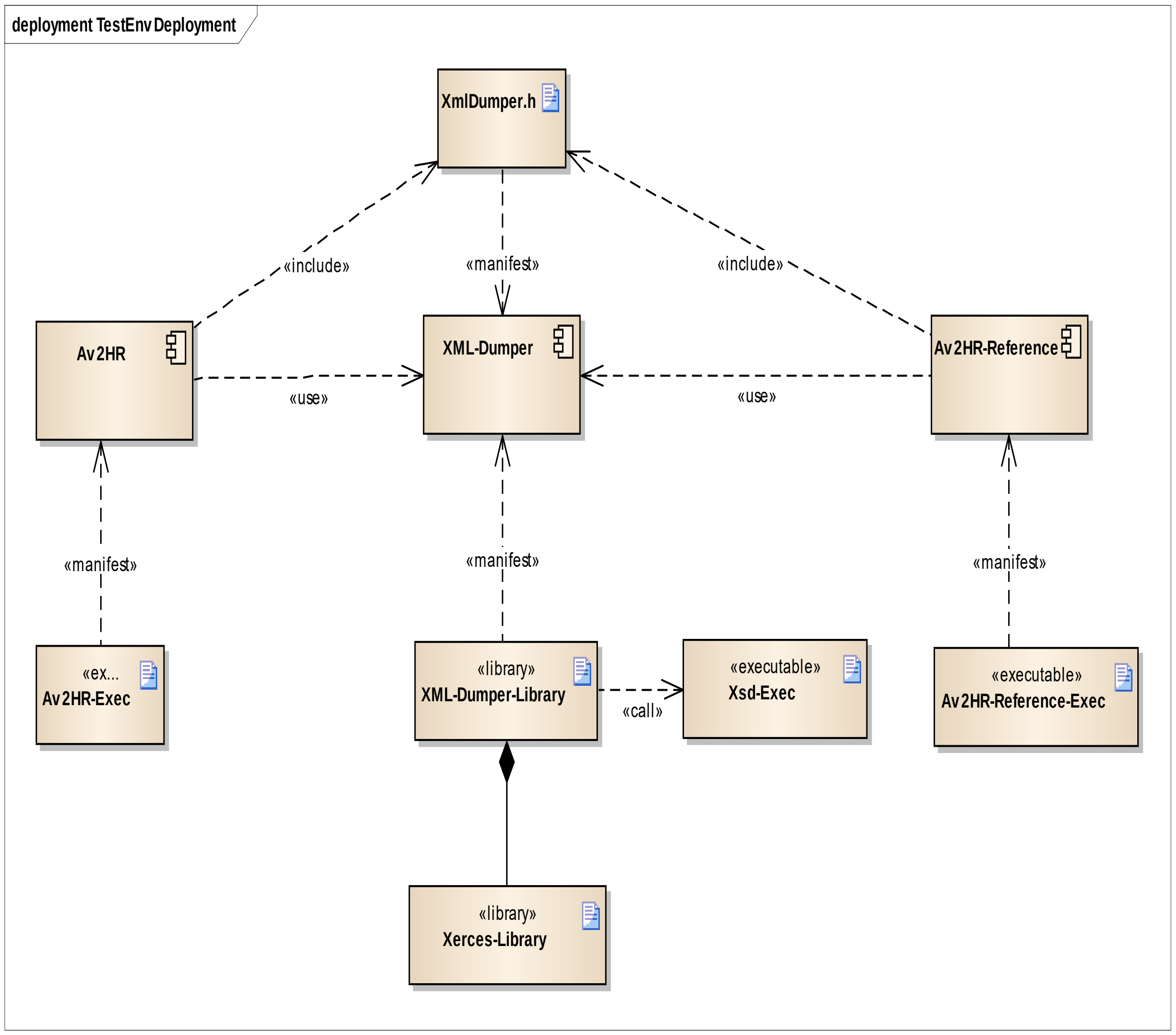


Figure 2.10: Application Interface [2]

Integration of the XML-Dumper into a reconstructor is explained thoroughly in below steps:

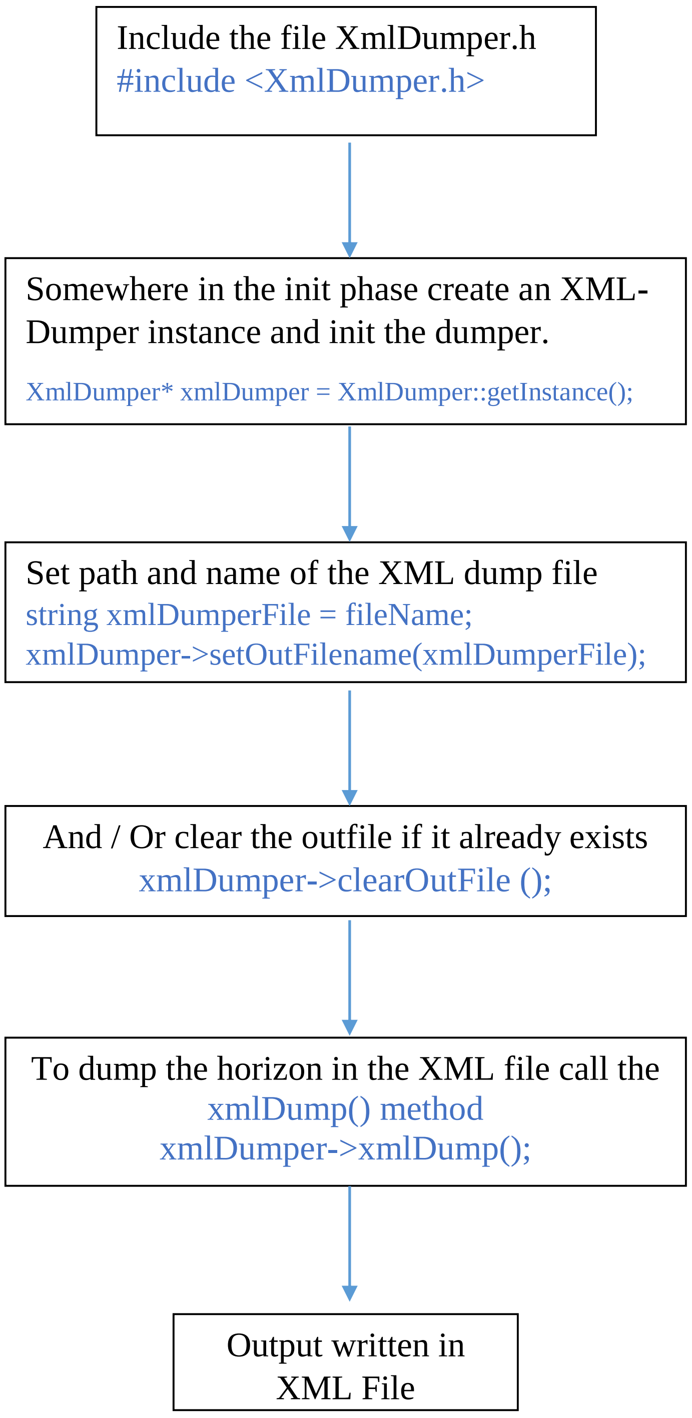

Include the file XmlDumper.h
#include <XmlDumper.h>
Somewhere in the init phase create an XML-Dumper instance and init the dumper.
XmlDumper* xmlDumper = XmlDumper::getInstance();
Set path and name of the XML dump file
string xmlDumperFile = fileName;
xmlDumper->setOutFilename(xmlDumperFile);
And / Or clear the outfile if it already exists
xmlDumper->clearOutFile ();
To dump the horizon in the XML file call the xmlDump() method
xmlDumper->xmlDump();
Output written in XML File

Figure 2.11: XML Dumper Integration

The basic structure of the XML created from XML Dumper looks like this:

```
<p1:time nTimestamp="18">

       <p1:car age="495" confidence="0" heading="0" index="0" lane="7" nTimestamp="0"
        offset="402" pathId="136" probability="40" speed="1660" vpStat="0"/>

       <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClass="2"
       lastStubAtOffset="true" numberOfLanesInDrivingDirection="2"
       numberOfLanesInoppositeDirection="0" offset="0" partOfCalculatedRoute="3"
       pathId="0" relativProbability="100" rightOfWay="2" subPathId="136" turnAngle="0"
       update="false">

              <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
              <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
              <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
              <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
              <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
              <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
              <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
              <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
              <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>
              <p1:profile offset="299" type="AV2_SLOPE_LIN">516</p1:profile>
              <p1:profile offset="323" type="AV2_SLOPE_LIN”>516</p1:profile>
              <p1:profile offset="353" type="AV2_SLOPE_LIN">518</p1:profile>
              <p1:profile offset="391" type="AV2_SLOPE_LIN">520</p1:profile>
              <p1:profile offset="448" type="AV2_SLOPE_LIN">520</p1:profile>
              <p1:profile offset="458" type="AV2_SLOPE_LIN">520</p1:profile>
              <p1:profile offset="548" type="AV2_SLOPE_LIN">523</p1:profile>
              <p1:profile offset="613" type="AV2_SLOPE_LIN">525</p1:profile>
              <p1:profile offset="676" type="AV2_SLOPE_LIN">525</p1:profile>
              <p1:profile offset="299" type="AV2_ROAD_CONDITION">8</p1:profile>
              <p1:profile offset="323" type="AV2_ROAD_CONDITION">8</p1:profile>
              <p1:profile offset="448" type="AV2_ROAD_CONDITION">8</p1:profile>
              <p1:profile offset="548" type="AV2_ROAD_CONDITION">8</p1:profile>
              <p1:profile offset="613" type="AV2_ROAD_CONDITION">8</p1:profile>
```

Figure 2.12: XML File created from Dumper

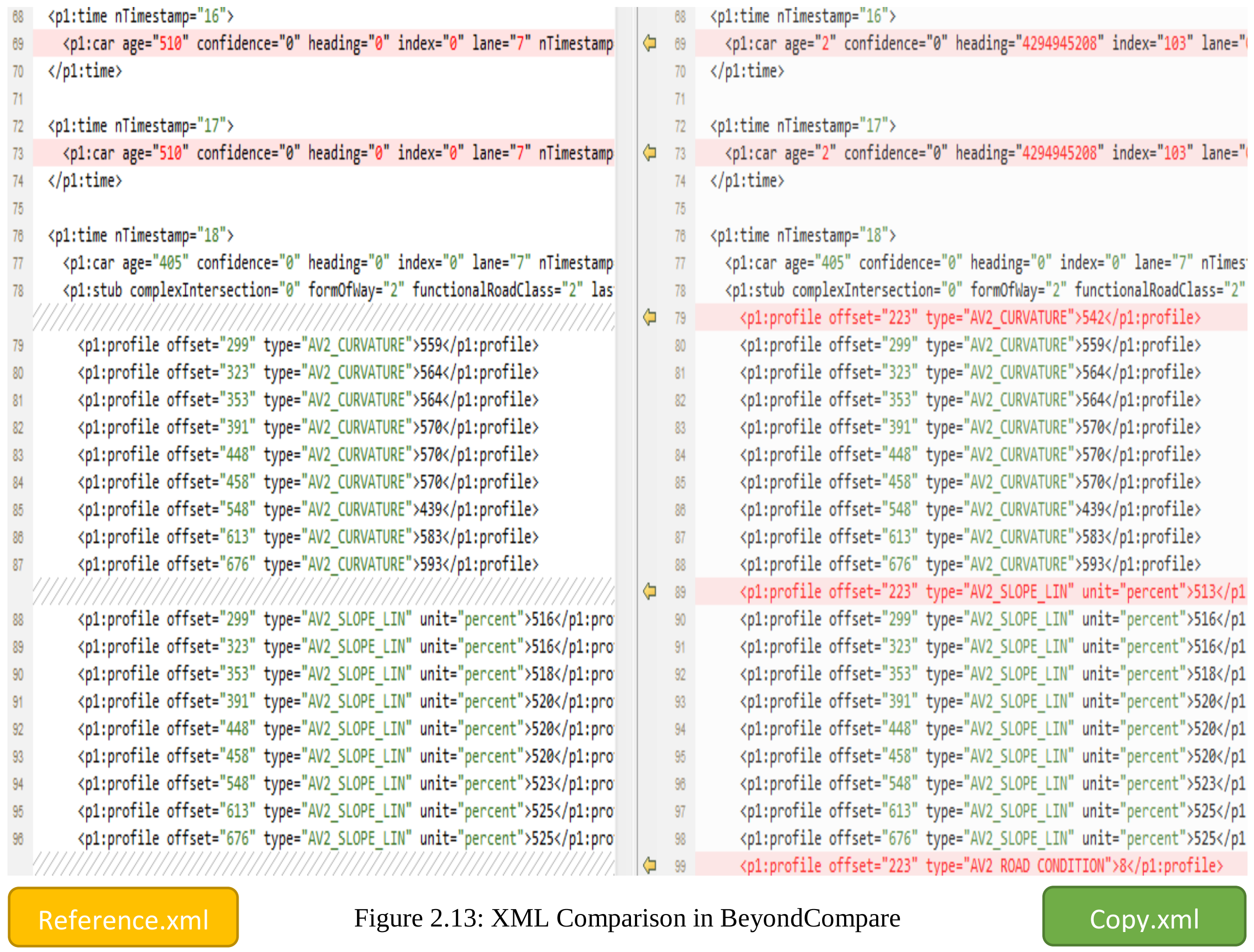
Each time the complete eHorizon is queried, the result is written into a new <time nTimestamp="">…</time> section and appended to the existing XML file. So at the end of a dumper session the complete eHorizon history since start is available.

To create the XML output, the entire eHorizon tree is queried starting from the tree root using the Av2HR DataStore API. Further explanation of XML file will be covered in upcoming section of AV2HR Protocol.

Using this file any horizon related query can be found out at specific time or position.

2.4. XML Comparison

XML comparison is the last stage of the test setup which is mentioned in Figure 2. Currently there is no implementation for this task which can apply specific rules and take results for order change. As far as comparison is concerned, we make use of software called BeyondCompare which uses a line to line comparison technique. The comparison window is presented below about how the differences look in BeyondCompare:



```
<p1:time nTimestamp="16">
  <p1:car age="510" confidence="0" heading="0" index="0" lane="7" nTimestamp
</p1:time>

<p1:time nTimestamp="17">
  <p1:car age="510" confidence="0" heading="0" index="0" lane="7" nTimestamp
</p1:time>

<p1:time nTimestamp="18">
  <p1:car age="405" confidence="0" heading="0" index="0" lane="7" nTimestamp
  <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClass="2" las

    <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
    <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
    <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
    <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>

    <p1:profile offset="299" type="AV2_SLOPE_LIN" unit="percent">516</p1:pro
    <p1:profile offset="323" type="AV2_SLOPE_LIN" unit="percent">516</p1:pro
    <p1:profile offset="353" type="AV2_SLOPE_LIN" unit="percent">518</p1:pro
    <p1:profile offset="391" type="AV2_SLOPE_LIN" unit="percent">520</p1:pro
    <p1:profile offset="448" type="AV2_SLOPE_LIN" unit="percent">520</p1:pro
    <p1:profile offset="458" type="AV2_SLOPE_LIN" unit="percent">520</p1:pro
    <p1:profile offset="548" type="AV2_SLOPE_LIN" unit="percent">523</p1:pro
    <p1:profile offset="613" type="AV2_SLOPE_LIN" unit="percent">525</p1:pro
    <p1:profile offset="676" type="AV2_SLOPE_LIN" unit="percent">525</p1:pro
```

```
<p1:time nTimestamp="16">
  <p1:car age="2" confidence="0" heading="4294945208" index="103" lane="
</p1:time>

<p1:time nTimestamp="17">
  <p1:car age="2" confidence="0" heading="4294945208" index="103" lane="
</p1:time>

<p1:time nTimestamp="18">
  <p1:car age="405" confidence="0" heading="0" index="0" lane="7" nTimes
  <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClass="2"
    <p1:profile offset="223" type="AV2_CURVATURE">542</p1:profile>
    <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
    <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
    <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
    <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>
    <p1:profile offset="223" type="AV2_SLOPE_LIN" unit="percent">513</p1
    <p1:profile offset="299" type="AV2_SLOPE_LIN" unit="percent">516</p1
    <p1:profile offset="323" type="AV2_SLOPE_LIN" unit="percent">516</p1
    <p1:profile offset="353" type="AV2_SLOPE_LIN" unit="percent">518</p1
    <p1:profile offset="391" type="AV2_SLOPE_LIN" unit="percent">520</p1
    <p1:profile offset="448" type="AV2_SLOPE_LIN" unit="percent">520</p1
    <p1:profile offset="458" type="AV2_SLOPE_LIN" unit="percent">520</p1
    <p1:profile offset="548" type="AV2_SLOPE_LIN" unit="percent">523</p1
    <p1:profile offset="613" type="AV2_SLOPE_LIN" unit="percent">525</p1
    <p1:profile offset="676" type="AV2_SLOPE_LIN" unit="percent">525</p1
    <p1:profile offset="223" type="AV2_ROAD_CONDITION">8</p1:profile>
```

Figure 2.13: XML Comparison in BeyondCompare

A sample difference of two files produced from dumper shows up in above manner where missing data is replaced by empty space in another file. If there is any value change then it's highlighted with red mark.

3. ADASIS v2 Protocol

For understanding the basics and further implementation of master thesis it is very crucial to cover ADASIS v2 protocol (Advanced Driver Assistance Systems Interface Specifications).

ADASIS forum have developed a specification to describe the road geometry with its related attributes ahead of a vehicle based on the vehicle's position and a digital map (ADAS Horizon). ADASIS v2 protocol is the advancement of ADASIS v1 protocol which was implemented before and was successfully tested.

The general architecture of ADAS applications are given by below entities:

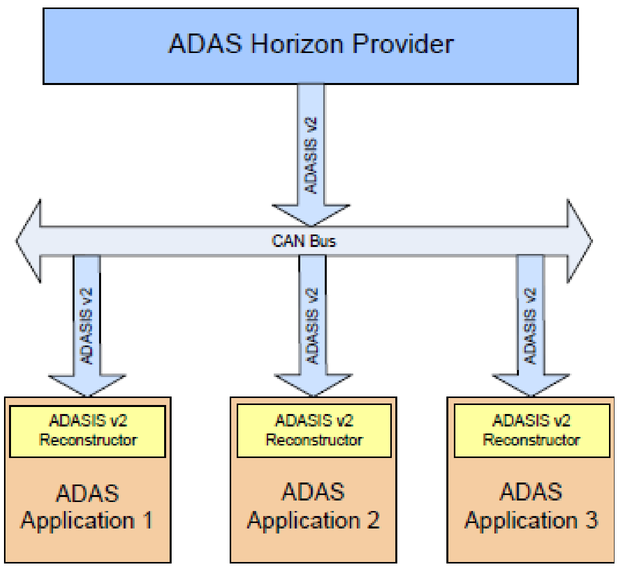


Figure 3.1: ADAS Application Architecture [3]

In the general architecture of ADAS applications, the main entities are:

ADAS Horizon Provider, which maintains the ADAS Horizon;

ADAS Protocol that defines how the ADAS Horizon will be sent from the ADAS Horizon Provider to the ADAS Applications;

ADAS Application is a client application that receives the ADAS Protocol messages then reconstructs and uses the ADAS Horizon;

ADAS Reconstructor as discussed in section 2.2 it is a common component of ADAS Applications that is built in accordance with this general architecture. The task of the ADAS Reconstructor is to receive, parse and interpret ADAS Protocol messages, and, in effect, reconstruct a copy of the ADAS Horizon on the client side.

3.1. Digital Map Database

In the digital map database, the road network is shown by collection of links and nodes which are shown in yellow and they define connectivity between links.

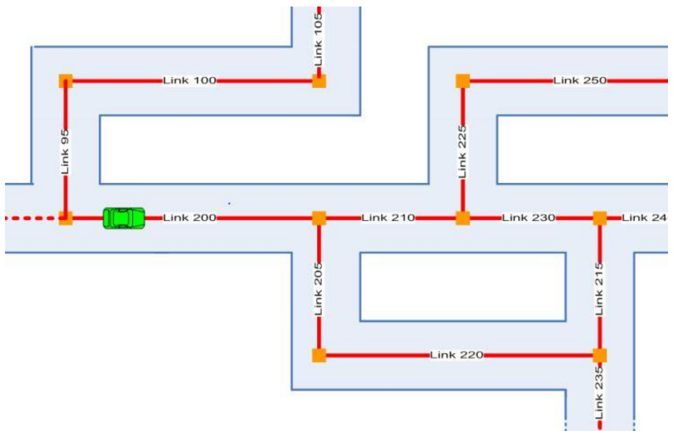


Figure 3.2: Digital Map [3]

In reference to map-enabled and map-enhanced ADAS application, the only roads of interest are those that are ahead of the vehicle and can be accessed in time. The ADAS Horizon (Electronic Horizon, eHorizon…) is the part of the digital map that contains only those roads in front of the vehicle.

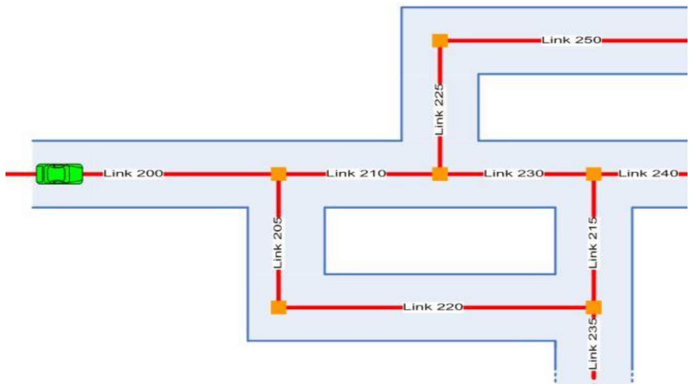


Figure 3.3: Only front view retained [3]

Comparing a simple extract of roads around the vehicle in Figure 3.2 and the ADAS Horizon in Figure 3.3, one can see that links 95, 100 and 105, being not important for majority of ADAS applications, are not in the ADAS Horizon. Moreover, not all link attributes available in the digital map need to be present on links of the ADAS Horizon.

So information like street names, housing number are not taken into much consideration. In other words, the ADAS Horizon provides to ADAS application an optimized view of the environment, allowing for more efficient processing.

For ADAS application analyzing ADAS Horizon is complex task. The reason is because as shown in figure 3.2 or 3.3 if our destination of interest is Link 235 then ADAS application should be able to reach it following 200->210->230->215->235 or 200->205->220->235.

It is more convenient for the ADAS application to deal with paths – trajectories the vehicle may follow in the near future. Internally, paths are built from database links and their connectivity, but each path is seen by the application as a single entity. [3]

3.1.1. Path Presentation

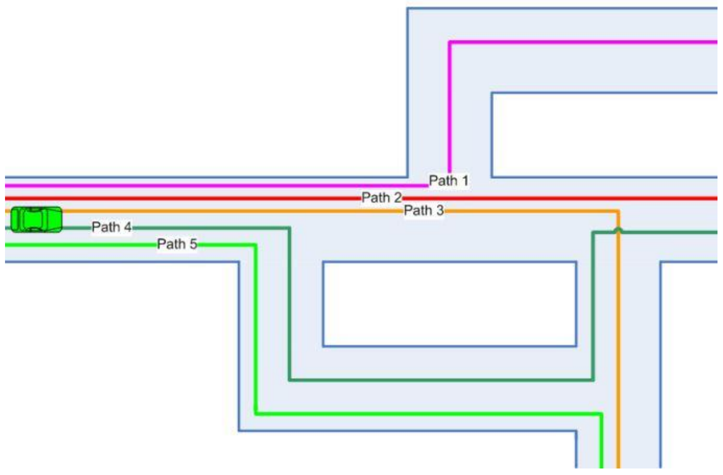


Figure 3.4: pure path representation [3]

On the basis of path representation presented in figure 3.4 applications can now easily recognize that there is a specific traffic sign on paths 3 and 5 (Figure 3.4). It may or may not be known that traffic signs on those two paths are the same physical traffic sign. For most applications, however, this information is not fundamentally important – the only significant information is that there is a traffic sign ahead and the distance to it. [3]

3.1.2. Optimized Path Presentation

This approach is used to reduce the amount of duplicated data but still providing advantages of path approach over network representation of ADAS Horizon. In Optimized Path Representation just one path needs to be present, this is called as the Main Path. The defining characteristic of the main path is that the current vehicle position is located on it. In Figure 3.4, Path 2 is the Main Path.

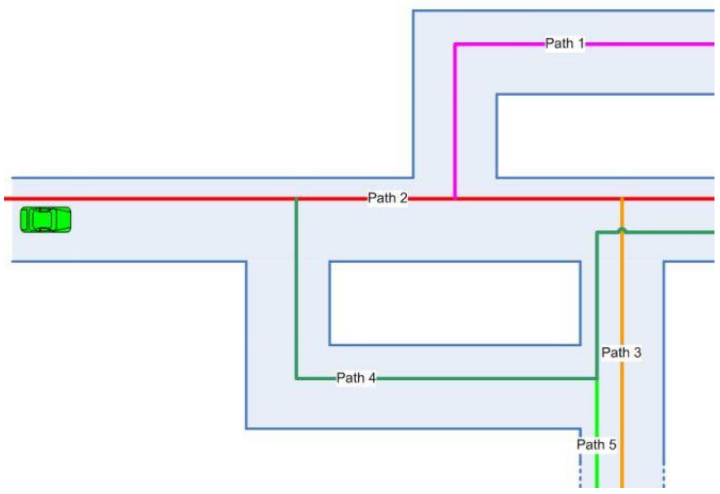


Figure 3.5: Optimized Path Representation [3]

The vehicle may turn from the Main Path to First-Level Sub-Paths. In the above example Paths 1, 3 and 4 are first-level sub-paths. From first-level sub-paths there are possible turns to Second-Level Sub-Paths (Path 5, for instance, from Path 4), etc. The ADAS Horizon's construction algorithm should choose the Main Path so that it appears to be the most likely alternative for the vehicle to continue driving. First-level sub-paths are less likely to be driven on and so on. It can be further seen in Application view point below.

On the ADAS Horizon, crossings, road attributes and even geometry may be seen only as characteristics of (one of) the paths. Therefore, Path is the main entity that needs to be accessed by the application in order to retrieve the desired information.

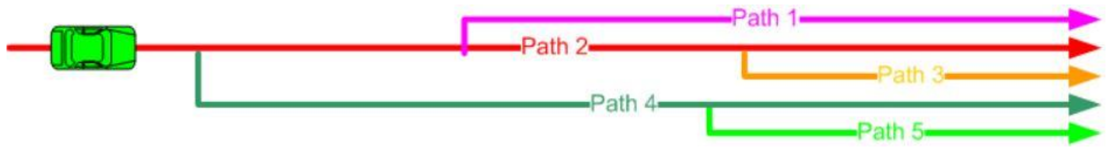


Figure 3.6: Application View [3]

3.2. Building Blocks

Stubs are entities that define the relationship between paths. Each stub marks the start of a sub-path and it is located on another parent path. The concepts of profiles will also be introduced, which are the characteristics of paths.

The position of Stubs and Profiles on the ADASIS v2 Horizon are defined by a path identifier and an offset along that path. All offsets are defined as the distance between an entity and the start of a path.

In general, ADASIS v2 Horizon Providers should initially consider and calculate all possible paths. However, the client ADAS application may need just a subset of those paths. Most applications will actually only need the main path. Therefore, in many installations it will be enough if the ADASIS v2 Horizon Provider (Av2HP) sends only the main path to the client (see Figure 3.7) [3]

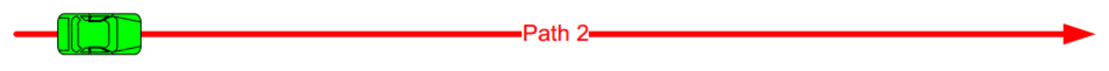


Figure 3.7: Main Path [3]

If only the main path has been provided, but the vehicle leaves it, the application will be “blind” for a moment until new information for the new path is available.

If the ADAS application also needs information for sub-paths, the ADASIS v2 Horizon Provider may provide transmitting preview information reaching into sub path structures. This is done by using "stubs" which indicate the start of a new path attached to a parent path, and which contain basic information about the attached road (e.g. turn angle, road class). This is shown in below Figure 3.8.

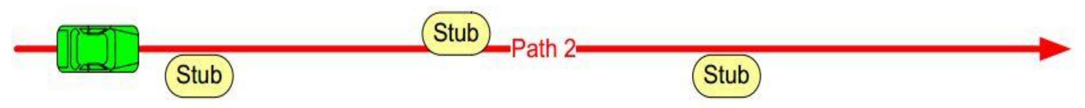


Figure 3.8: Main Paths with stubs [3]

In context of figure 3.10 if the application is sensitive to a short "blindness" when the vehicle leaves the main path then the Horizon Provider must also preventively transmit information about upcoming sub paths. With the availability of sub paths, the client will receive the information when the vehicle position has, for example changed from Path 2 to Path 1, no gap will occur in the available track preview data on the client side.

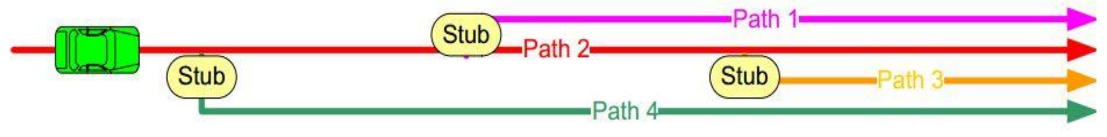


Figure 3.9: Main Paths with first level stubs [3]

In case memory is not a barrier for client and he is interested in more information, the ADASIS v2 Horizon Provider can also transmit the full available map information with higher-level sub paths as shown in Figure 3.10. In that case, the client will never be in a "blind" state and will always have immediate map information available when the vehicle leaves a path.

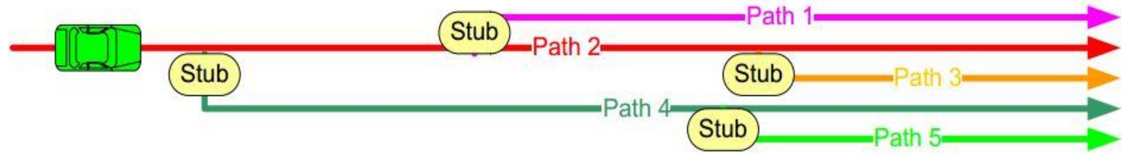


Figure 3.10: Full Horizon [3]

3.3. Paths and Offsets

As discussed, each path is uniquely identified by some number which is called Path Identifier, path identifier defines the position of map entities and the vehicle position and the distance from the start of the path. This along-path distance is called Offset. The start of each path is defined to be at offset zero.

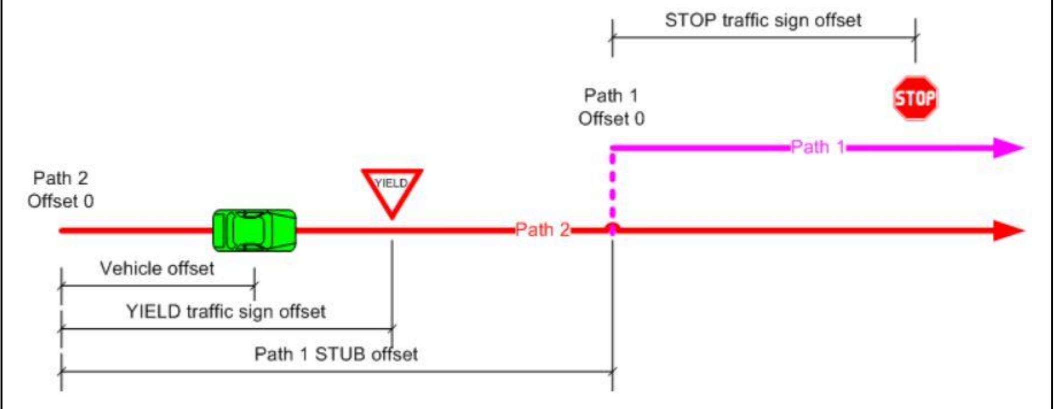


Figure 3.11: Offsets [3]

At intersections encountered on the main path, sub-paths may branch off. These entries into branches are called stubs. Like other map entities, the location of stubs along the ADASIS v2 Horizon is described by specifying the (parent) path identifier and offset.

Physical crossings on the path can be described as locations with one or more stubs – one for each alternate road at the crossing. From an abstract perspective, both the ADASIS v2 Horizon Provider and the ADASIS v2 Horizon Reconstructor/Client do not need to have any limitation of the number of paths present in each moment and of the length of each path.

A path has a defined start point (offset 0), but potentially no end at all. Therefore, the offset can be any large positive number. As we will see later, the ADASIS v2 Protocol defines only 6 bits for the identification of the path that is referenced by each CAN message. For the offset, just 13 bits are available. After removing special values, an ADASIS v2 message accommodates 56 numbers that can be used for path identification and offsets between 0 and 8190 meters inclusive. [3]

3.3.1. Path length limits

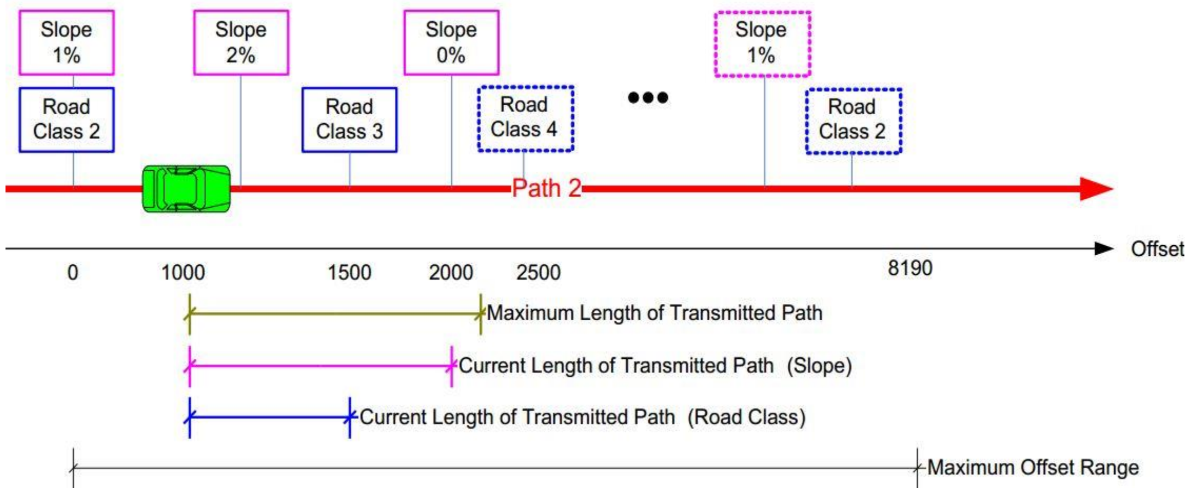
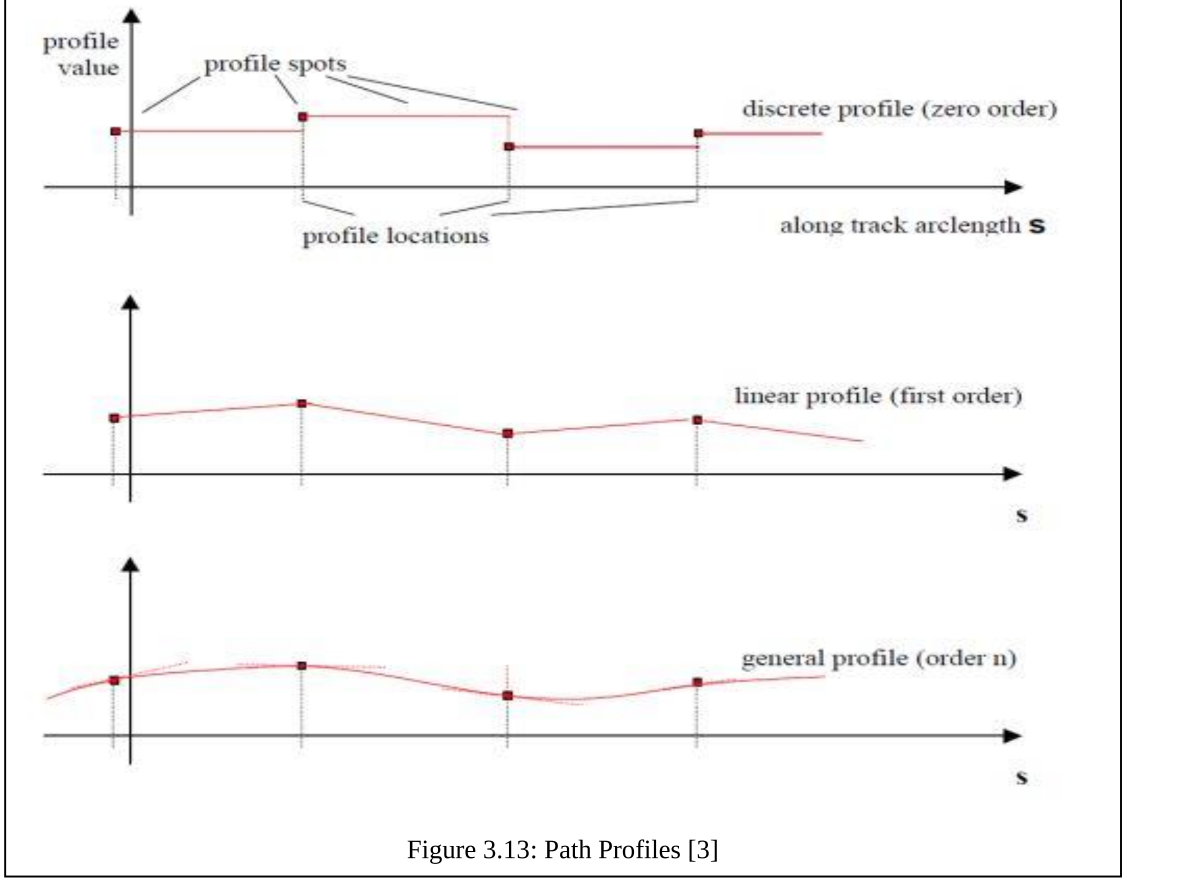


Figure 3.12: Path Length Limits [3]

- Figure 3.13 shows an example of the different length values.
- The current vehicle position is at offset value 1000m
- The "maximum offset range" is limited to 8190
- The "maximum length of transmitted path" is configured to 1200m
- The "current length of transmitted path" for the short profile slope is 1000m, because the provider has sent out a new slope information 0% at offset value 2000
- The "current length of transmitted path" for the segment message (containing the road class) is only 500m, because the farthest road class information is at offset 1500 and the next one at offset 2500 is too far away and has not yet been transmitted. [3]

3.3.2. Path Profiles

Path Profile is a property that has a value for any location along a path (e.g., curvature, form of way, number of lanes, speed limit, and horizontal geometry). A Path Profile is made up of a quantity of "Profile Spots" and the specification of a Profile Interpolation Type.


profile value
profile spots
discrete profile (zero order)
profile locations
along track arclength s
linear profile (first order)
s
general profile (order n)
s
Figure 3.13: Path Profiles [3]

- Profile Type is the property that the profile shall represent.

- Profile Interpolation Type is a code specifying how intermediate profile values are to be calculated.

- Profile Offset is a position along a path, defined by its distance from the path origin, measured as the arc or poly-line length along the path.

- Profile Spot is the numeric description value of a property at a Path Offset

3.4. Messages

ADASIS v2 undertakes one way communication between an ADASIS v2 Horizon Provider and ADASIS v2 Horizon Clients. Six types of messages are defined in the ADASIS v2.

3.4.1. Message Types

In this section, those messages are ordered by priority. Besides a list and descriptions of message fields, the CAN layout for each message is also defined. Five bits are common to every ADASIS v2 message:

1). the three bits define message type. This field is critical in multiplexing setups (Table 2), in a non-multiplexing CAN environment message type can be deduced from the CAN identifier of message (Table 3) and this field is redundant. TYPE field always occupies bits 7-5 of byte 0 of each CAN frame.

2). Two bits of each message are a cyclic message counter for each specific message type (and each profile type). It will be used to detect missing frames at the client side. [3]

The following message types are defined which are shown in Table1:

- POSITION message specifies the current position of the vehicle.
- STUB message indicates the start of a new path that has origin at an existing one.
- SEGMENT message specifies the most important attributes of a part of the path.
- PROFILE SHORT message describes attribute of the path whose value can be expressed in 10 bits.
- PROFILE LONG message describes attribute of the path whose value can be expressed in 32 bits.
- META-DATA message contains utility data.

3.4.2. Message Types Description

Message Type	Message
0	System Specific
1	Position
2	Segment
3	Stub
4	Profile Short
5	Profile Long
6	Meta-Data
7	Reserved

Table 1: Message Type Field

In each message, the 3-bit field Message Type determines semantics of rest of the message. Therefore, the content of the Message Type field determines what fields are present in each CAN frame used in the protocol. The minimum number of CAN identifiers that must be assigned to ADASIS v2 protocol is one.

CAN Identifier	Content of Message Type	Message
100	0	System Specific
100	1	Position
100	2	Segment
100	3	Stub
100	4	Profile Short
100	5	Profile Long
100	6	Meta-Data
n/a	7	Reserved

Table 2: ADASIS v2 as Multiplexed CAN Protocol

If multiplexing (polymorphism) of CAN frames is not desirable, different CAN identifiers can be assigned to the different messages just the way it is shown in Table 3 below. Most systems support single-level multiplexing in CAN traffic. If no other requirements are set, only the Message Type field can be used as the multiplexing field, since each message type has fixed structure.

CAN Identifier	Content of Message Type	Message
100	0	System Specific
100	1	Position
100	2	Segment
100	3	Stub
100	4	Profile Short
100	5	Profile Long
100	6	Meta-Data
n/a	7	Reserved

Table 3: ADASIS v2 as Non Multiplexed CAN Protocol

4. Concept of XML Comparison Tool

This part explains the concept and need to come up with xml comparison tool. Here the research prior to implementation of this tool is described in brief.

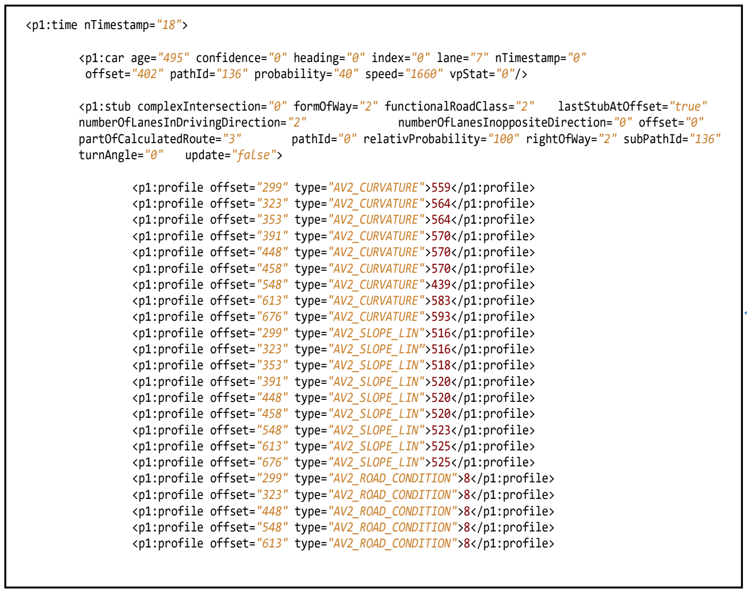
4.1. XML File

Before I discuss how research was undertaken for XML comparison it is mandatory that I explain the structure of XML file and type of data it represents. As explained in Figure 2.12 we just observed a XML file without going in brief for what it stands for.

4.1.1. XML File Description

This XML file as shown in figure 2.12 contains the information of map data. The time in this data file can range from 1 to 330…. timestamps. Here for the scope of this document we have just considered 18th timestamp. It represents and sums up in following manner:

Map Data = Links + Nodes + Road Geometry + Vehicle Position + Additional Info



```xml
<p1:time nTimestamp="18">

        <p1:car age="495" confidence="0" heading="0" index="0" lane="7" nTimestamp="0"
         offset="402" pathId="136" probability="40" speed="1660" vpStat="0"/>

        <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClass="2"    lastStubAtOffset="true"
        numberOfLanesInDrivingDirection="2"              numberOfLanesInoppositeDirection="0" offset="0"
        partOfCalculatedRoute="3"        pathId="0" relativProbability="100" rightOfWay="2" subPathId="136"
        turnAngle="0"    update="false">

                <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
                <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
                <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
                <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
                <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
                <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
                <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
                <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
                <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>
                <p1:profile offset="299" type="AV2_SLOPE_LIN">516</p1:profile>
                <p1:profile offset="323" type="AV2_SLOPE_LIN”>516</p1:profile>
                <p1:profile offset="353" type="AV2_SLOPE_LIN">518</p1:profile>
                <p1:profile offset="391" type="AV2_SLOPE_LIN">520</p1:profile>
                <p1:profile offset="448" type="AV2_SLOPE_LIN">520</p1:profile>
                <p1:profile offset="458" type="AV2_SLOPE_LIN">520</p1:profile>
                <p1:profile offset="548" type="AV2_SLOPE_LIN">523</p1:profile>
                <p1:profile offset="613" type="AV2_SLOPE_LIN">525</p1:profile>
                <p1:profile offset="676" type="AV2_SLOPE_LIN">525</p1:profile>
                <p1:profile offset="299" type="AV2_ROAD_CONDITION">8</p1:profile>
                <p1:profile offset="323" type="AV2_ROAD_CONDITION">8</p1:profile>
                <p1:profile offset="448" type="AV2_ROAD_CONDITION">8</p1:profile>
                <p1:profile offset="548" type="AV2_ROAD_CONDITION">8</p1:profile>
                <p1:profile offset="613" type="AV2_ROAD_CONDITION">8</p1:profile>
```

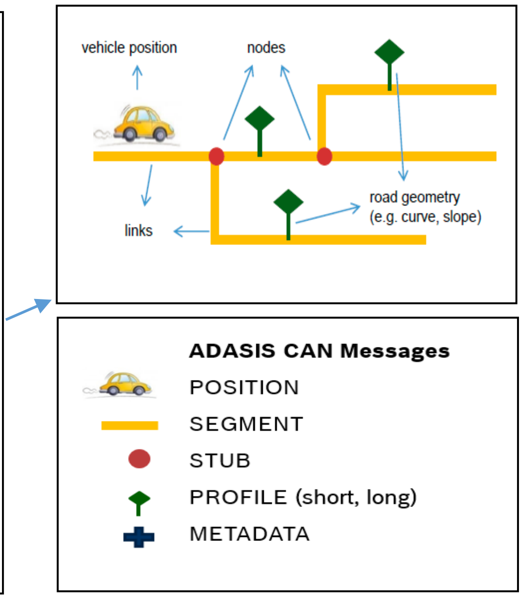


Figure 4.1: XML Map Representation

In Figure 4.1 the xml file represents the data which is stored in map database in the form of figure shown above. XML file describes the path and gives information of current car position, stub, new paths and type of road geometry.

A car when travelling has to only take into consideration the horizon preview on which it is present. That means it should delete or eliminate data that is not required any more. It can be done in the form of forgetting the attributes like path, position, offsets etc. of such location from where the car has already passed.

As the vehicle moves forward it don't need the values of such route from which it already passed by. In Figure 4.2 it can be seen that how car stores only the information of current horizon. This is very efficient method as it saves memory and as the car moves forward ADAS horizon provider can send more information related to current path at one time.

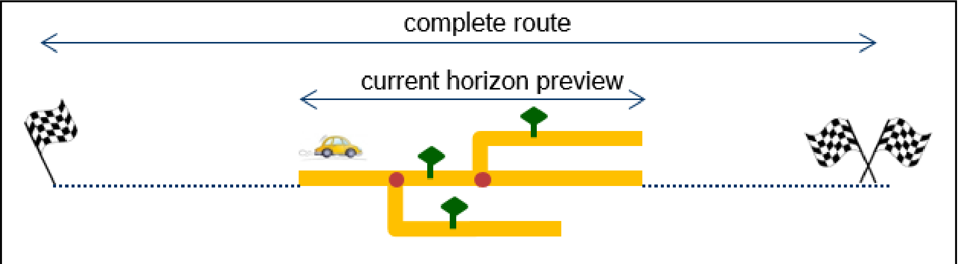
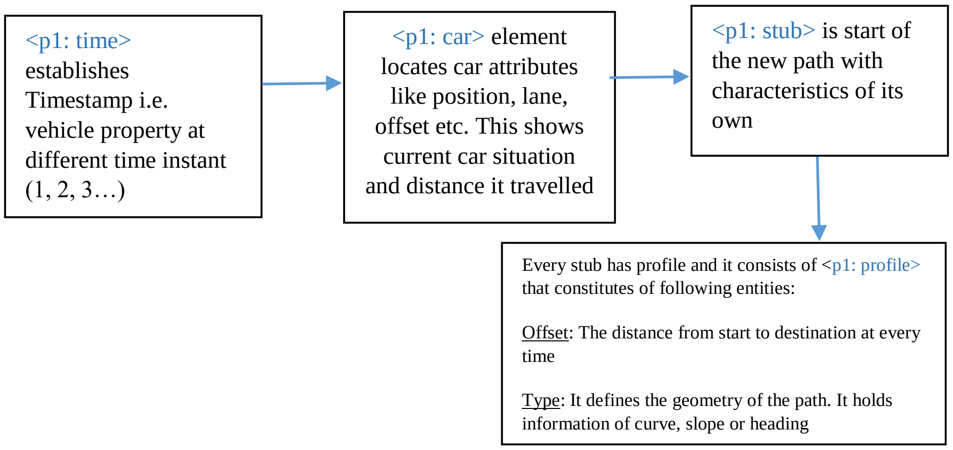


Figure 4.2: Horizon Preview

4.1.2. XML structure

Clearer version of XML file is presented in Figure 4.4. A fully developed file has all the information of path in XML format at every instant of time. XML file shown in Figure 4.4 and 4.1 is structured in following manner:



<p1: time> establishes Timestamp i.e. vehicle property at different time instant (1, 2, 3…)

<p1: car> element locates car attributes like position, lane, offset etc. This shows current car situation and distance it travelled

<p1: stub> is start of the new path with characteristics of its own

Every stub has profile and it consists of <p1: profile> that constitutes of following entities:

Offset: The distance from start to destination at every time

Type: It defines the geometry of the path. It holds information of curve, slope or heading

Figure 4.3: XML file structure

Above mentioned file structure is based on data given in Figure 4.4. This will give a proper view upon schema that xml file uses in our project.

```
# Parent Element “time”

<p1:time nTimestamp="18">

# First child Element “car”

       <p1:car age="495" confidence="0" heading="0" index="0" lane="7" nTimestamp="0"
       offset="402" pathId="136" probability="40" speed="1660" vpStat="0"/>

# Second child Element “stub”

       <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClass="2"
       lastStubAtOffset="true"      numberOfLanesInDrivingDirection="2"
       numberOfLanesInoppositeDirection="0" offset="0"    partOfCalculatedRoute="3"
       pathId="0" relativProbability="100" rightOfWay="2" subPathId="136" turnAngle="0"
       update="false">

# first sub - child Elements “profile”

              <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
              <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
              <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
              <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
              <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
              <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
              <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
              <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
              <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>
              <p1:profile offset="299" type="AV2_SLOPE_LIN">516</p1:profile>
```

Figure 4.4: XML file format

XML file format shown in Figure 4.4 is the same format used in files that are generated out of xml log dumper. These files are written out of reference reconstructor and Bosch reconstructor at the same time. These are the files which we wish to compare.

As the labels in figure suggest the order of elements, this will be same order while parsing files for xml comparison. Each parent and child element has many attributes which explains elements details in brief. These attributes are great source of information as they suggest the nature of path, car position, distance etc. [4]

4.2. University standard Open Street Map (OSM)

One of the important question which was suggested by prof. Wolfram Hardt in my concept presentation was to include the difference between university's current working standard of OSM and relevant work in xml mapping that was taking place at Bosch in my research work.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.0.2">
<bounds minlat="54.0889580" minlon="12.2487570" maxlat="54.0913900" maxlon="12.2524800"/>

<node id="298884269" lat="54.0901746" lon="12.2482632" user="SvenHRO" uid="46882"
visible="true" version="1" changeset="676636" timestamp="2008-09-21T21:37:45Z"/>

<node id="261728686" lat="54.0906309" lon="12.2441924" user="PikoWinter" uid="36744"
visible="true" version="1" changeset="323878" timestamp="2008-05-03T13:39:23Z"/>

<node id="1831881213" version="1" changeset="12370172" lat="54.0900666" lon="12.2539381"
user="lafkor" uid="75625" visible="true" timestamp="2012-07-20T09:43:19Z">
  <tag k="name" v="Neu Broderstorf"/>
  <tag k="traffic_sign" v="city_limit"/>

<relation id="56688" user="kmvar" uid="56190" visible="true" version="28" changeset="6947637"
timestamp="2011-01-12T14:23:49Z">
  <member type="node" ref="294942404" role=""/>
  ...
  <member type="node" ref="364933006" role=""/>
  <member type="way" ref="4579143" role=""/>
  ...
  <member type="node" ref="249673494" role=""/>
  <tag k="name" v="Küstenbus Linie 123"/>
  <tag k="network" v="VVW"/>
  <tag k="operator" v="Regionalverkehr Küste"/>
  <tag k="ref" v="123"/>
  <tag k="route" v="bus"/>
  <tag k="type" v="route"/>
 </relation>

</node>
</osm>
```

Figure 4.5: Sample OSM Map file [6]

Looking at the structure of OSM file in figure below we can say that there are lots of similarities in schema and how map information is attached with XSD file used in OSM.

4.2.1. Open Street Map Attributes

OSM uses structure where map information is attached in specifically in the form of nodes, way, tags and relation. It provides complete map information data with these major elements. The XSD file of OSM is semantically similar to mapping done to xml file in our company as the details are expressed in same manner.

Main elements of OSM consists of:

Node: A node represents any point in space. It is generally defined by its latitude and longitude. Each node has at least an id number and a pair of co-ordinates. Nodes can also define standalone features. Be it any shop, traffic signals or highways.

Tag: Tag conveys the meaning of element to which they are attached. Two text fields are crucial for the consideration of tags, “Key” and “Value”. ‘Key’ describes the broad class of feature and ‘Value’ details the specific feature. A most simplistic example of tag can be ‘name=Park Avenue’. This is a tag for which value field conveys name of some street.

Relation: A relation is a data structure that establishes relation between two or more data elements. E.g. a route relation forms a relation between routes like highways, tunnel routes or heavy vehicle routes. It can also be a restriction like speed restriction.

Way: Ways can represent linear features like rivers and roads. They can show limits of areas like forests or building. [7]

4.2.2. Open Street Map (OSM) vs Bosch XML Mapping

However, there are some major differences related to type of work that OSM intends and what we work on in Bosch. Some of the differences are:

OSM is mainly used for creating maps on higher scale. Whereas the xml mapping that is used in our company is limited to what vehicle can see in nearby environment and not much further.

OSM emphasizes more on providing accurate location, paths and services. While in our case we are more concerned with the path and road geometry that vehicle follows. The path from which vehicle is consider obsolete. OSM can be integrated in the work we do only when there is strong need to create full fledge mapping of source to destination.

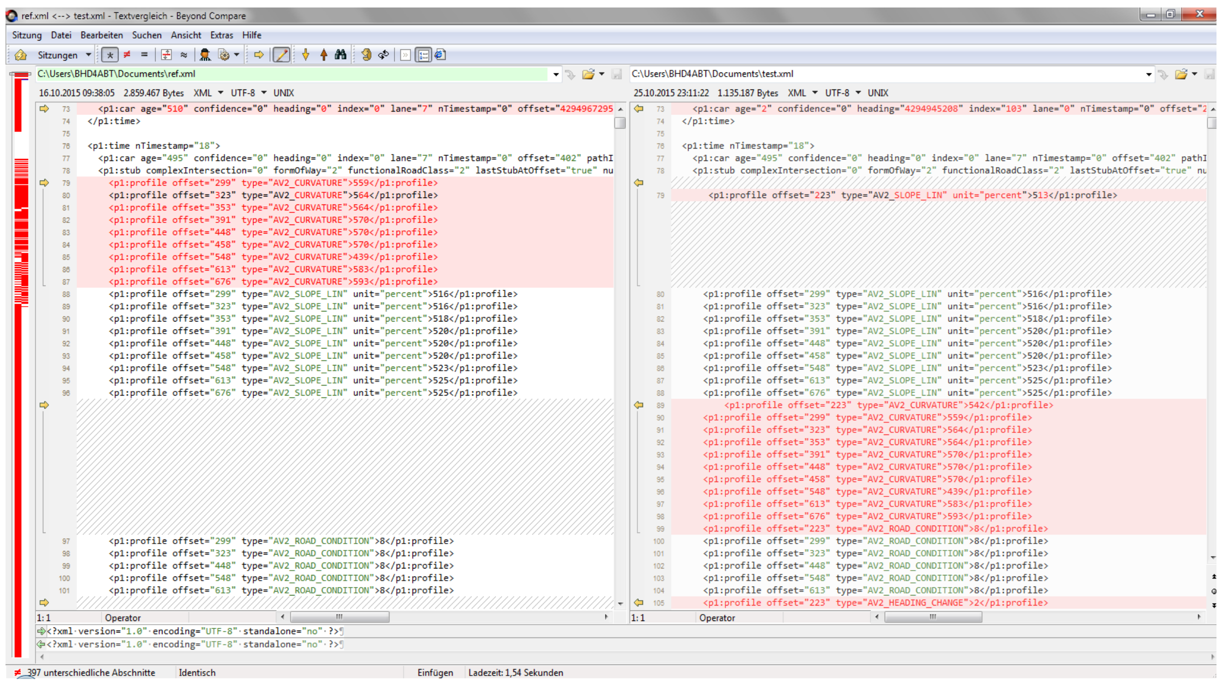
4.3. Necessity for XML Comparison Tool

XML Comparison was a needed to deal with following problems:

- To design a GUI tool for XML comparison that will highlight missing or invalid data
- Algorithm should compare despite order change
- Easy to use comparison interface
- Implementing Re-ordering of test XML file according to Reference File
- Research about different data comparison techniques
- Saving log files for later use

4.3.1. Availability:

As we discussed in state of the art that currently only a traditional tool named BeyondCompare is only available for calculating the differences. We can now look to some of the features that BeyondCompare possess:



ref.xml <--> test.xml - Textvergleich - Beyond Compare
Sitzung Datei Bearbeiten Suchen Ansicht Extras Hilfe
Sitzungen

C:\Users\BHD4ABT\Documents\ref.xml
16.10.2015 09:38:05 2.859.467 Bytes XML UTF-8 UNIX

```
    <p1:car age="510" confidence="0" heading="0" index="0" lane="7" nTimestamp="0" offset="4294967295
  </p1:time>

  <p1:time nTimestamp="18">
    <p1:car age="495" confidence="0" heading="0" index="0" lane="7" nTimestamp="0" offset="402" pathI
    <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClass="2" lastStubAtOffset="true" nu
      <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
      <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
      <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
      <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
      <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
      <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
      <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
      <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
      <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>
      <p1:profile offset="299" type="AV2_SLOPE_LIN" unit="percent">516</p1:profile>
      <p1:profile offset="323" type="AV2_SLOPE_LIN" unit="percent">516</p1:profile>
      <p1:profile offset="353" type="AV2_SLOPE_LIN" unit="percent">518</p1:profile>
      <p1:profile offset="391" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
      <p1:profile offset="448" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
      <p1:profile offset="458" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
      <p1:profile offset="548" type="AV2_SLOPE_LIN" unit="percent">523</p1:profile>
      <p1:profile offset="613" type="AV2_SLOPE_LIN" unit="percent">525</p1:profile>
      <p1:profile offset="676" type="AV2_SLOPE_LIN" unit="percent">525</p1:profile>
      <p1:profile offset="299" type="AV2_ROAD_CONDITION">8</p1:profile>
      <p1:profile offset="323" type="AV2_ROAD_CONDITION">8</p1:profile>
      <p1:profile offset="448" type="AV2_ROAD_CONDITION">8</p1:profile>
      <p1:profile offset="548" type="AV2_ROAD_CONDITION">8</p1:profile>
      <p1:profile offset="613" type="AV2_ROAD_CONDITION">8</p1:profile>
```

C:\Users\BHD4ABT\Documents\test.xml
25.10.2015 23:11:22 1.135.187 Bytes XML UTF-8 UNIX

```
    <p1:car age="2" confidence="0" heading="4294945208" index="103" lane="0" nTimestamp="0" offset="2
  </p1:time>

  <p1:time nTimestamp="18">
    <p1:car age="495" confidence="0" heading="0" index="0" lane="7" nTimestamp="0" offset="402" pathI
    <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClass="2" lastStubAtOffset="true" nu
        <p1:profile offset="223" type="AV2_SLOPE_LIN" unit="percent">513</p1:profile>
      <p1:profile offset="299" type="AV2_SLOPE_LIN" unit="percent">516</p1:profile>
      <p1:profile offset="323" type="AV2_SLOPE_LIN" unit="percent">516</p1:profile>
      <p1:profile offset="353" type="AV2_SLOPE_LIN" unit="percent">518</p1:profile>
      <p1:profile offset="391" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
      <p1:profile offset="448" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
      <p1:profile offset="458" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
      <p1:profile offset="548" type="AV2_SLOPE_LIN" unit="percent">523</p1:profile>
      <p1:profile offset="613" type="AV2_SLOPE_LIN" unit="percent">525</p1:profile>
      <p1:profile offset="676" type="AV2_SLOPE_LIN" unit="percent">525</p1:profile>
        <p1:profile offset="223" type="AV2_CURVATURE">542</p1:profile>
      <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
      <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
      <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
      <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
      <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
      <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
      <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
      <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
      <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>
      <p1:profile offset="223" type="AV2_ROAD_CONDITION">8</p1:profile>
      <p1:profile offset="299" type="AV2_ROAD_CONDITION">8</p1:profile>
      <p1:profile offset="323" type="AV2_ROAD_CONDITION">8</p1:profile>
      <p1:profile offset="448" type="AV2_ROAD_CONDITION">8</p1:profile>
      <p1:profile offset="548" type="AV2_ROAD_CONDITION">8</p1:profile>
      <p1:profile offset="613" type="AV2_ROAD_CONDITION">8</p1:profile>
      <p1:profile offset="223" type="AV2_HEADING_CHANGE">2</p1:profile>
```

1:1 Operator | 1:1 Operator

```
<?xml·version="1.0"·encoding="UTF-8"·standalone="no"·?>¶
<?xml·version="1.0"·encoding="UTF-8"·standalone="no"·?>¶
```

397 unterschiedliche Abschnitte | Identisch | Einfügen | Ladezeit: 1,54 Sekunden

Figure 4.6: BeyondCompare comparison window

The way in which differences are presented in beyond compare is shown in above figure. It aligns the data in such a manner that if data is present in one file and missing in another then it will show empty spaces in the portion of file where data is missing.

Apart from this wherever there is mismatch caused in data then it will be marked red.

4.3.2. Problems with BeyondCompare

BeyondCompare is easy to use tool and very productive. However, there are some major disadvantages like it fails to detect the order change like showed in Figure 4.6.

There can be many types of order changes like:

1). Mismatching order of "stubs" in both files

2). Order change in Timestamps

3). When data is complex there can be order change in deep children elements like "Profile"

With reference to xml file above there can be great variation in order change problems when data is complex.

Likewise there are also no options for saving data for further use. Moreover, it does a line to line comparison which is far simplistic.

4.4. Research for Implementation

Before going into implementation, a research for implementation was established. This section explains research regarding means to obtain the end goal. Brief explanations about parsers, tools and libraries is mentioned in this section.

4.4.1. XML Parsers

XMl parsing in python can be done in many ways. One of the generic way is making use of DOM (Document Object Model) and SAX (Simple API for XML) parsers. However due to availability of ElementTree library as python's inbuilt standard library. I tried using it and results generated were very appealing. [8]

The way ElementTree performs is it uses parse () function which takes filename and it parses the entire document at once. It returns an objects which are presented in Code 2. Core concepts of implementing parser is explained in section 6.1.2. Once results were confirmed from the parser a working method of how information can be extracted from the XML elements was established.

4.4.2. Working with Libraries

On further parsing some documents using above mentioned ElementTree library, I figure out that parsing was getting really slow for long size documents. This can be a greater drawback as files used for comparisons are way too complex.

Upon some research I came across some useful libraries like BeautifulSoup, minidom, cElementTree (faster version of ElementTree). Using them for results was not a problem. But minidom uses DOM parser and it is not useful if we have large number of data because it stores information in the form of node and there can be many of them in large data.

However with more amount of data there was needed something versatile and easy to use. It also should have good tool support. So while reading about ElementTree I came across LXML, it is open source library made by third party but it has full compatibility with ElementTree. Moreover, it is built on popular C libraries like libxml2 and libxslt. Putting it to test produced interesting and fast results. Accessing elements and their roots was not a problem anymore. [9]

So by this time a working concept of parser and library was found which would allow me at least finish scripting comparator.

4.4.3. User Interface

Just like scripting, a firm research was needed to know which GUI library can provide us needed functionality. Python have great support for making interactive user interfaces, but while working in corporate environment we have to take care of licenses as well.

Majority of the popular support for user interface included wxPython, PyQT, PyGtk, gnome. However except python's inbuilt library Tkinter all of the other libraries had GPL licenses. So project made by GPL compliant libraries also has to be open source which was not possible in development of my project. Tkinter is 100% open source with BSD license so user have full rights to modify, sell his/her products without even necessarily providing source code for it. [10]

Additionally, Tkinter had great features and overriding any of its inbuilt feature was very simplistic. This gave it an edge over other tools. In my work I want something which can be easily modified to user requirements and Tkinter is best library used so far.

4.4.4. Software tools used

Summing everything up for developing comparison algorithm I have made use of following tools and libraries:

<u>Integrated Development Environment (IDE)</u>: Eclipse for C/C++ developers, version: Juno

<u>Programming Language</u>: Python 2.7 [5]

<u>XML parsing Library:</u> LXML (Open source MIT license)

<u>User Interface Library:</u> Tkinter (Open source BSD license)

4.5. Planned Research Approach

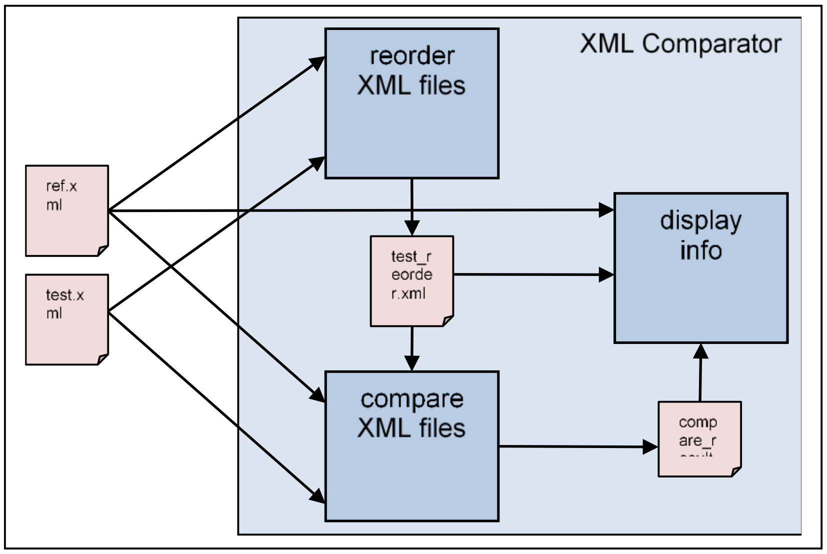


Figure 4.7: Planned approach for XML comparison tool

This block diagram presents an idea of how the tool work was planned over the duration of thesis. There are three main stages of interest – reordering, comparing and displaying. All the calculations was based on certain rules which were researched prior implementation for gaining desired output.

4.5.1. Primary Stages

The order of working was 1). Reordering 2). Comparing and 3) Displaying

1). Reordering: Reordering takes place before comparison as here we align the order of <stubs> (fig. 4.7) in "Copy.xml" or test.xml according to the one given in "Reference.xml"

2). Comparing: Here we systematically produce the results of data with respect to following objectives or rules:

<u>Rules:</u>

1. Finding missing elements
2. Establishing invalid points
3. Correct comparison despite order change (<Profile> elements)

3). Displaying: Displaying deals with how we present our output. This module should give ease of operation to the user and at the same time highlight the differences.

<u>Rules:</u>

1. Show errors under timestamps label
2. Highlight violating data points
3. Options to reordering and saving should be present

4.5.2. Saving Stage

The idea behind this stage was to research an algorithm which can introduce the feature of BeyondCompare software in which it deploys "space" to whichever points are missing from either file. This can be viewed in figure 4.6, here that large chunk of space is left by the tool to align both file format in correct manner.

Before implementing this feature a research was mandatory because a possibility has to be established weather this can be done or not. The requirement for this stage aroused when detail reading was undertaken for understanding working of BeyondCompare. Research finding for designing this algorithm is shortly summarized as:

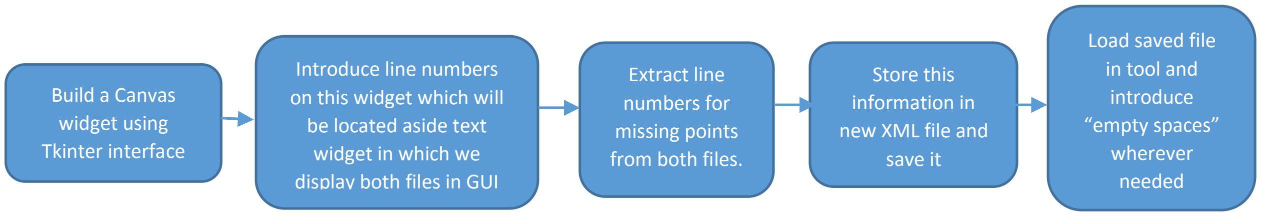

Build a Canvas widget using Tkinter interface
Introduce line numbers on this widget which will be located aside text widget in which we display both files in GUI
Extract line numbers for missing points from both files.
Store this information in new XML file and save it
Load saved file in tool and introduce "empty spaces" wherever needed

Figure 4.8: Saving results to XML file

According to research this task is possible but it may not be work as expected if the format of two files varies slightly. This approach can still work in most of the cases but needs to be backed by upgraded interface for handling and showing xml files. Moreover, in GUI that was developed has certain restriction over how functionality of widgets can be overridden.

To implement this approach we have to override the present functionality of involved Tkinter widgets by larger proportion.

Steps explained in figure above suggests closest solution possible to implement the feature that we need from BeyondCompare to our software tool. All other approaches would include changes to be done to xml file externally before loading it in XML file in order to achieve what we need.

5. Concept of CAN Stream Generation Tool

In this section description of research behind CAN stream generation tool is presented. Main aim of this task is to make a user interface for python scripts that were responsible to produce CAN data.

5.1. Script Structure

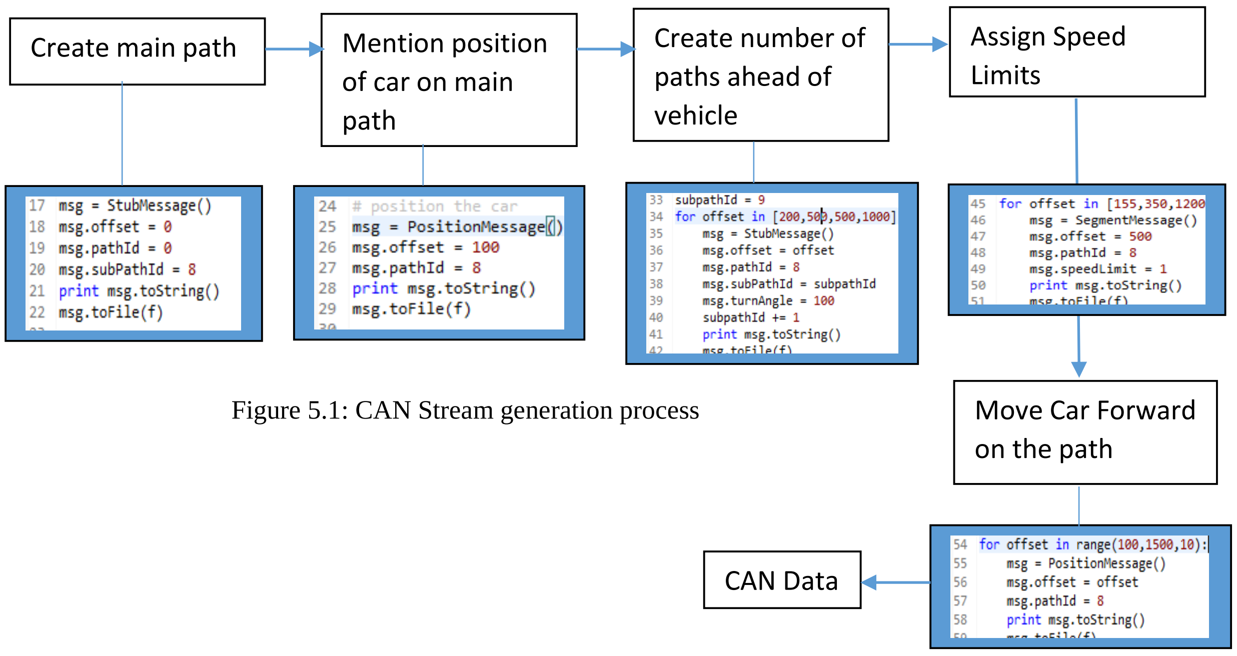
The script structure explained in figure 2.1 explains the task flow for stream generation. In this section we can understand in detail how the process takes place in detail.

5.1.1. CAN stream generation algorithm

Desired scripting of the CAN stream can vary for various purpose. For instance the flow chart that is explained below is an approach to generate CAN script data that simulates the process of giving car position, speed limits, creating paths and moving car over that path.

Since this entire test environment simulates the natural environment of moving car, CAN stream is used to model that environment.

The structure is as follows:


Create main path
Mention position of car on main path
Create number of paths ahead of vehicle
Assign Speed Limits
Move Car Forward on the path
CAN Data

```
msg = StubMessage()
msg.offset = 0
msg.pathId = 0
msg.subPathId = 8
print msg.toString()
msg.toFile(f)
```

```
# position the car
msg = PositionMessage()
msg.offset = 100
msg.pathId = 8
print msg.toString()
msg.toFile(f)
```

```
subpathId = 9
for offset in [200,500,500,1000]
    msg = StubMessage()
    msg.offset = offset
    msg.pathId = 8
    msg.subPathId = subpathId
    msg.turnAngle = 100
    subpathId += 1
    print msg.toString()
    msg.toFile(f)
```

```
for offset in [155,350,1200
    msg = SegmentMessage()
    msg.offset = 500
    msg.pathId = 8
    msg.speedLimit = 1
    print msg.toString()
    msg.toFile(f)
```

```
for offset in range(100,1500,10):
    msg = PositionMessage()
    msg.offset = offset
    msg.pathId = 8
    print msg.toString()
    msg.toFile(f)
```

Figure 5.1: CAN Stream generation process

5.1.2. Assigning Attributes to CAN data

In section 5.1 we can see that how simulation takes place of creating paths, assigning speed limits and driving car over the created path in the form of moving offset points. In above figure there are classes called like stubMessge (), positionMessage () and segmentMessage ().

These classes are imported from another file and whenever these methods are called, they define attributes for specific paths like shown in below figure.

```
subPathId = property(get_sub_path_id, set_sub_path_id, None, None)
lastStub = property(get_last_stub, set_last_stub, None, None)
turnAngle = property(get_turn_angle, set_turn_angle, None, None)
update = property(get_update, set_update, None, None)
relProbb = property(get_rel_probb, set_rel_probb, None, None)
funcRoadClass = property(get_func_road_class, set_func_road_class, None, None)
partOfCalcRoute = property(get_part_of_calc_route, set_part_of_calc_route, None, None)
complexInter = property(get_complex_inter, set_complex_inter, None, None)
formOfWay = property(get_form_of_way, set_form_of_way, None, None)
rightOfWay = property(get_right_of_way, set_right_of_way, None, None)
numOfLaneOppDir = property(get_num_of_lane_opp_dir, set_num_of_lane_opp_dir, None, None)
numOfLaneDrvDir = property(get_num_of_lane_drv_dir, set_num_of_lane_drv_dir, None, None)
retrans = property(get_retrans, set_retrans, None, None)
```

Figure 5.2: Path Attributes

Now if we recall what was discussed in section 4.1.1 and figure 4.4 regarding XML file format, all the attributes that were written in XML file were originally created from this methods listed above. CAN data that is generated after calling this classes and methods looks like the one presented in figure 2.2 in section 2.1.1.

Just like methods shown in Figure 5.2 stands for defining any path characteristics all different methods like positionMessage () and segmentMessage () are there to define vehicle's attributes. This all data is then converted into CAN stream and given as an input to the Reconstructor in next stage.

5.2. Necessity for CAN Stream Generation Tool

The main reason for developing tool was to expand its capabilities and make human interfacing less complex. Further development was needed to build user interface that would be able to take input from user in specific way and generate CAN stream just the way scripts are generating.

Automate Manual Operations

As we discussed the script structure in section 5.1, we can see that current implementation include only scripts for working. Anyone who intends to use the script has to understand the functions and accordingly input appropriate offsets inside file.

With the introduction of user interface this task can be automated. Also making sure that user has the option to input offset points which represent car as moving on path. Moreover, this tool will not be limited for testing only one type of script. Rather user can write script for any scenario and use tool to generate output. [11]

5.3. Research for Implementation

Just like XML comparator tool this section deals with research findings for confirming the feasibility of solutions. There is no difference between user interface libraries that I used previously. But concept is altogether different to implement needed features.

5.3.1. Parameters

Main parameters for building user interface for the scripts are offsets. Offsets are data points that are responsible for calling methods which can define path and vehicle attributes. These parameters may be data points which can have a specific range.

5.3.2. Input to Parameters

Tools and libraries used for this task include:

Integrated Development Environment (IDE): Eclipse for C/C++ developers, version: Juno
Programming Language: Python 2.7
User Interface Library: Tkinter (Open source BSD license)

Input to this parameters has to be given in a way that there should be a user interface like popup window that were users should be allowed to mention range or in the form of any numbers. These values will define the offset points. There should be another method implemented where after the user is done inputting range, these values can be extracted to the main window.

On main window for every selected parameters some sliders should be displayed on main window which gives users to pass any offset points from the mentioned range. Since Tkinter library has implementation of inbuilt sliders. Selecting various values over the given range is possible.

Apart from sliders there also should be option of inputting such values which are not range dependent. So this can be attained by the use of Label which is also an option available in Tkinter library.

5.3.3. Saving output to file

After all the values are loaded into user interface, there should also be option where all the loaded values are stored in another file according to the titles. This file should be then imported in another file so that all the values which were stored in this file are available for scripts that contain program to generate CAN data. Whole structure can be viewed in following manner:

6. Realization and Implementation

This section includes implementation for all the research work that was undertaken over the duration of thesis. Unlike section 4 and section 5 where I divided the research and concept work for both tools separately, in this section explanations is given to all the implementation in one section with individual sub sections.

The basic organization of this section is done in a manner where implementation for XML Comparator tool stages is described first and then followed with implementation of CAN Script Generator. All the research and implementation for this tool was prepared from scratch.

Before starting approaches it will be helpful to go through the file structure of XML comparison. XML comparator comprises of three python script files and four XML files. Throughout implementation all the logic implemented in these scripts and files will be explained.

Reference.xml	Contains the data generated by reference reconstructor
Copy.xml	Contains the data generated by Bosch reconstructor
Reordered_copy.xml	New ordered format of “Copy.xml” that has order according to “Reference.xml”
ReorderFile.py	Script to generate “Reordered_copy.xml”
Stub_Structure.py	Script to compare xml files and contains user interface building
Saving_file.py	Script to save compared results in saveddata.xml
Saveddata.xml	File that contains all the compared results in form of index numbers and other formatting info.

Table 4: XML Comparator (File Structure)

6.1. Reordering Approach for XML Comparison

Reordering has to be done before comparison and the reason is presented below:

Reference.xml

```
<time “nTimestamp” = 18>
<car></car>
<stub1></stub1>
<stub2></stub2>
<stub3></stub3>
</time>

<time “nTimestamp” = 19>
<car></car>
<stub1></stub1>
<stub2></stub2>
<stub3></stub3>
<stub4></stub4>
</time>
```

“Copy.xml”

```
<time “nTimestamp” = 18>
<car></car>
<stub1></stub1>
<stub2></stub3>
<stub3></stub2>
</time>

<time “nTimestamp” = 19>
<car></car>
<stub1></stub1>
<stub2></stub3>
<stub4></stub4>
<stub3></stub2>
</time>
```

New_”Copy.xml” (reordered)

```
<time “nTimestamp” = 18>
<car></car>
<stub1></stub1>
<stub2></stub2>
<stub3></stub3>
</time>

<time “nTimestamp” = 19>
<car></car>
<stub1></stub1>
<stub2></stub2>
<stub3></stub3>
<stub4></stub4>
</time>
```

Figure 6.1: mismatching file contents

In figure 6.1 both files represents an easy to understand structure that defines xml file. It can be seen that order of stubs in “Copy.xml” file is completely changed. This might have happened when the file was generated out of the reconstructor. Using this file directly with any comparison tool will yield error in difference because the order of children of all xml files must be same.

This is where reordering algorithm comes handy. A simple working is presented below:

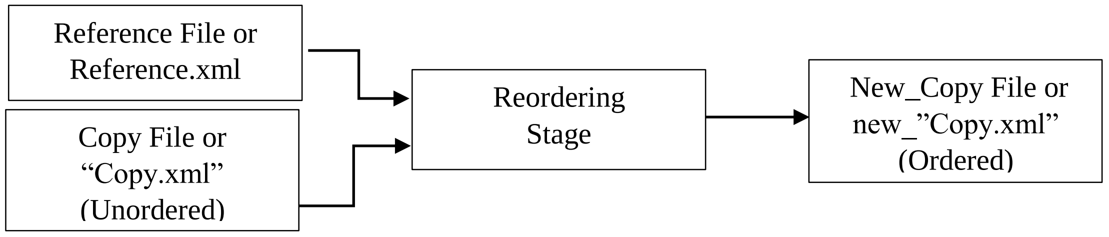

Reference File or Reference.xml
Copy File or “Copy.xml” (Unordered)
Reordering Stage
New_Copy File or new_”Copy.xml” (Ordered)

Figure 6.2: Reordering Process

6.1.1. Program Structure

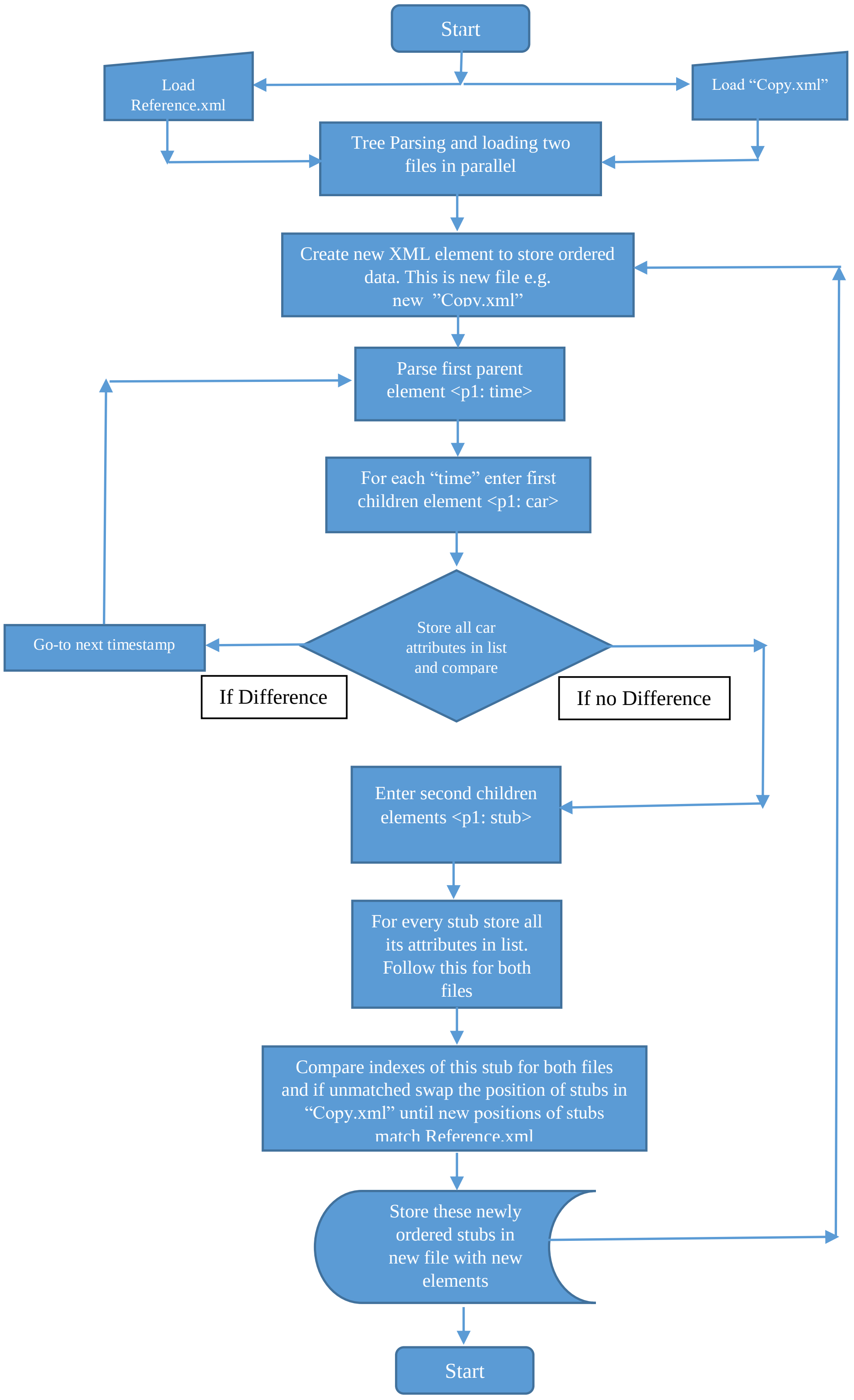

Start
Load Reference.xml
Load "Copy.xml"
Tree Parsing and loading two files in parallel
Create new XML element to store ordered data. This is new file e.g. new "Copy.xml"
Parse first parent element <p1: time>
For each "time" enter first children element <p1: car>
Store all car attributes in list and compare
Go-to next timestamp
If Difference
If no Difference
Enter second children elements <p1: stub>
For every stub store all its attributes in list. Follow this for both files
Compare indexes of this stub for both files and if unmatched swap the position of stubs in "Copy.xml" until new positions of stubs match Reference.xml
Store these newly ordered stubs in new file with new elements
Start

Figure 6.3: Reordering Approach

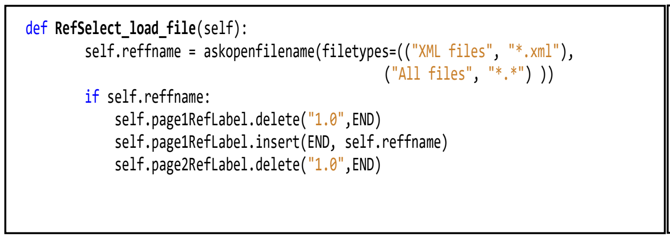
6.1.2. Logic Implementation

Flowchart shown in Figure 6.3 explains how reordering is performed. First few stages of parsing files are same as shown in Figure 4.5 of comparison approach. However moving further we store the new order of stubs which were taken from “Copy.xml” into new file with matching XML schema. Below is the working implementation of script for all stages.

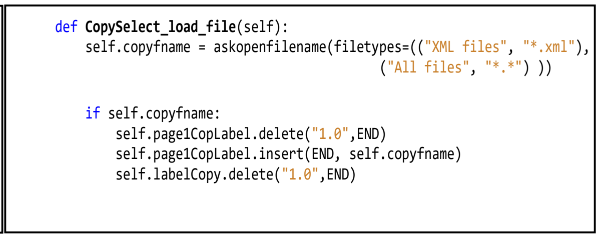
1). File loading:

This is first stage where tool asks for two files from user. These two files are generated out of XML log dumper. One file is “Copy.xml” which represents data in Bosch Reconstructor and other is “Reference.xml” which represents data from ADASIS Reconstructor.

This can be observed in Figure 2 where both dumpers are individually connected to the both reconstructors. Our main aim here is to change the order of “Copy.xml” and to make new file named as “Reordered_copy.xml” that matches the order of “Reference.xml”. This order change is necessary because most of the times XML dumper does not maintains the correct order while writing “Copy.xml”.



```
def RefSelect_load_file(self):
        self.reffname = askopenfilename(filetypes=(("XML files", "*.xml"),
                                           ("All files", "*.*") ))
        if self.reffname:
            self.page1RefLabel.delete("1.0",END)
            self.page1RefLabel.insert(END, self.reffname)
            self.page2RefLabel.delete("1.0",END)
```



```
def CopySelect_load_file(self):
    self.copyfname = askopenfilename(filetypes=(("XML files", "*.xml"),
                                        ("All files", "*.*") ))

    if self.copyfname:
        self.page1CopLabel.delete("1.0",END)
        self.page1CopLabel.insert(END, self.copyfname)
        self.labelCopy.delete("1.0",END)
```

Code 1: Load “Reference.xml” and “Copy.xml” (stub_structure.py)

2). Parsing XML Files

For parsing XML files a specific use of python based LXML library which is used for xml parsing. There are many libraries that are available for parsing xml files like python’s state of the art ElementTree, Minidom, cElementTree, BeautifulSoup etc.

I chose to use LXML because it’s built on the C libraries libxml2 and libxslt. This gives it edge over speed and memory performance while keeping format simple to use while programming in python. It is easy to install and its ability to parse xml documents in Tree format is exceptionally fast and simple to understand. Due to its simplicity it becomes easy to modify or extract the deep located elements in xml format.

We will see how below given xml file resembles to data when parsed:

```
<p1:time nTimestamp="18">

        <p1:car age="495" confidence="0" heading="0" index="0" lane="7" nTimestamp="0"
                offset="402" pathId="136" probability="40" speed="1660" vpStat="0"/>

        <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClass="2"
        lastStubAtOffset="true"      numberOfLanesInDrivingDirection="2"
        numberOfLanesInoppositeDirection="0" offset="0"    partOfCalculatedRoute="3"
        pathId="0" relativProbability="100" rightOfWay="2" subPathId="136"       turnAngle="0"
        update="false">

               <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
               <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
               <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
               <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
               <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
               <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
               <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
               <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
               <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>
               <p1:profile offset="299" type="AV2_SLOPE_LIN">516</p1:profile>
               <p1:profile offset="323" type="AV2_SLOPE_LIN">516</p1:profile>
               <p1:profile offset="353" type="AV2_SLOPE_LIN">518</p1:profile>
               <p1:profile offset="391" type="AV2_SLOPE_LIN">520</p1:profile>
               <p1:profile offset="448" type="AV2_SLOPE_LIN">520</p1:profile>
               <p1:profile offset="458" type="AV2_SLOPE_LIN">520</p1:profile>
               <p1:profile offset="548" type="AV2_SLOPE_LIN">523</p1:profile>
               <p1:profile offset="613" type="AV2_SLOPE_LIN">525</p1:profile>
               <p1:profile offset="676" type="AV2_SLOPE_LIN">525</p1:profile>
               <p1:profile offset="299" type="AV2_ROAD_CONDITION">8</p1:profile>
               <p1:profile offset="323" type="AV2_ROAD_CONDITION">8</p1:profile>
               <p1:profile offset="448" type="AV2_ROAD_CONDITION">8</p1:profile>
               <p1:profile offset="548" type="AV2_ROAD_CONDITION">8</p1:profile>
               <p1:profile offset="613" type="AV2_ROAD_CONDITION">8</p1:profile>
```

```
<Element {http://www.example.org/eHorizon}time at 0x33910a8>
<Element {http://www.example.org/eHorizon}car at 0xc7acaa8>
<Element {http://www.example.org/eHorizon}stub at 0xbdccdc8>
<Element {http://www.example.org/eHorizon}profile at 0xbde3cd8>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3d00>,
<Element {http://www.example.org/eHorizon}profile at 0xbde34e0>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3530>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3cb0>,
<Element {http://www.example.org/eHorizon}profile at 0xbde37d8>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3738>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3788>,
<Element {http://www.example.org/eHorizon}profile at 0xbde37b0>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3940>
<Element {http://www.example.org/eHorizon}profile at 0xbde3b48>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3c60>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3b20>,
<Element {http://www.example.org/eHorizon}profile at 0xbde39b8>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3c38>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3a58>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3c88>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3d28>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3c10>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3be8>,
<Element {http://www.example.org/eHorizon}profile at 0xbde3d78>,
```

Code 2: XML parsing

Here the structure of parsing and xml file is consisted of following order:

Elements	Attributes
Parent Element – time	Attribute – nTimestamp 18, 19, 20…
Children Element 1 – Car	Attribute – age, confidence, heading…
Children Element 2 – Stub	Attribute – offset, complex Intersection…
Sub children elements of Stub – Profile	Attribute – Offset, Type, Unit…

Table 5: XML Elements and Attributes

As shown in Table 5 this structure is also called Tree parsing because in any given xml file, the parsing forms a sequential parent to children relationship that works like tree.

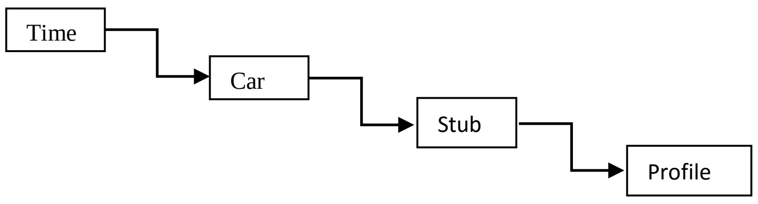


Figure 6.4: Tree Parsing

After user opens the file in interface, as we seen in Figure 6.3 that variable “self.reffname” extracts the “Reference.xml” file from given location. Similarly, “Copy.xml” file is also stored in variable “self.copyfname”.

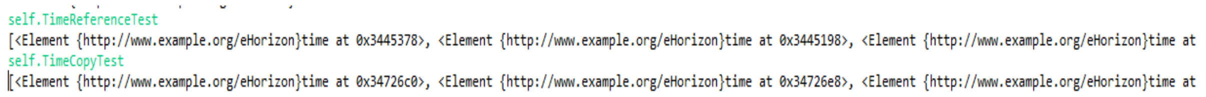
In order to parse these two xml files using LXML library following steps are implemented for both files – “Reference.xml” and “Copy.xml”

```
recovering_parser = etree.XMLParser(recover=True)
AdasReference = etree.parse(self.reffname, parser=recovering_parser).getroot()
            ToStringAdasReference = etree.tostring(AdasReference)
            AdasReferenceString = etree.fromstring(ToStringAdasReference,
            parser=recovering_parser)
            self.TimeReferenceTest =
AdasReferenceString.findall("{http://www.example.org/eHorizon}time")
```

```
recovering_parser = etree.XMLParser(recover=True)
AdasCopy = etree.parse(self.copyfname, parser=recovering_parser).getroot()
            ToStringAdasCopy = etree.tostring(AdasCopy)
            AdasCopyString = etree.fromstring(ToStringAdasCopy,
            parser=recovering_parser)
            self.TimeCopyTest = AdasCopyString.findall("{http://www.example.org/eHorizon}time")
```

Code 3: parsing and storing data of “Reference.xml” and “Copy.xml” (stub_structure.py)

As we see in above figure newly formed variables “self.TimeReferenceTest” and “self.TimeCopyTest” are having the values of all time events. A quick debugging in Eclipse shows the value of this data which is shown below.



```
self.TimeReferenceTest
[<Element {http://www.example.org/eHorizon}time at 0x3445378>, <Element {http://www.example.org/eHorizon}time at 0x3445198>, <Element {http://www.example.org/eHorizon}time at
self.TimeCopyTest
[<Element {http://www.example.org/eHorizon}time at 0x34726c0>, <Element {http://www.example.org/eHorizon}time at 0x34726e8>, <Element {http://www.example.org/eHorizon}time at
```

Code 4: Timestamps after parsing xml file

These are nothing but all the parent elements with format *<p1: time “nTimestamp”=‘1, 2, 3...’>* in xml file. As we discussed earlier there can be more than 200 to 300 timestamps with “Time” as parent element and “car”, “stub” and “profile” as children elements.

In order to compare two file at same instant and timestamp, it is mandatory that two files has to run in parallel. This is done using python’s “zip” function. The main advantage of this method is one can compare every element and attributes within it at same instant. Though making sure that there is no simple line to line comparison being done.

The format of timestamps in both files are specific and in correct order. Order change may only happen inside children elements which is taken care of while comparing as we go forward.

In implementation parsed data of two files run in parallel by following the tree structure explained in Figure 6.4

```
for i,j in zip(self.TimeReferenceTest,self.TimeCopyTest):
            self.stubFrame_a=0
            self.stubFrame_b=0

         for car1,car2 in zip(i.getchildren(),j.getchildren()):

              # some data


for k,l in zip(i.findall
              ("{http://www.example.org/eHorizon}stub"),j.findall("{http://www.exam-
              ple.org/eHorizon}stub")):

      self.Reflinenumber = 0
      self.Coplinenumber = 0
      self.stubIndex = self.stubIndex+1
      MakeFrame = 0
```

Code 5: Looping elements in parallel (stub_structure.py)

3). Create new XML element in new file:

The significance of this stage is that reordered data set is not overwritten on "Copy.xml" itself. The reason for this is it increases the complexity of the data which is showed and also resulting file may or may not be accurate. This process corrects the order of "Copy.xml" and this corrections are written in new file – "Reordered_copy.xml".

Making a new XML file and creating new elements inside just like the way "Reference.xml" is created will give us with replica of new copy file which has stubs according to "Reference.xml". New element is written on xml file in following manner:

Function to make new file named

```
def make_reorder_file(self):
        self.saveFile="C:/Users/BHD4ABT/Desktop/ReorderData.xml"
        status.config(text="Reordering initiated. Please wait. Writing files")
        Reordering = Reorder()
        Reordering.reOrder(self.reffname,self.copyfname, self.saveFile,
                                                   pbar_f, bottomframe5)
        pbar_f["value"] = 0
        status.config(text="Reordering done. File saved in desktop")
```

Creating new XML element in newly created file:

```
root = etree.Element('{http://www.example.org/eHorizon}adasis_eh',nsmap={'p1':'http://www.exam-
ple.org/eHorizon',
'xsi':'http://www.example.org/eHorizon'})


root.text= '\n\n\n'
```

Code 6: Making new XML element (stub_structure.py)

4). Storing car attributes:

This stage is similar to the one described in comparison approach but only difference here algorithm cannot proceed to next stub if we spot any difference in car attributes. This is because it is assumed that if car attributes are not completely matching then the algorithm should not proceed to other children elements and it should transit to next timestamp.

```
for carRef,carCopy in zip(i.getchildren(), j.getchildren()):

     # carCopy.set('age', 'Updated')
       car_reference_attributes = [i.get('nTimestamp'), carRef.attrib.get('age'),
       carRef.attrib.get('confidence'), carRef.attrib.get('heading'), carRef.attrib.get('in-
       dex'), carRef.attrib.get('lane'), carRef.attrib.get('offset'), carRef.attrib.get('path-
       Id'), carRef.attrib.get('probablity'), carRef.attrib.get('speed'), carRef.at-
       trib.get('vpStat')]

       car_copy_attributes = [j.get('nTimestamp'), carCopy.attrib.get('age'), carCopy.at-
       trib.get('confidence'), carCopy.attrib.get('heading'), carCopy.attrib.get('index'),
       carCopy.attrib.get('lane'), carCopy.attrib.get('offset'), carCopy.attrib.get('pathId'),
       carCopy.attrib.get('probablity'), carCopy.attrib.get('speed'), carCopy.at-
       trib.get('vpStat')]

      if car_reference_attributes == car_copy_attributes:
              reorderStub = j.findall("{http://www.example.org/eHorizon}stub")
              for k,l in zip(i.findall("{http://www.example.org/eHorizon}stub"), reorderStub):
```

Code 7: Storing car attributes in reordering (ReorderFile.py)

5). Comparing and Reordering stubs:

Here variable “ref_list” and “cop_list” are list that stores the attributes and current position of stubs from both “Reference.xml” and “Copy.xml”. In below mentioned method for every stub in ref_list, it is compared with all the stubs of cop_list. If the index of both stubs are same that means that the position of both stubs is same and hence we should not do anything and break the loop.

However if the indexes are not same then the position of stub in cop_list has to be swapped so that its index match the stub in ref_list. So after the position is matched we should head on to next stub and so on.

```
ref_list.append(reference_stub_attributes)
cop_list.append(copy_stub_attributes)

    if ref_list and cop_list:

       if ref_list [0][1]!=cop_list[0][1]:
             break
       else:
           for stub1 in ref_list:
               for stub2 in cop_list:
                   if stub1[1] == stub2[1]:
                        stub1_index = ref_list.index(stub1)
                        stub2_index = cop_list.index(stub2)
                        if stub1_index==stub2_index:
                           break
                        else:
                            temp = cop_list[stub2_index]
                            cop_list[stub2_index] = cop_list[stub1_index]
                            cop_list[stub1_index] = temp
```

Code 8: Comparing Indexes (ReorderFile.py)

After swapping the list of stubs has to be written back to the file. These stubs are now in correct order and they are written back to file ““Reordered_copy.xml”” which we discussed in 3rd point from above. Code for writing is given below where “saveFile” variable has location of “Reordered_copy.xml”

```
root.append(j)
etree.ElementTree(root).write(saveFile)
```

Code 9: Write ordered stubs to XML file

6.2. Comparison approach for XML Comparator

For XML comparator I have discussed my implementation regarding the approach that I discussed in section 4.5.

6.2.1. Program Structure

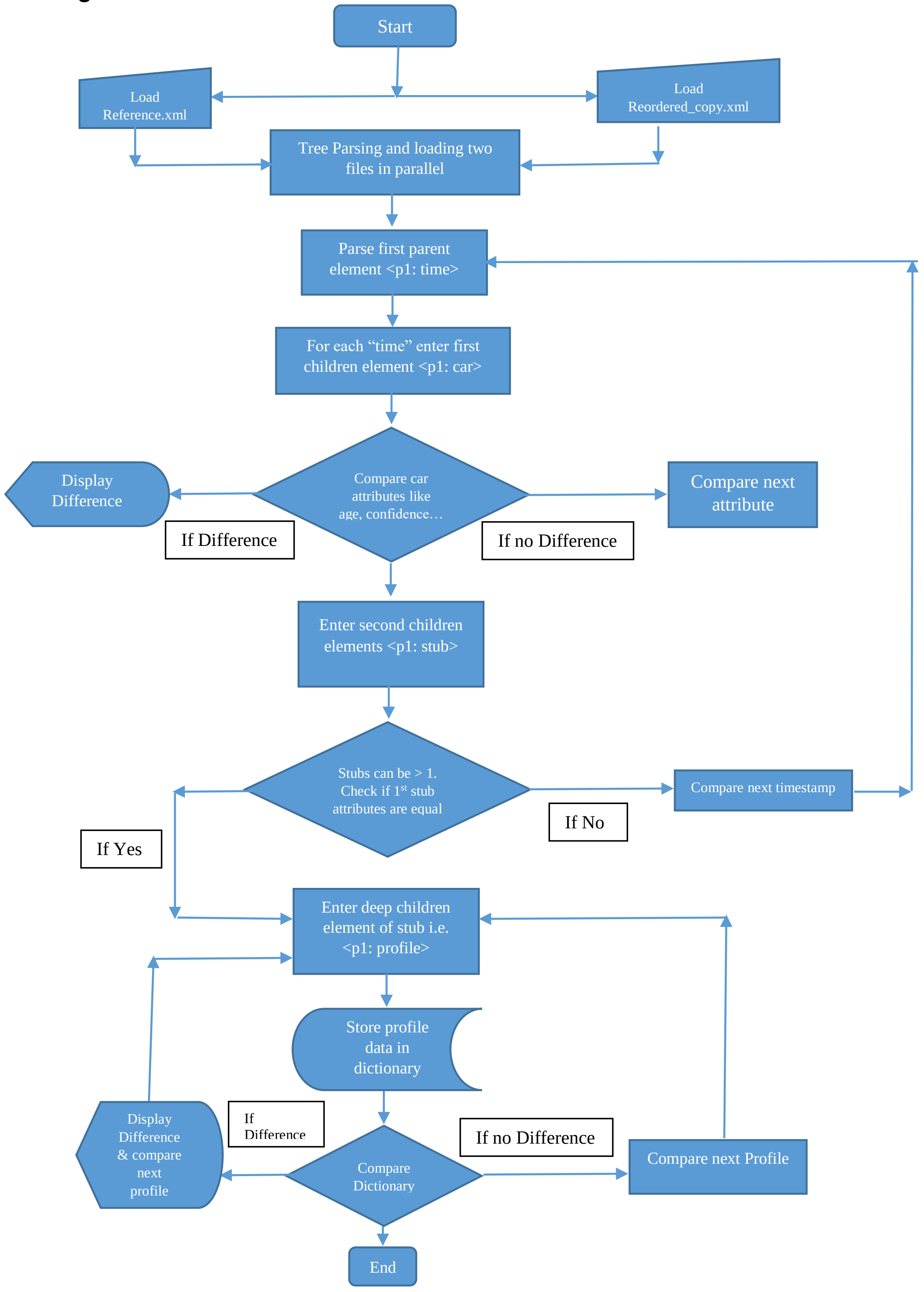

Start
Load Reference.xml
Load Reordered_copy.xml
Tree Parsing and loading two files in parallel
Parse first parent element <p1: time>
For each "time" enter first children element <p1: car>
Compare car attributes like age, confidence…
Display Difference
If Difference
If no Difference
Compare next attribute
Enter second children elements <p1: stub>
Stubs can be > 1. Check if 1st stub attributes are equal
If Yes
If No
Compare next timestamp
Enter deep children element of stub i.e. <p1: profile>
Store profile data in dictionary
Display Difference & compare next profile
If Difference
If no Difference
Compare Dictionary
Compare next Profile
End

Figure 6.5: Comparison Approach (XML Comparator Tool)

6.2.2. Logic Implementation

Flowchart in Figure 6.5 resembles to the comparison of data shown in XML file of figure 4.4. The programmatic structure in context of research approach is described. The steps shown in flowchart is implementation of research algorithm. In later part of this implementation a user interface is displayed that would present the differences and highlight it for users.

Here brief explanation of individual stages is discussed. It comprehends the details about research in every phase of my implementation.

1). File Loading and Parsing:

This two methods are same as explained in section 6.1.2 in points 1 and 2 respectively. Since comparison also needs to have input from two files, entire process of loading files in reordering approach discussed in section 4.3.2 follows same here. For parsing also same method is used.

Our main aim here is to find difference between two files and try to estimate the data irrelevancy of “Reordered_copy.xml” from “Reference.xml”. Our main aim as discussed earlier is to check the compliancy of Bosch Reconstructor with ADASIS reconstructor.

2). Comparing Car Attributes

As discussed above in parsing files section every “time” element is parent element and each of “time” element has children elements like car and stub. So according to approach discussed above we have to compare every individual attributes of child starting from “car”.

```
for car1,car2 in zip(i.getchildren(),j.getchildren()):
     carFlag = False
     self.stubIndex = 0

     if car1.attrib.get('age')!= car2.attrib.get('age'):
               if self.t and self.var==[self.t, int(i.get('nTimestamp'))]:
                          string = 'Age Difference'
```

Code 10: Comparing Car Attribute (stub_structure.py)

One simple example in Code 10 shows how every individual attributes of car element is compared. “car1” here represents child element from “Reference.xml”, “car2” represents child element from “Reordered_copy.xml” file and “age” is common attribute in both file at given timestamp. Since the loop is run in parallel it is possible to compare attributes of both child elements at one timestamp.

Above Figure Code 10 only presents the difference checking code of one attribute i.e. “Age”, but in main program it compares all attributes like “Age”, “Confidence”, “PathId”, and etc. which are located in the file.

In case of any difference the value has to be written as output. Currently implemented script generates output on user interface that was designed and will discuss it in brief ahead.

3). Comparing Stub Attributes

As we can observe from xml file that “stubs” are always 2nd child element after “car”. Once car attributes are compared and output is displayed, we compare stub. As the comparison is done in parallel loops as shown in Code 10, first all stubs are recorded at any given timestamps.

Before we move further a check is done which assures that order of first stub is always correct. It’s shown below:

```
for k,l in zip
       (i.findall("{http://www.example.org/eHorizon}stub"),j.findall("{http://www.exam-
       ple.org/eHorizon}stub")):

                self.Reflinenumber = 0
                self.Coplinenumber = 0
                self.stubIndex = self.stubIndex+1
                MakeFrame = 0

       if carFlag==False:
              if self.Ref.get(int(i.get('nTimestamp')))[0] ==
              self.Copy.get(int(j.get('nTimestamp')))[0]:

                       for m in k.getchildren():
```

Code 11: Checking 1st stub order (stub_structure.py)

In above figure “k” stands for stub from “Reference.xml” file and ‘l’ stands for stub from “Reordered_copy.xml”. As this loop runs in parallel, we compare the attributes of stub stored in dictionary “self.Ref” for “Reference.xml” with attributes stored in stub of “Reordered_copy.xml” declared by variable “self.Copy” in Figure 4.12 above.

Before proceeding for comparing further a check is made to confirm that attributes of first stub of any timestamp in “Copy.xml” is matching the attributes of same stub at given timestamp in “Reference.xml”.

4). Comparing Profile elements inside stubs

Every stub has long list of profile elements which hold an important details about road geometry. This elements characterize different stubs. Their types are in the form of "AV2_CURVATURE","AV2_SLOPE_LIN","AV2_HEADING_CHANGE","AV2_HEADING_CHANGE", "AV2_ROAD_CONDITION"

A specific approach is derived in following manner to compare individual profile elements.

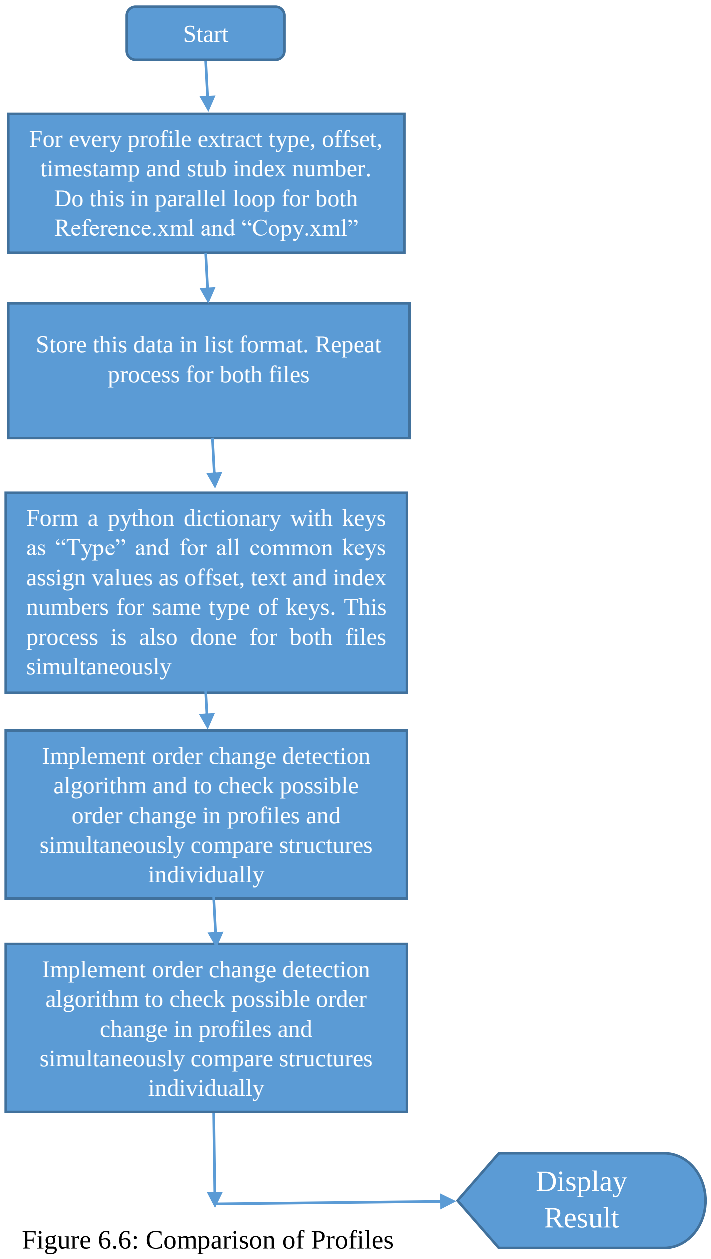

Start
For every profile extract type, offset, timestamp and stub index number. Do this in parallel loop for both Reference.xml and "Copy.xml"
Store this data in list format. Repeat process for both files
Form a python dictionary with keys as "Type" and for all common keys assign values as offset, text and index numbers for same type of keys. This process is also done for both files simultaneously
Implement order change detection algorithm and to check possible order change in profiles and simultaneously compare structures individually
Implement order change detection algorithm to check possible order change in profiles and simultaneously compare structures individually
Display Result

Figure 6.6: Comparison of Profiles

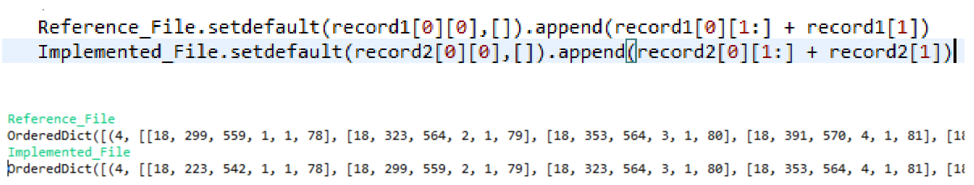
Step 1: Extract Data:

```
for m in k.getchildren():
    self.Reflinenumber=self.Reflinenumber+1
    record1 = [[m.get('type'), int(j.get('nTimestamp')),
               int(m.get('offset')), int(m.text)],[self.Reflinenumber,self.stubIndex,
               m.sourceline]]

    if record1[0][0]=='AV2_EFF_SPEED_LIMIT':
       record1[0][0]=1
    elif record1[0][0]=='AV2_ROAD_CONDITION':
       record1[0][0]=2
    elif record1[0][0]=='AV2_SLOPE_LIN':
       record1[0][0]=3
    elif record1[0][0]=='AV2_CURVATURE':
       record1[0][0]=4
    elif record1[0][0]=='AV2_HEADING_CHANGE':
       record1[0][0]=5


for n in l.getchildren():
    self.Coplinenumber=self.Coplinenumber+1
    record2 = [[n.get('type'), int(j.get('nTimestamp')),
               int(n.get('offset')), int(n.text)],[self.Coplinenumber,self.stubIndex,
               n.sourceline]]

   if record2[0][0]=='AV2_EFF_SPEED_LIMIT':
      record2[0][0]=1
   elif record2[0][0]=='AV2_ROAD_CONDITION':
      record2[0][0]=2
   elif record2[0][0]=='AV2_SLOPE_LIN':
      record2[0][0]=3
   elif record2[0][0]=='AV2_CURVATURE':
      record2[0][0]=4
   elif record2[0][0]=='AV2_HEADING_CHANGE':
      record2[0][0]=5
```

Code 12: storing profile data (stub_structure.py)

In the figure above we can observe how profile attributes are stored in list called "record1" for "Reference.xml" and "record2" for "Copy.xml". This storing of attributes is designed to perform in parallel for both loops.

Step 2: Forming a Python dictionary



```
Reference_File.setdefault(record1[0][0],[]).append(record1[0][1:] + record1[1])
Implemented_File.setdefault(record2[0][0],[]).append(record2[0][1:] + record2[1])
```

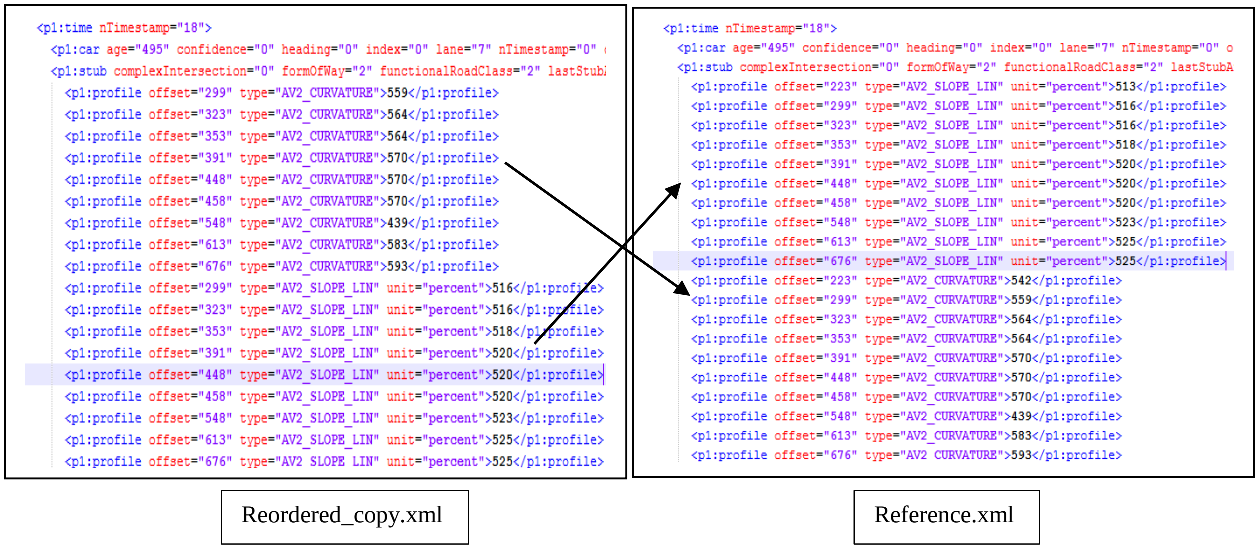
```
Reference_File
OrderedDict([(4, [[18, 299, 559, 1, 1, 78], [18, 323, 564, 2, 1, 79], [18, 353, 564, 3, 1, 80], [18, 391, 570, 4, 1, 81], [18
Implemented_File
OrderedDict([(4, [[18, 223, 542, 1, 1, 78], [18, 299, 559, 2, 1, 79], [18, 323, 564, 3, 1, 80], [18, 353, 564, 4, 1, 81], [18
```

Code 13: Storing data in Dictionary (stub_structure.py)

In Code 12, first two lines shows how data is stored in dictionary format. The last two lines are the output of debugger showing that "4" is the key which resembles to type "AV2_CURVATURE" from Code 12. The values which are stored inside keys are offsets, index number, text values etc. "Reference_File" dictionary belongs to "Reference.xml" and "Implemented_File" dictionary variable belongs to "Reordered_copy.xml".

Step 3: Order Change Detection

Order change detection has to be used to detect difference between following set of data:



```
<p1:time nTimestamp="18">
  <p1:car age="495" confidence="0" heading="0" index="0" lane="7" nTimestamp="0"
  <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClass="2" lastStub
    <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
    <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
    <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
    <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>
    <p1:profile offset="299" type="AV2_SLOPE_LIN" unit="percent">516</p1:profile>
    <p1:profile offset="323" type="AV2_SLOPE_LIN" unit="percent">516</p1:profile>
    <p1:profile offset="353" type="AV2_SLOPE_LIN" unit="percent">518</p1:profile>
    <p1:profile offset="391" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
    <p1:profile offset="448" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
    <p1:profile offset="458" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
    <p1:profile offset="548" type="AV2_SLOPE_LIN" unit="percent">523</p1:profile>
    <p1:profile offset="613" type="AV2_SLOPE_LIN" unit="percent">525</p1:profile>
    <p1:profile offset="676" type="AV2_SLOPE_LIN" unit="percent">525</p1:profile>
```

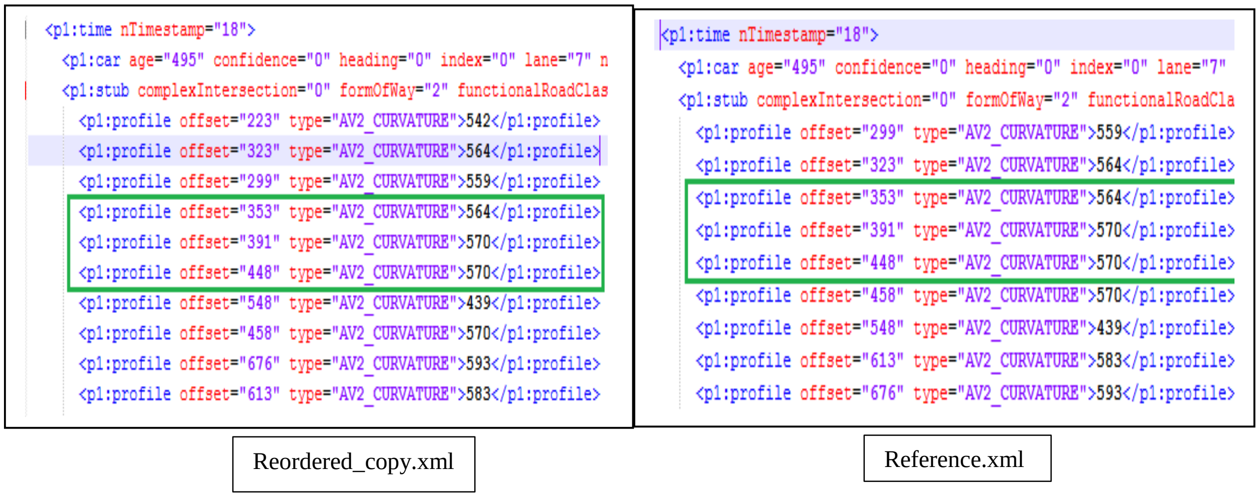
Reordered_copy.xml

```
<p1:time nTimestamp="18">
  <p1:car age="495" confidence="0" heading="0" index="0" lane="7" nTimestamp="0"
  <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClass="2" lastStubA
    <p1:profile offset="223" type="AV2_SLOPE_LIN" unit="percent">513</p1:profile>
    <p1:profile offset="299" type="AV2_SLOPE_LIN" unit="percent">516</p1:profile>
    <p1:profile offset="323" type="AV2_SLOPE_LIN" unit="percent">516</p1:profile>
    <p1:profile offset="353" type="AV2_SLOPE_LIN" unit="percent">518</p1:profile>
    <p1:profile offset="391" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
    <p1:profile offset="448" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
    <p1:profile offset="458" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
    <p1:profile offset="548" type="AV2_SLOPE_LIN" unit="percent">523</p1:profile>
    <p1:profile offset="613" type="AV2_SLOPE_LIN" unit="percent">525</p1:profile>
    <p1:profile offset="676" type="AV2_SLOPE_LIN" unit="percent">525</p1:profile>
    <p1:profile offset="223" type="AV2_CURVATURE">542</p1:profile>
    <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
    <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
    <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
    <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>
```

Reference.xml

Figure 6.7: First Objective – detect order Change in different “Type” attribute

Here in above mentioned first scenario any generic tool would list out difference in the XML files. But realistically this is not a difference. There is only an order change that has taken place within profile elements. This may have been caused by reconstructor or xml dumper but it’s not a difference.



```
<p1:time nTimestamp="18">
  <p1:car age="495" confidence="0" heading="0" index="0" lane="7" n
  <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClas
    <p1:profile offset="223" type="AV2_CURVATURE">542</p1:profile>
    <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
    <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
    <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>
    <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
```

Reordered_copy.xml

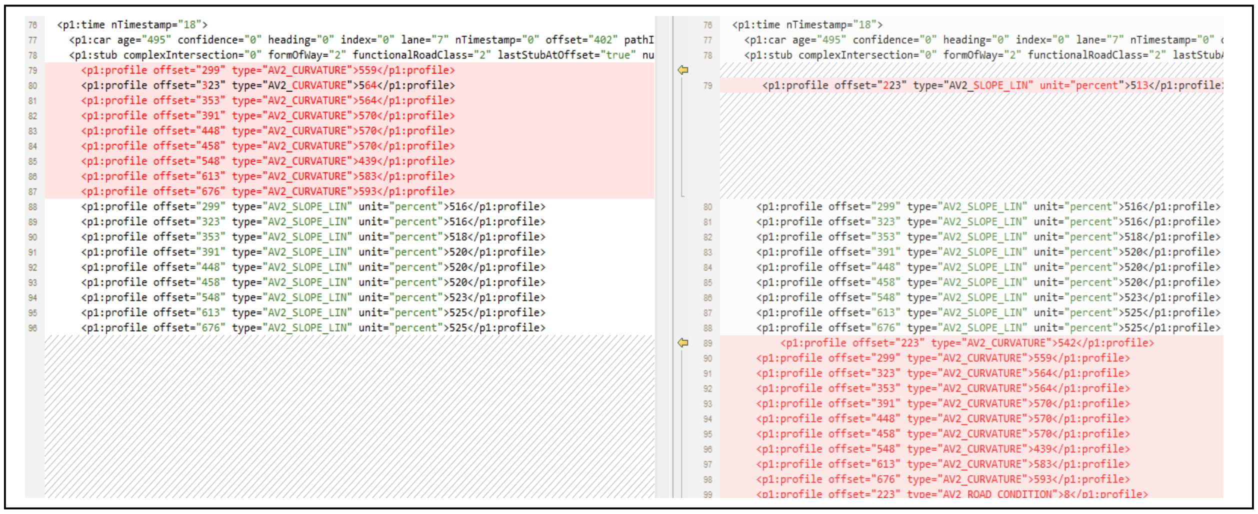
```
<p1:time nTimestamp="18">
  <p1:car age="495" confidence="0" heading="0" index="0" lane="7"
  <p1:stub complexIntersection="0" formOfWay="2" functionalRoadCla
    <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
    <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
    <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
    <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>
```

Reference.xml

Figure 6.8: second objective – detect order change in same “Type” attribute

In second scenario, there is order change detected in same type. However on having a closer look at points highlighted by green mark, we can observe that these points are not violating any rules. These are valid points hence, our algorithm should not detect them as an error. This is something not possible with generic tool which we use so far in our firm.

Failure in BeyondCompare Tool



```
<p1:time nTimestamp="18">
  <p1:car age="495" confidence="0" heading="0" index="0" lane="7" nTimestamp="0" offset="402" pathI
  <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClass="2" lastStubAtOffset="true" nu
    <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
    <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
    <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
    <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>
    <p1:profile offset="299" type="AV2_SLOPE_LIN" unit="percent">516</p1:profile>
    <p1:profile offset="323" type="AV2_SLOPE_LIN" unit="percent">516</p1:profile>
    <p1:profile offset="353" type="AV2_SLOPE_LIN" unit="percent">518</p1:profile>
    <p1:profile offset="391" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
    <p1:profile offset="448" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
    <p1:profile offset="458" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
    <p1:profile offset="548" type="AV2_SLOPE_LIN" unit="percent">523</p1:profile>
    <p1:profile offset="613" type="AV2_SLOPE_LIN" unit="percent">525</p1:profile>
    <p1:profile offset="676" type="AV2_SLOPE_LIN" unit="percent">525</p1:profile>
```

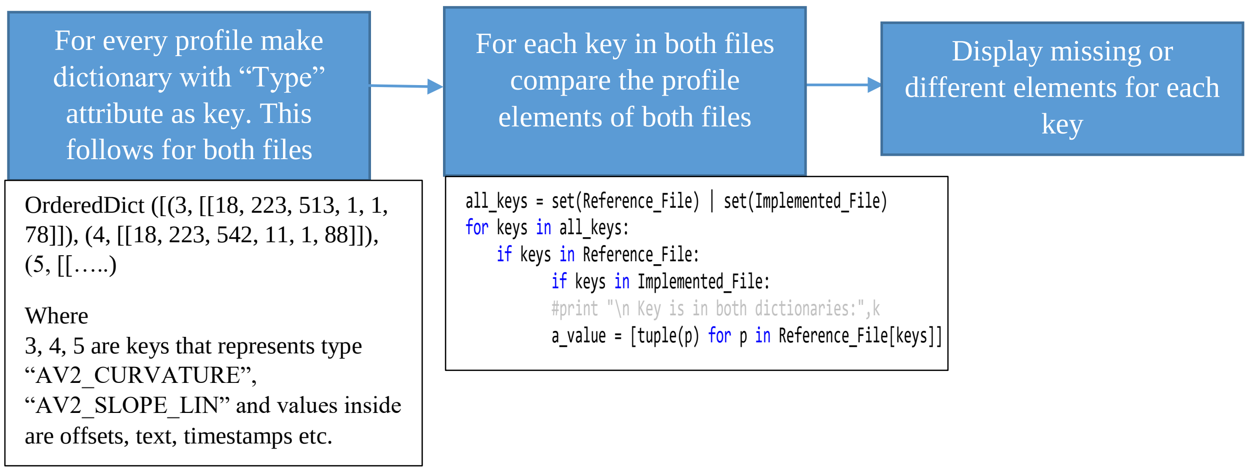
```
<p1:time nTimestamp="18">
  <p1:car age="495" confidence="0" heading="0" index="0" lane="7" nTimestamp="0" c
  <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClass="2" lastStub
      <p1:profile offset="223" type="AV2_SLOPE_LIN" unit="percent">513</p1:profile
    <p1:profile offset="299" type="AV2_SLOPE_LIN" unit="percent">516</p1:profile>
    <p1:profile offset="323" type="AV2_SLOPE_LIN" unit="percent">516</p1:profile>
    <p1:profile offset="353" type="AV2_SLOPE_LIN" unit="percent">518</p1:profile>
    <p1:profile offset="391" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
    <p1:profile offset="448" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
    <p1:profile offset="458" type="AV2_SLOPE_LIN" unit="percent">520</p1:profile>
    <p1:profile offset="548" type="AV2_SLOPE_LIN" unit="percent">523</p1:profile>
    <p1:profile offset="613" type="AV2_SLOPE_LIN" unit="percent">525</p1:profile>
    <p1:profile offset="676" type="AV2_SLOPE_LIN" unit="percent">525</p1:profile>
        <p1:profile offset="223" type="AV2_CURVATURE">542</p1:profile>
    <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
    <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
    <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
    <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
    <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
    <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>
    <p1:profile offset="223" type="AV2_ROAD_CONDITION">8</p1:profile>
```

Figure 6.9: BeyondCompare error

As we discussed about BeyondCompare tool in state of the art section. In above figure it fails to detect order change and mentions difference in data which has no difference. Lines in red denotes difference or missing elements.

My Implementation for first objective:

To tackle problem mentioned in first scenario of Figure 6.6 I have developed a working approach of comparing elements despite order change. It works as follows:


For every profile make dictionary with "Type" attribute as key. This follows for both files
For each key in both files compare the profile elements of both files
Display missing or different elements for each key

OrderedDict ([(3, [[18, 223, 513, 1, 1, 78]]), (4, [[18, 223, 542, 11, 1, 88]]), (5, [[…..)

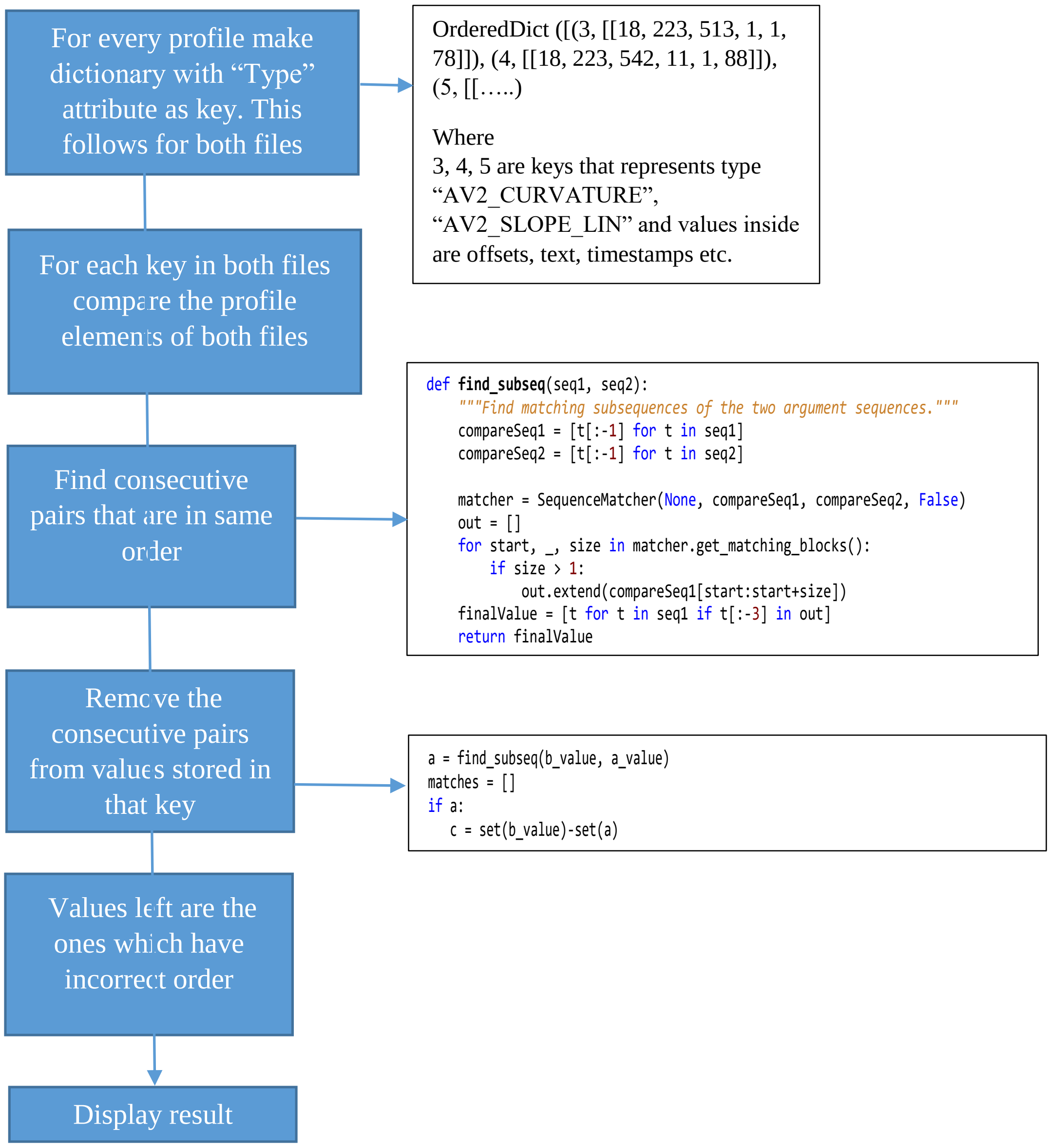
Where
3, 4, 5 are keys that represents type "AV2_CURVATURE", "AV2_SLOPE_LIN" and values inside are offsets, text, timestamps etc.

```
all_keys = set(Reference_File) | set(Implemented_File)
for keys in all_keys:
    if keys in Reference_File:
            if keys in Implemented_File:
            #print "\n Key is in both dictionaries:",k
            a_value = [tuple(p) for p in Reference_File[keys]]
```

Figure 6.10: First scenario solution

My implementation for second objective:

For solving the problem in second scenario as listed in Figure 6.7, we have to ignore the consecutive pairs of data which is in right order for both files. And later output should be such data which is not in order resulting in only different data to be displayed.


For every profile make dictionary with “Type” attribute as key. This follows for both files
For each key in both files compare the profile elements of both files
Find consecutive pairs that are in same order
Remove the consecutive pairs from values stored in that key
Values left are the ones which have incorrect order
Display result

OrderedDict ([(3, [[18, 223, 513, 1, 1, 78]]), (4, [[18, 223, 542, 11, 1, 88]]), (5, [[…..)

Where
3, 4, 5 are keys that represents type “AV2_CURVATURE”, “AV2_SLOPE_LIN” and values inside are offsets, text, timestamps etc.

```
def find_subseq(seq1, seq2):
    """Find matching subsequences of the two argument sequences."""
    compareSeq1 = [t[:-1] for t in seq1]
    compareSeq2 = [t[:-1] for t in seq2]

    matcher = SequenceMatcher(None, compareSeq1, compareSeq2, False)
    out = []
    for start, _, size in matcher.get_matching_blocks():
        if size > 1:
            out.extend(compareSeq1[start:start+size])
    finalValue = [t for t in seq1 if t[:-3] in out]
    return finalValue
```

```
a = find_subseq(b_value, a_value)
matches = []
if a:
   c = set(b_value)-set(a)
```

Figure 6.11: second scenario solution

6.3. Saving information of Comparison

As discussed in section 4.5.2 about saving comparison results. Here the implementation of saving results into XML file is described in brief.

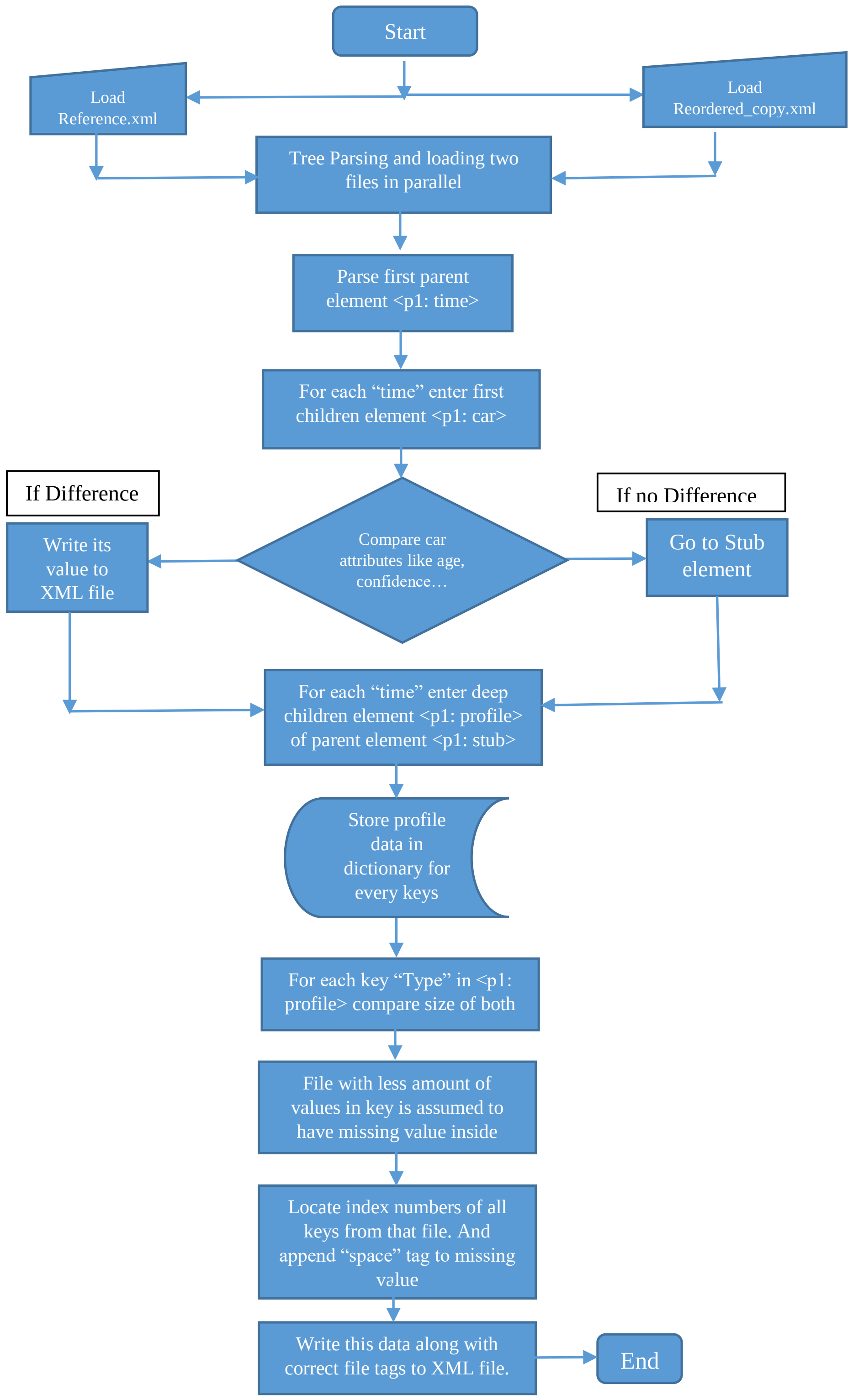

Start
Load Reference.xml
Load Reordered_copy.xml
Tree Parsing and loading two files in parallel
Parse first parent element <p1: time>
For each "time" enter first children element <p1: car>
If Difference
If no Difference
Compare car attributes like age, confidence…
Write its value to XML file
Go to Stub element
For each "time" enter deep children element <p1: profile> of parent element <p1: stub>
Store profile data in dictionary for every keys
For each key "Type" in <p1: profile> compare size of both
File with less amount of values in key is assumed to have missing value inside
Locate index numbers of all keys from that file. And append "space" tag to missing value
Write this data along with correct file tags to XML file.
End

Figure 6.12: Saving comparison results to XML file

Algorithm Description

Above implementation of XML file for saving comparison results is done in reference to the steps explained in section 4.5.2. Main aim of this algorithm is to save all comparison results in a XML file which can be later used for formatting purpose. So much of its implementation is same as comparison as far as parsing of two files is concerned.

Only change is when results are generated they are written in another XML file which includes all the necessary information like timestamp, stub number, file etc. where error was generated. In above shown algorithm since all the information of stages related to comparison are covered before. Here the major focus is given on explaining implementation of file saving feature.

1). Writing Car Difference

```
if age=='False' or confidence=='False' or heading == 'False' or index == 'False' or lane ==
'False' or nTimestamp == 'False' or offset == 'False' or pathId == 'False' or probablity ==
'False' or speed == 'False' or vpStat == 'False':

     etree.SubElement(self.AHR, 'Car')
     self.SavedError.append(self.MainRoot)
     writeValue = minidom.parseString(etree.tostring(self.SavedError)).toprettyxml()
     with open('C:\Users\BHD4ABT\Desktop\saveddata.xml','w') as f:
               f.write(writeValue)
```

Code 14: Writing Car Difference (saving_file.py)

Writing difference of Car element is far simplistic. Here for every "timestamp" we compare attributes of both files. If they are matching, the result is "True" or else it is "False".

If result is false we proceed further to write it. In this file new main element is created which will have "timestamp" value listed in which data was found faulty. Like stated in above figure later new sub element with name "Car" is created and result is written to the file – in our case it is "saveddata.xml"

2). Writing Profile difference inside Stub element

Profile element is located inside Stub element. And while writing difference we are more concerned about differences in Profile elements. In above algorithm we have to detect the size of two keys from both files. If the size of key in one file is less than that of another, then there is a missing element inside.

```
a_value = [tuple(p) for p in self.Reference_File[keys]]
b_value = [tuple(p) for p in self.Implemented_File[keys]]

if len(a_value)<len(b_value):
       self.compareFlag = 'true'
       space = len(b_value)-len(a_value)
       for values in range(space):
           a_value.append((0,0,0,'space'))

if len(b_value)<len(a_value):
      self.compareFlag = 'true'
      space = len(a_value)-len(b_value)
      for values in range(space):
          b_value.append((0,0,0,'space'))
```

Code 15: Calculating length of values inside keys (saving_file.py)

“a_value” refers to the values found in “Reference.xml” and “b_value” refers to values found in “Reordered_copy.xml”. Wherever this missing elements are found we have to append a tag – in our case it is string “space”.

This means that when this file is loaded back in tool, at whichever instance this string is found we have to insert an empty line in the text widget which carries our xml data. Moreover, if this is not achieved then this file can also be used by developer to load back results in tool and observe the difference.

Based on file in which difference was found differences are written to another XML file using method given in code below.

The way this data is written onto file is shown below:

```
for location in a_value:
    value_ref.append(location[3])

for values in b_value:
    copy_value.append(values[3])

if 'space' in value_ref:
     self.AHR = etree.SubElement(self.MainRoot, "RefAHR")
     self.stub = etree.SubElement(self.AHR, 'Stub', attrib = self.stubAttrib)
     etree.SubElement(self.stub, 'Profile',
                      type = keys,
                      location = str(value_ref).strip('[]'))

if 'space' in copy_value:
    self.TestAHR = etree.SubElement(self.MainRoot, "TestAHR")
    self.TestStub = etree.SubElement(self.TestAHR, 'Stub', attrib = self.stubAttrib)
    etree.SubElement(self.TestStub, 'Profile',
                     type = keys,
                     location = str(copy_value).strip('[]'))

self.SavedError.append(self.MainRoot)
writeValue = minidom.parseString(etree.tostring(self.SavedError)).toprettyxml()
with open('C:\Users\BHD4ABT\Desktop\saveddata.xml','w') as f:
          f.write(writeValue)
```

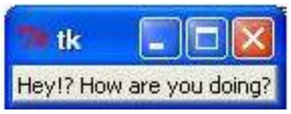
Code 16: Writing profile value to XML file (saving_file.py)

6.4. Displaying Information of Comparison

This section deals with displaying of results on the user interface. It also deals with how usage of Tkinter library was made to build the interface.

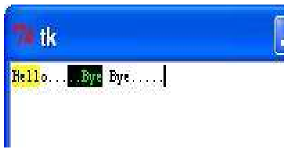
Before we move on to structure of how I implemented user interface I want to depict the important widgets that Tkinter provides and I made extensive use of:

Label: This is a widget which provides display box for implementing text or images. It is defined in following manner:



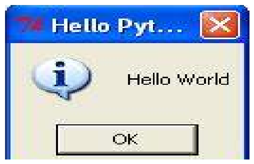
Hey!? How are you doing?

Text field: This widget allows to edit multiline text or insert any text inside it. You can get index number of string or highlight specific string.



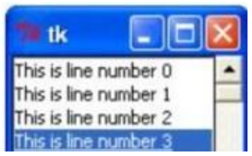
Hello.....Bye Bye.....

Buttons: This widget allows us to bind any even on button press. Majority of times I have bind functions with button press.



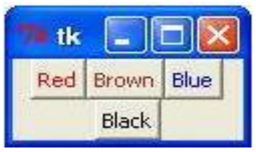
Hello World

Scrollbar: This widget allows vertical and horizontal scrolling of widgets on frame or canvas.



This is line number 0
This is line number 1
This is line number 2
This is line number 3

Frame: Lastly there is Frame which is used to group and organize above given widgets. It helps in arranging widgets in given position.



6.4.1. Developing User Interface

As shown in section 4.4.4 I made use of Tkinter as my GUI interface building library. Following is the algorithm that was used to create needed interface:

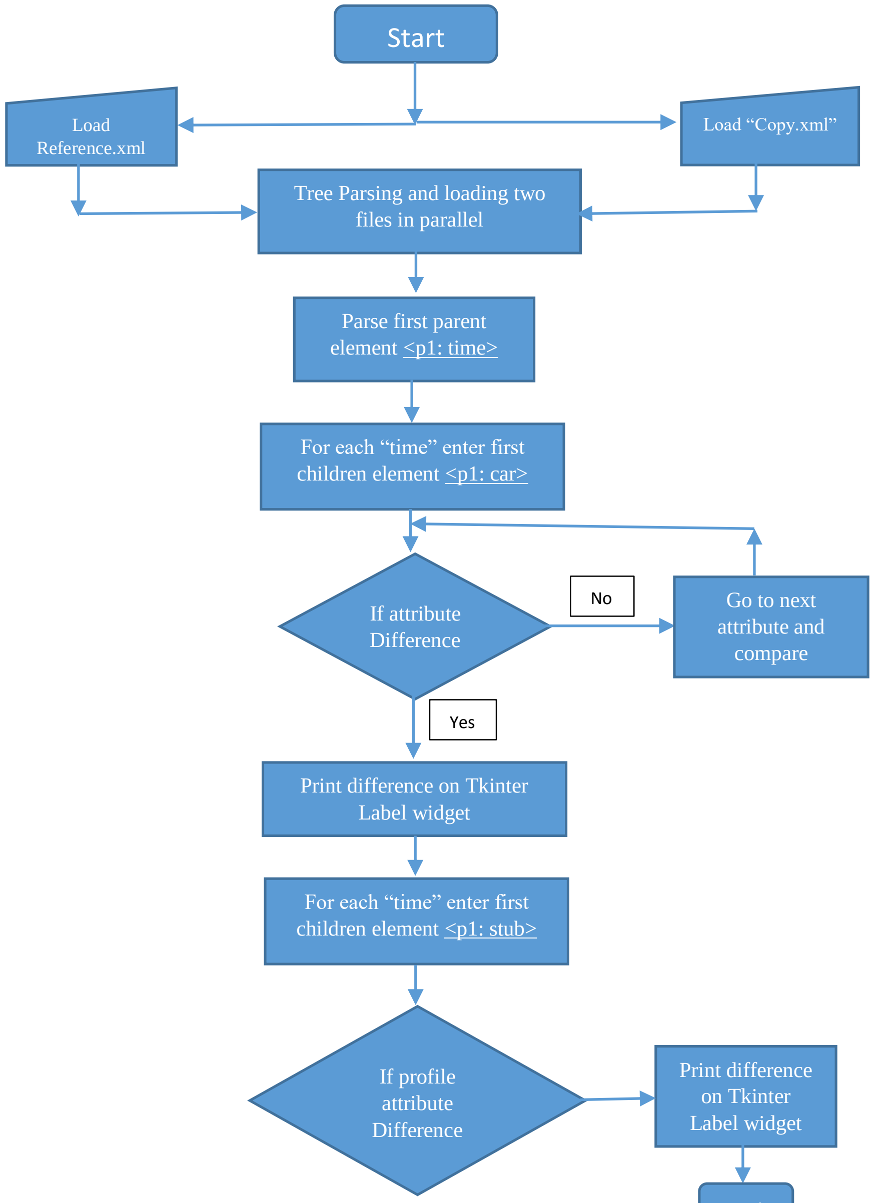

Start
Load Reference.xml
Load "Copy.xml"
Tree Parsing and loading two files in parallel
Parse first parent element <p1: time>
For each "time" enter first children element <p1: car>
If attribute Difference
No
Go to next attribute and compare
Yes
Print difference on Tkinter Label widget
For each "time" enter first children element <p1: stub>
If profile attribute Difference
Print difference on Tkinter Label widget
End

Figure 6.13: Approach for user interface

Flowchart explained above shows how comparison results are saved on widgets provided by Tkinter library. For every instance of compared results it is written on GUI widgets. This widgets are created only when difference is sensed. All the differences are then presented over the Label widget as described in section 6.4.

6.4.2. Binding results to User Interface

In this section complete working results are not shown. Rather bigger part focus is given on explaining how results were bonded with user interface. In section 7.1 I have shown full working results of my implementation.

1). Building options for opening files

For taking input of files a user interface is built in such a way that it asks for option of opening two files for comparison – “Reference.xml” and “Reordered_copy.xml” comparatively.

```
self.page1ref = Button(page1, text="Open Reference File",
                       command=self.RefSelect_load_file)
self.page1ref.grid(row =1, column=0, padx = 10, pady=30)
self.page1RefLabel = Text(page1, height=1, width = 60)
self.page1RefLabel.grid(row=1, column=1, padx=10, pady=10)
self.page1cop = Button(page1, text="Open Copy File",
                      command=self.CopySelect_load_file)
self.page1cop.grid(row=2, column=0, pady=10)
```

```
fileOpen = open(self.reffname)
self.labelReference.insert(END, self.reffname)
ReferenceXML.insert(END,fileOpen.read())
```

```
fileOpen = open(self.copyfname)
self.labelCopy.insert(END,self.copyfname)
CopyXML.insert(END, fileOpen.read())
```

Code 17: Opening/Loading files for comparison (stub_structure.py)

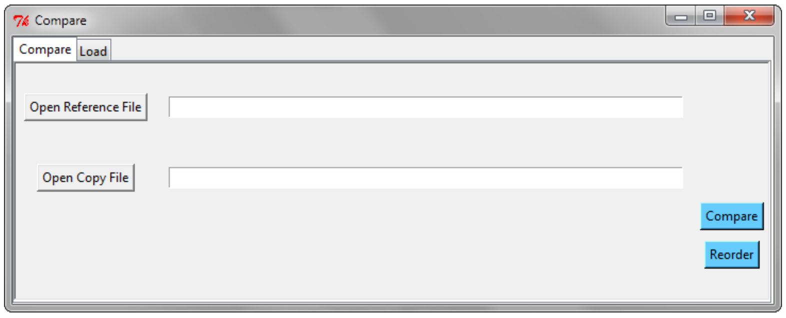


Figure 6.14: Interface for opening/loading files

In above figure I have shown how implementation of file loading interface was done. The first coding box in Code 17 suggests how buttons are designed to ask for opening reference and copy files. Then in next code box it is displayed how files are opened and the data is then inserted into text boxes which are meant to display two files. “ReferenceXML” and “CopyXML” are two text boxes meant to display the content.

2). Displaying reordered data

Before comparison the data has to be reordered so there has to be option to reorder the current "Copy.xml" as we shown in Figure 6.13. For the reordering we provided the button option in file open/close dialogue box in following manner:

```
self.ReorderFile = Button(page1, text="Reorder", height = 1, bg = '#66CCFF',
                            command=self.make_reorder_file)


def make_reorder_file(self):
        self.saveFile="C:/Users/BHD4ABT/Desktop/ReorderData.xml"
        status.config(text="Reordering initiated. Please wait. Writing files")
        Reordering = Reorder()
        Reordering.reOrder(self.reffname,self.copyfname, self.saveFile,
         pbar_f, bottomframe5)

        pbar_f["value"] = 0
        status.config(text="Reordering done. File saved in desktop")
```

Code 18: Displaying Reordered Data (stub_structure.py)

So pressing a button creates an event which calls function "make_reorder_file" function which calls "Reordering" class and saves the file in desktop. The status is also shown to user in the form of progress bar.

3). Displaying comparison results

As discussed earlier for showing difference on user interface, there is need to bind the output to the user interface. So to perform this appropriate Frame widget has to be made and for every difference we have to make a Label on that Frame as shown in below figure with label "ttk.Label" and inside that label we have to save our results.

```
self.t = ResultFrame(self.callbackFrame, text= 'Timestamp:' + i.get('nTimestamp'), re-
lief="raised", borderwidth=1, width=20)
self.t.pack(fill="x", expand=1, pady=2, padx=2, anchor="n")

string = 'Age Difference'
self.var=[self.t, int(i.get('nTimestamp'))]
carLabel = ttk.Label(self.t.error_frame, text=string, justify=LEFT, width=20)
carLabel.pack(fill="x", expand=1, pady=2, padx=2, anchor="n")
```

Code 19: Creating Frame and Label for Car element (Children Label) (stub_structure.py)

Apart from difference we also make a parent label for timestamp in which this differences are developed. They are shown in below figure.

```
self.title_frame = ttk.Frame(self)
self.title_frame.pack(fill="x", expand=1)
ttk.Label(self.title_frame, width=20, text=text).pack(side="right", fill="x", expand=1) #for
showing title on error frames

self.error_button = ttk.Checkbutton(self.title_frame, text='+',
                                    command=self.plusButton,
variable=self.show, width=7, style='Toolbutton')
```

Code 20: Creating label for Timestamp element (Parent label) (stub_structure.py)

Above mentioned figure is the way we write car difference onto the label used in the user interface. Now we will discuss about how we bind compared output that was generated out of stub element.

For stub element we use same method that we used for showing car difference. But here some checks has to be performed that makes sure that no frames are overwritten. So if the difference from stub and car element are generated from same timestamp then there should be common parent label like shown in Figure 6.14. Under this parent label the difference should be children labels comprising of car and stub element.

A working sample looks like this:

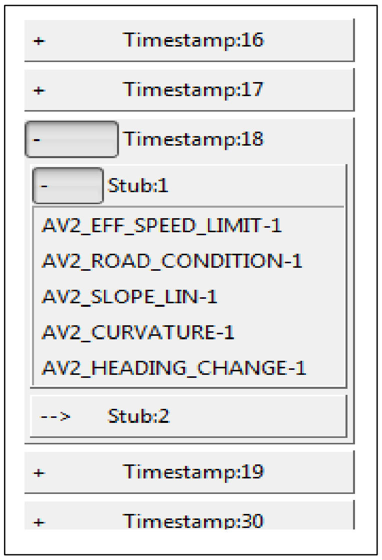


Figure 6.15: Comparison Results

4). Highlighting difference

After listing the difference out on user interface there should also be provision for highlighting errors. So a defined highlight class gets called whenever user clicks on label that defines difference.

Highlighting Class

```
def highlightStub(self,event,start,end,copyMissing,refMissing,color,label):

        findRefScrollPos=float(ReferenceXML.search(start, '1.0', stopindex=END))
        autoScrollRefXML = ReferenceXML.index(str(findRefScrollPos))
        ReferenceXML.see(autoScrollRefXML)
        findCopyScrollPos=float(CopyXML.search(start, '1.0', stopindex=END))
        autoScrollCopyXML = CopyXML.index(str(findCopyScrollPos))
        CopyXML.see(autoScrollCopyXML)

        for index in copyMissing:
            self.color_config(event, label, color)
            sequence = 0
            ReferenceXML.tag_configure("yellow", background="yellow")
            CopyXML.tag_configure("yellow", background="yellow")
            end = ReferenceXML.search(end, '1.0', stopindex=END)
```

Capturing button events

```
stubLabel.bind("<Button-1>",lambda event, start = selectedTimestamp,
       end = '<p1:time nTimestamp="{0}">'.format(int(i.get('nTimestamp'))+1),
       copyMissing = copyValue, refMissing = refValue, label=stubLabel,
       color = "red": self.highlightStub(event,start,end,copyMissing,
       refMissing,color,label))
```

Code 21: highlighting events and calling class (stub_structure.py)

<u>Highlighting algorithm works in following manner:</u>

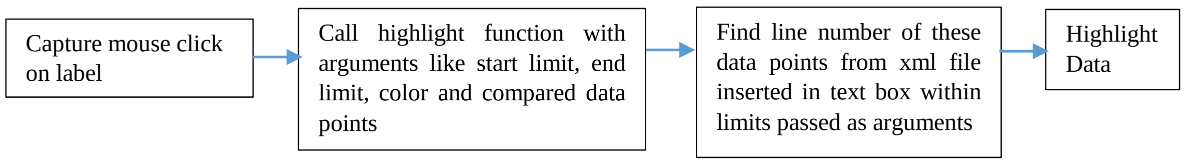
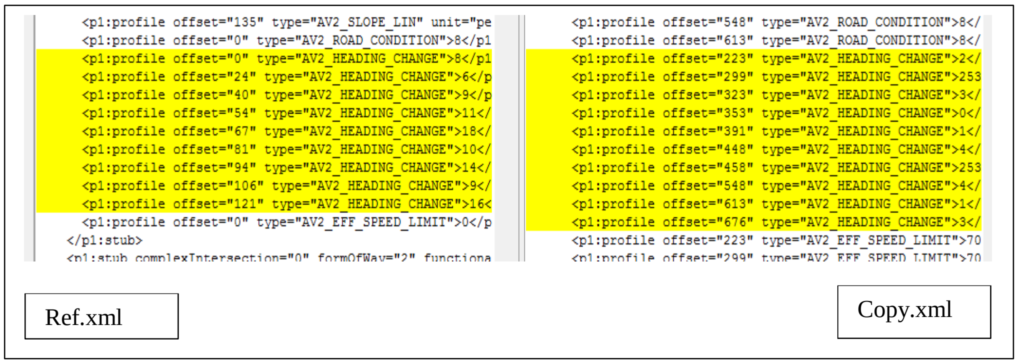

Capture mouse click on label
Call highlight function with arguments like start limit, end limit, color and compared data points
Find line number of these data points from xml file inserted in text box within limits passed as arguments
Highlight Data

Figure 6.16: Highlighting algorithm

Our main aim is to highlight xml lines which is stored in text box. For that difference has to be located which matches xml lines in both text box that is used to show our data in user interface. To detect the correct line we have to find the data in correct range. So starting and ending limits are set on specific timestamps in which error was detected. Lastly, based on line number of those errors situated in timestamps entire line is highlighted.

Sample of highlighting is presented below:



```
    <p1:profile offset="135" type="AV2_SLOPE_LIN" unit="pe
    <p1:profile offset="0" type="AV2_ROAD_CONDITION">8</p1
    <p1:profile offset="0" type="AV2_HEADING_CHANGE">8</p1
    <p1:profile offset="24" type="AV2_HEADING_CHANGE">6</p
    <p1:profile offset="40" type="AV2_HEADING_CHANGE">9</p
    <p1:profile offset="54" type="AV2_HEADING_CHANGE">11</
    <p1:profile offset="67" type="AV2_HEADING_CHANGE">18</
    <p1:profile offset="81" type="AV2_HEADING_CHANGE">10</
    <p1:profile offset="94" type="AV2_HEADING_CHANGE">14</
    <p1:profile offset="106" type="AV2_HEADING_CHANGE">9</
    <p1:profile offset="121" type="AV2_HEADING_CHANGE">16<
    <p1:profile offset="0" type="AV2_EFF_SPEED_LIMIT">0</p
  </p1:stub>
  <p1:stub complexIntersection="0" formOfWay="2" functiona
```

```
    <p1:profile offset="548" type="AV2_ROAD_CONDITION">8</
    <p1:profile offset="613" type="AV2_ROAD_CONDITION">8</
    <p1:profile offset="223" type="AV2_HEADING_CHANGE">2</
    <p1:profile offset="299" type="AV2_HEADING_CHANGE">253
    <p1:profile offset="323" type="AV2_HEADING_CHANGE">3</
    <p1:profile offset="353" type="AV2_HEADING_CHANGE">0</
    <p1:profile offset="391" type="AV2_HEADING_CHANGE">1</
    <p1:profile offset="448" type="AV2_HEADING_CHANGE">4</
    <p1:profile offset="458" type="AV2_HEADING_CHANGE">253
    <p1:profile offset="548" type="AV2_HEADING_CHANGE">4</
    <p1:profile offset="613" type="AV2_HEADING_CHANGE">1</
    <p1:profile offset="676" type="AV2_HEADING_CHANGE">3</
    <p1:profile offset="223" type="AV2_EFF_SPEED_LIMIT">70
    <p1:profile offset="299" type="AV2_EFF_SPEED_LIMIT">70
```

Figure 6.17: Highlighting Results

As given in figure at some specific timestamps two sets of “AV2_HEADING_CHANGE” differed greatly. The difference can be seen in “offset” as these two lines in both files are highlighted. On opening two files in file dialogue box they are inserted into text box.

6.5. Implementing interface for CAN Script Generator

In this section I have shown my implementation work concerning CAN script generator. I have explained program structure which depicts a flowchart explaining my work flow. In later section I have briefly showed my logic implementation concerning the work flow.

6.5.1. Program Structure

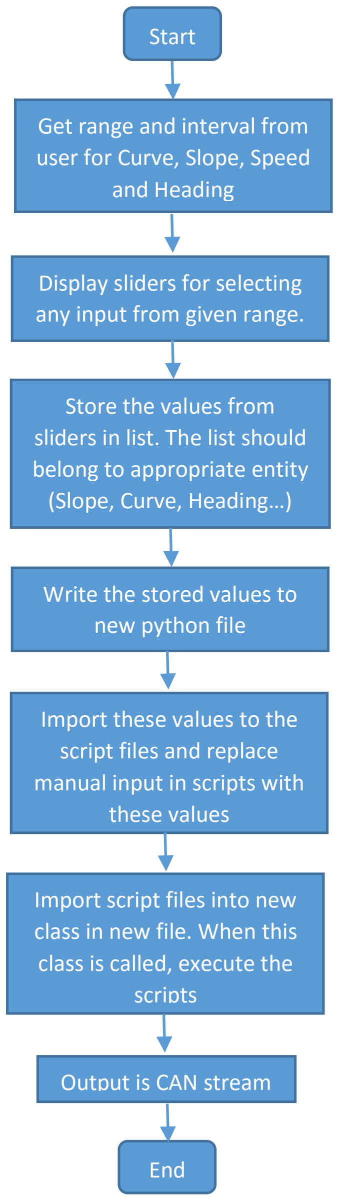

Start
Get range and interval from user for Curve, Slope, Speed and Heading
Display sliders for selecting any input from given range.
Store the values from sliders in list. The list should belong to appropriate entity (Slope, Curve, Heading…)
Write the stored values to new python file
Import these values to the script files and replace manual input in scripts with these values
Import script files into new class in new file. When this class is called, execute the scripts
Output is CAN stream
End

Figure 6.18: CAN stream generator Implementation

6.5.2. Logic Implementation

Here logic is described for implementing stages that was explained in section 6.5.1. It is depicted in the form of coding snippets and wherever possible have shown some working interface pics.

File structure of CAN stream generator is organized into following category

can_structure.py	My implementation of user interface for the can script generation
can.py	Used for storing user selected values from the interface
generation_script.py	Script file where can.py is imported and all values are made available for the scripts to generate can data. It also imports framework.py
framework.py	Defines attributes for generating a can data. It has all the attributes like road geometry, car distance, path ids, stubs etc. this data is then made available in the form of CAN stream

Table 6: CAN Script Generator (File Structure)

1). Getting range and interval from user

Here dialogue box is designed which will ask user to enter range and set intervals. There will be option for users to enter different values for different parameters. These parameters are then given to sliders on the basis of which sliders determine their range.

```
# for curve label

curveLabel = Label(self.menu_frame1,text='Curve',bg='cyan')
curveLabel.pack(side=LEFT)

curveEntry1 = Entry(self.menu_frame1)
curveEntry1.pack(side=LEFT, padx=20)

curveEntry2 = Entry(self.menu_frame1)
curveEntry2.pack(side=LEFT, padx=20)
```

Output:

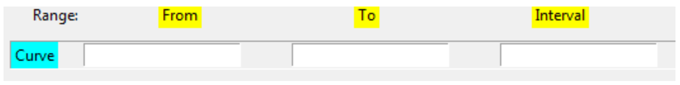


Figure 6.19: Creating entries (can_structure.py)

Curve label, Slope label, heading label etc. are the entries that we are interested in taking values for. Just like in above figure an output represents a simple interface to get values from user.

2). Displaying Sliders for selecting values over given range

Sliders are part of Tkinter library and they are defined by “Scale” widget for which following syntax is used:

```
self.slideCurve = Scale(curveFrame, from_=int(curveEntry1.get()), to=int(curveEntry2.get()),
                             length = 700, tickinterval=int(curveEntry3.get()), orient=HORIZONTAL)
self.slideCurve.pack()
```

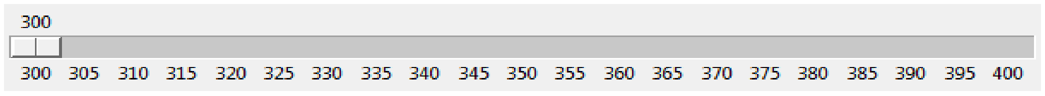


Figure 6.20: Scale widget Tkinter (can_structure.py)

Above mentioned sliders are used to retrieve the values from user for given range. So the values that user enters in Figure 6.19 are connected with the variables “from_”, “to” and “tickinterval” of scale widget. This variables can be viewed in above figure 6.20.

3). Storing and writing values

In this stage values are retrieved which are stored by user using sliders or any other input field. These values are then written to another python file.

can_structure.py

```
def write_values(self):

     print self.curve
     print self.curveEntry.get()

     f=open('C:\Users\BHD4ABT\Desktop\can.py','w')
     f.write('CurveSlider={}'.format(self.curve))
     f.write('\n')
     f.write('CurveManual=[{}]'.format(self.curveEntry.get()))
     f.close()
```

can.py

```
CurveSlider=[3135, 3437, 3698, 3940, 4123, 3985]
CurveManual=[56,78,98,89]
```

Code 22: Write values entered by user to file

4). Importing newly written file in script

Now that file with newly written values is generated, they have to be imported into script file so that all the values shown in above figure are accessed by the scripts to generate CAN data based on these files.

This is done in following manner:

```
from framework import *
from can import *
# create the sub pathes / max number
subpathId = 9
for offset in CurveSlider:
    msg = StubMessage()
    msg.offset = offset
    msg.pathId = 8
    msg.subPathId = subpathId
    msg.turnAngle = 100
    subpathId += 1
    print msg.toString()
    msg.toFile(f)
```

Code 23: importing values and CAN stream generation (generation_script.py)

5). Generating CAN data

CAN stream can be viewed in figure 2.2 and section 2.1.1. After implementing above steps one final work is to call the scripts from program. Once it is called CAN stream is generated and user can use interface for writing different values for desired output.

In reference to code in Code 23, "generation_script.py" imports framework class inside its own file. It then calls methods located in "framework.py" like stubMessage(), positionMessage() etc. which assigns attributes in the form of CAN stream according to received arguments. A sample of such class can be viewed below:

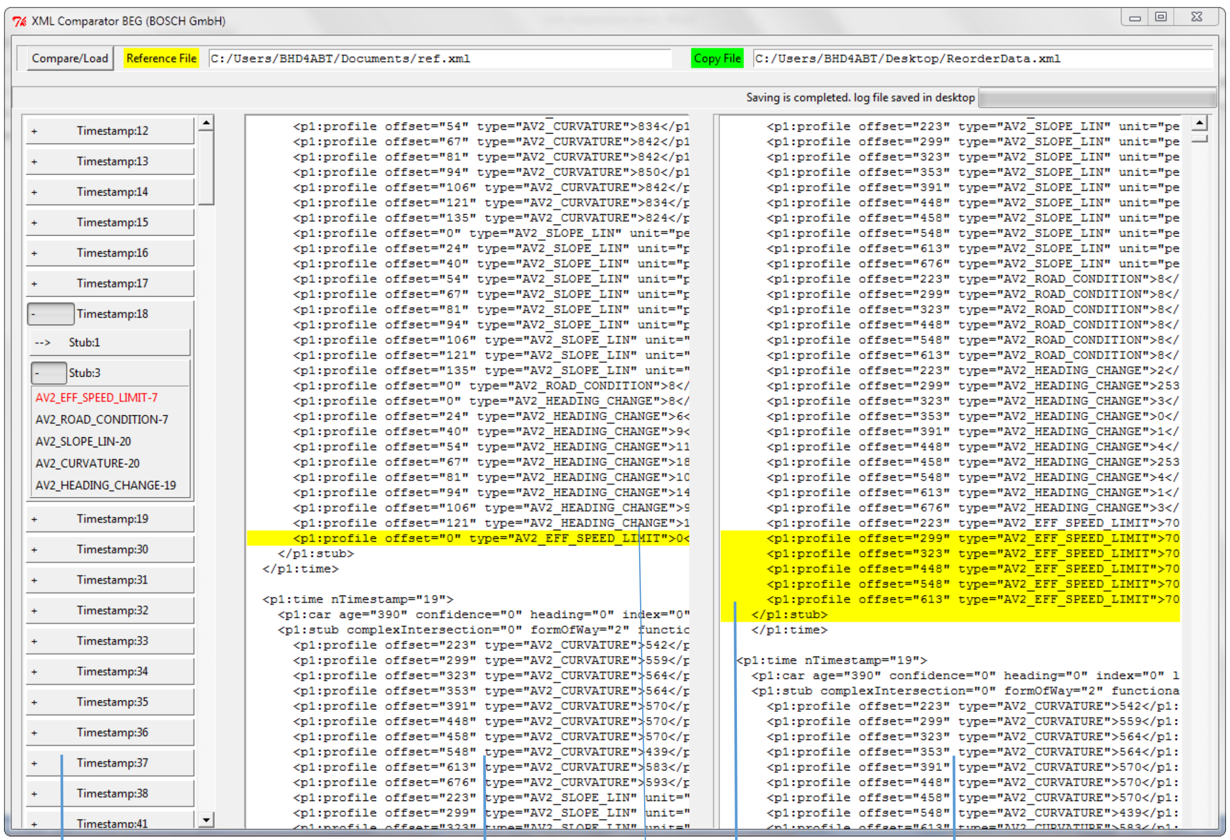
```
class StubMessage(AdasisMessage):
# ---------------------------------------------------------------------------
    _cyclicCtr = 0
    #
    # Constructor
    #
    # rawData (mandatory) - raw data bytes from CAN message (size = 8 bytes)
    def __init__(self, rawData=None):
        AdasisMessage.__init__(self,rawData)
        if rawData is not None:
            # take values from rawData
            self.decode()
        else:
            # set default values
            self.set_msg_type(MessageType.AV2_MSG_STUB)
            self.set_sub_path_id(0) # unknown
            self.set_last_stub(0) # 0 = false
            self.set_turn_angle(255) # N/A
            self.set_update(0) # false = no update
            self.set_rel_probb(31) # N/A
```

Code 24: StubMessage class (framework.py)

7. Results

In this concluding section of implementation I want to present some snapshots of working results that were generated. Along with snapshots the description of features are also explained.

7.1. XML Comparator



C:/Users/BHD4ABT/Documents/ref.xml

C:/Users/BHD4ABT/Desktop/ReorderData.xml

Saving is completed. log file saved in desktop

```
+ Timestamp:12
+ Timestamp:13
+ Timestamp:14
+ Timestamp:15
+ Timestamp:16
+ Timestamp:17
- Timestamp:18
  --> Stub:1
  - Stub:3
    AV2_EFF_SPEED_LIMIT-7
    AV2_ROAD_CONDITION-7
    AV2_SLOPE_LIN-20
    AV2_CURVATURE-20
    AV2_HEADING_CHANGE-19
+ Timestamp:19
+ Timestamp:30
+ Timestamp:31
+ Timestamp:32
+ Timestamp:33
+ Timestamp:34
+ Timestamp:35
+ Timestamp:36
+ Timestamp:37
+ Timestamp:38
+ Timestamp:41
```

```
      <p1:profile offset="54" type="AV2_CURVATURE">834</p1
      <p1:profile offset="67" type="AV2_CURVATURE">842</p1
      <p1:profile offset="81" type="AV2_CURVATURE">842</p1
      <p1:profile offset="94" type="AV2_CURVATURE">850</p1
      <p1:profile offset="106" type="AV2_CURVATURE">842</p
      <p1:profile offset="121" type="AV2_CURVATURE">834</p
      <p1:profile offset="135" type="AV2_CURVATURE">824</p
      <p1:profile offset="0" type="AV2_SLOPE_LIN" unit="pe
      <p1:profile offset="24" type="AV2_SLOPE_LIN" unit="p
      <p1:profile offset="40" type="AV2_SLOPE_LIN" unit="p
      <p1:profile offset="54" type="AV2_SLOPE_LIN" unit="p
      <p1:profile offset="67" type="AV2_SLOPE_LIN" unit="p
      <p1:profile offset="81" type="AV2_SLOPE_LIN" unit="p
      <p1:profile offset="94" type="AV2_SLOPE_LIN" unit="p
      <p1:profile offset="106" type="AV2_SLOPE_LIN" unit="
      <p1:profile offset="121" type="AV2_SLOPE_LIN" unit="
      <p1:profile offset="135" type="AV2_SLOPE_LIN" unit="
      <p1:profile offset="0" type="AV2_ROAD_CONDITION">8</
      <p1:profile offset="0" type="AV2_HEADING_CHANGE">8</
      <p1:profile offset="24" type="AV2_HEADING_CHANGE">6<
      <p1:profile offset="40" type="AV2_HEADING_CHANGE">9<
      <p1:profile offset="54" type="AV2_HEADING_CHANGE">11
      <p1:profile offset="67" type="AV2_HEADING_CHANGE">18
      <p1:profile offset="81" type="AV2_HEADING_CHANGE">10
      <p1:profile offset="94" type="AV2_HEADING_CHANGE">14
      <p1:profile offset="106" type="AV2_HEADING_CHANGE">9
      <p1:profile offset="121" type="AV2_HEADING_CHANGE">1
      <p1:profile offset="0" type="AV2_EFF_SPEED_LIMIT">0<
    </p1:stub>
  </p1:time>

  <p1:time nTimestamp="19">
    <p1:car age="390" confidence="0" heading="0" index="0"
    <p1:stub complexIntersection="0" formOfWay="2" functic
      <p1:profile offset="223" type="AV2_CURVATURE">542</p
      <p1:profile offset="299" type="AV2_CURVATURE">559</p
      <p1:profile offset="323" type="AV2_CURVATURE">564</p
      <p1:profile offset="353" type="AV2_CURVATURE">564</p
      <p1:profile offset="391" type="AV2_CURVATURE">570</p
      <p1:profile offset="448" type="AV2_CURVATURE">570</p
      <p1:profile offset="458" type="AV2_CURVATURE">570</p
      <p1:profile offset="548" type="AV2_CURVATURE">439</p
      <p1:profile offset="613" type="AV2_CURVATURE">583</p
      <p1:profile offset="676" type="AV2_CURVATURE">593</p
      <p1:profile offset="223" type="AV2_SLOPE_LIN" unit="
      <p1:profile offset="299" type="AV2_SLOPE_LIN" unit="
```

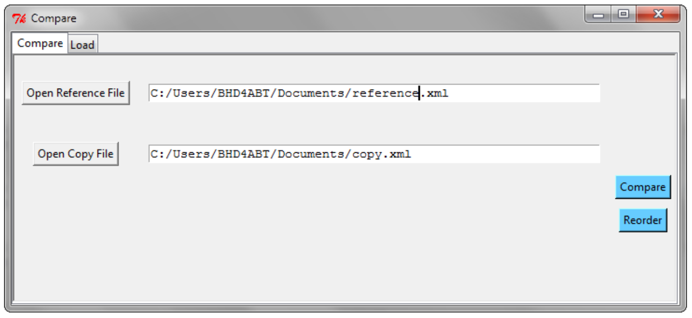
```
      <p1:profile offset="223" type="AV2_SLOPE_LIN" unit="pe
      <p1:profile offset="299" type="AV2_SLOPE_LIN" unit="pe
      <p1:profile offset="323" type="AV2_SLOPE_LIN" unit="pe
      <p1:profile offset="353" type="AV2_SLOPE_LIN" unit="pe
      <p1:profile offset="391" type="AV2_SLOPE_LIN" unit="pe
      <p1:profile offset="448" type="AV2_SLOPE_LIN" unit="pe
      <p1:profile offset="458" type="AV2_SLOPE_LIN" unit="pe
      <p1:profile offset="548" type="AV2_SLOPE_LIN" unit="pe
      <p1:profile offset="613" type="AV2_SLOPE_LIN" unit="pe
      <p1:profile offset="676" type="AV2_SLOPE_LIN" unit="pe
      <p1:profile offset="223" type="AV2_ROAD_CONDITION">8</
      <p1:profile offset="299" type="AV2_ROAD_CONDITION">8</
      <p1:profile offset="323" type="AV2_ROAD_CONDITION">8</
      <p1:profile offset="448" type="AV2_ROAD_CONDITION">8</
      <p1:profile offset="548" type="AV2_ROAD_CONDITION">8</
      <p1:profile offset="613" type="AV2_ROAD_CONDITION">8</
      <p1:profile offset="223" type="AV2_HEADING_CHANGE">2</
      <p1:profile offset="299" type="AV2_HEADING_CHANGE">253
      <p1:profile offset="323" type="AV2_HEADING_CHANGE">3</
      <p1:profile offset="353" type="AV2_HEADING_CHANGE">0</
      <p1:profile offset="391" type="AV2_HEADING_CHANGE">1</
      <p1:profile offset="448" type="AV2_HEADING_CHANGE">4</
      <p1:profile offset="458" type="AV2_HEADING_CHANGE">253
      <p1:profile offset="548" type="AV2_HEADING_CHANGE">4</
      <p1:profile offset="613" type="AV2_HEADING_CHANGE">1</
      <p1:profile offset="676" type="AV2_HEADING_CHANGE">3</
      <p1:profile offset="223" type="AV2_EFF_SPEED_LIMIT">70
      <p1:profile offset="299" type="AV2_EFF_SPEED_LIMIT">70
      <p1:profile offset="323" type="AV2_EFF_SPEED_LIMIT">70
      <p1:profile offset="448" type="AV2_EFF_SPEED_LIMIT">70
      <p1:profile offset="548" type="AV2_EFF_SPEED_LIMIT">70
      <p1:profile offset="613" type="AV2_EFF_SPEED_LIMIT">70
    </p1:stub>
    </p1:time>

  <p1:time nTimestamp="19">
    <p1:car age="390" confidence="0" heading="0" index="0" l
    <p1:stub complexIntersection="0" formOfWay="2" functiona
      <p1:profile offset="223" type="AV2_CURVATURE">542</p1:
      <p1:profile offset="299" type="AV2_CURVATURE">559</p1:
      <p1:profile offset="323" type="AV2_CURVATURE">564</p1:
      <p1:profile offset="353" type="AV2_CURVATURE">564</p1:
      <p1:profile offset="391" type="AV2_CURVATURE">570</p1:
      <p1:profile offset="448" type="AV2_CURVATURE">570</p1:
      <p1:profile offset="458" type="AV2_CURVATURE">570</p1:
      <p1:profile offset="548" type="AV2_CURVATURE">439</p1:
```

Figure 7.1: XML Comparator Main Window (Latest Interface)

This is the main window of XML comparator wherein comparison results are showed by list of labels for every timestamps in which data was found to be faulty. Violating points are shown stub wise so they are easy keep track off. Labels carry "Type" from xml data. "Reordered_copy.xml" is generated out of "Copy.xml". In image it cannot be shown but process is explained in section 6.1.

On pressing Compare/Load file a dialogue box opens which is described in below figure.

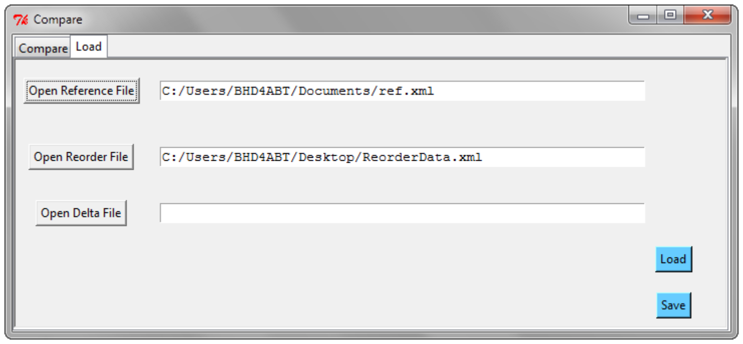
Compare and Reorder window



C:/Users/BHD4ABT/Documents/reference.xml

C:/Users/BHD4ABT/Documents/copy.xml

Reorder and Save



C:/Users/BHD4ABT/Documents/ref.xml

C:/Users/BHD4ABT/Desktop/ReorderData.xml

Figure 7.2: XML Comparator Compare/Load

First dialogue box is for reordering and comparing. Reordering has to be done before comparing because the original "Copy.xml" may have unordered stubs which needs to be arranged in proper order for comparison. If this step is skipped and direct comparison is done with original "Copy.xml" then the results will be predicted right by the tool, however they cannot be correctly interpreted by the user.

Second dialogue box is for loading comparison results while taking reordered file into consideration. Also there will be option for saving the current results. Current results are stored in the form of new XMl file which can be used for further calculations.

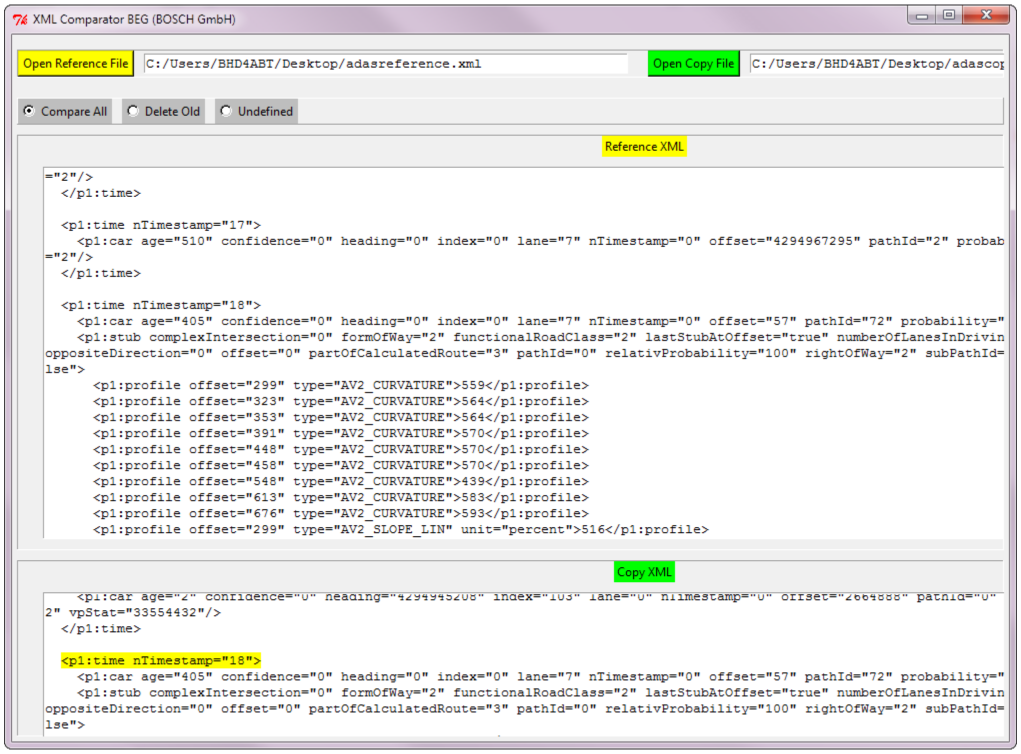
XML comparison results saving file as we discussed in section 6.3 are shown below. This file is generated on pressing save button in above dialogue box when user has completed reordering operation and has loaded that file inside the tool.

```xml
<Time nTimestamp="17">
        <TestAHR>
            <Car/>
        </TestAHR>
</Time>

<Time nTimestamp="18">
  <RefAHR>
     <Stub index="1">
         <Profile location="33, 34, 35, 36, 37, 'space'" type="AV2_EFF_SPEED_LIMIT"/>
         <Profile location="19, 20, 21, 22, 23, 'space'" type="AV2_ROAD_CONDITION"/>
         <Profile location="10, 11, 12, 13, 14, 15, 16, 17, 18,
                      'space'"type="AV2_SLOPE_LIN"/>
         <Profile location="1, 2, 3, 4, 5, 6, 7, 8, 9, 'space'"
                      type="AV2_CURVATURE"/>
          <Profile location="24, 25, 26, 27, 28, 29, 30, 31, 32, 'space'"
                      type="AV2_HEADING_CHANGE"/>
      </Stub>

       <Stub index="3">
           <Profile location="31, 'space', 'space', 'space', 'space', 'space'"
       type="AV2_EFF_SPEED_LIMIT"/>
            <Profile location="21, 'space', 'space', 'space', 'space', 'space'"
       type="AV2_ROAD_CONDITION"/>
            <Profile location="22, 23, 24, 25, 26, 27, 28, 29, 30, 'space'"
               type="AV2_HEADING_CHANGE"/>
       </Stub>
    </RefAHR>
 </Time>

<Time nTimestamp="19">
    <RefAHR>
       <Stub index="3">
          <Profile location="36, 'space', 'space', 'space', 'space', 'space'"
        type="AV2_EFF_SPEED_LIMIT"/>
        </Stub>
    </RefAHR>
</Time>
```

Code 25: Final saved data from comparator (saveddata.xml)

In above file "RefAHR" stands for Reference ADAS Horizon Reconstructor and it means the data belongs to "Reference.xml" and "TestAHR" stands for Test ADAS Horizon Reconstructor and this suggests that data located inside this element is taken from "Reordered_Copy.xml"

7.2. Old Comparator interface vs New Comparator Interface



C:/Users/BHD4ABT/Desktop/adasreference.xml

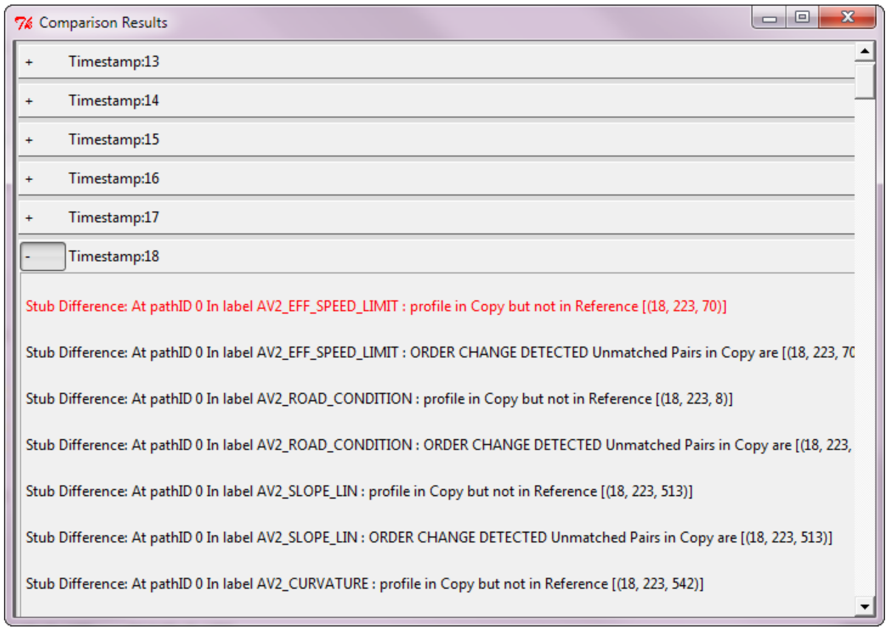
C:/Users/BHD4ABT/Desktop/adascop

(●) Compare All ( ) Delete Old ( ) Undefined

```
="2"/>
  </p1:time>

  <p1:time nTimestamp="17">
    <p1:car age="510" confidence="0" heading="0" index="0" lane="7" nTimestamp="0" offset="4294967295" pathId="2" probab
="2"/>
  </p1:time>

  <p1:time nTimestamp="18">
    <p1:car age="405" confidence="0" heading="0" index="0" lane="7" nTimestamp="0" offset="57" pathId="72" probability="
    <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClass="2" lastStubAtOffset="true" numberOfLanesInDrivin
oppositeDirection="0" offset="0" partOfCalculatedRoute="3" pathId="0" relativProbability="100" rightOfWay="2" subPathId=
lse">
      <p1:profile offset="299" type="AV2_CURVATURE">559</p1:profile>
      <p1:profile offset="323" type="AV2_CURVATURE">564</p1:profile>
      <p1:profile offset="353" type="AV2_CURVATURE">564</p1:profile>
      <p1:profile offset="391" type="AV2_CURVATURE">570</p1:profile>
      <p1:profile offset="448" type="AV2_CURVATURE">570</p1:profile>
      <p1:profile offset="458" type="AV2_CURVATURE">570</p1:profile>
      <p1:profile offset="548" type="AV2_CURVATURE">439</p1:profile>
      <p1:profile offset="613" type="AV2_CURVATURE">583</p1:profile>
      <p1:profile offset="676" type="AV2_CURVATURE">593</p1:profile>
      <p1:profile offset="299" type="AV2_SLOPE_LIN" unit="percent">516</p1:profile>
```

```
    <p1:car age="2" confidence="0" heading="4294945208" index="103" lane="0" nTimestamp="0" offset="2664888" pathId="0"
2" vpStat="33554432"/>
  </p1:time>

  <p1:time nTimestamp="18">
    <p1:car age="405" confidence="0" heading="0" index="0" lane="7" nTimestamp="0" offset="57" pathId="72" probability="
    <p1:stub complexIntersection="0" formOfWay="2" functionalRoadClass="2" lastStubAtOffset="true" numberOfLanesInDrivin
oppositeDirection="0" offset="0" partOfCalculatedRoute="3" pathId="0" relativProbability="100" rightOfWay="2" subPathId=
lse">
```

Main Window



```
+    Timestamp:13
+    Timestamp:14
+    Timestamp:15
+    Timestamp:16
+    Timestamp:17
-    Timestamp:18

Stub Difference: At pathID 0 In label AV2_EFF_SPEED_LIMIT : profile in Copy but not in Reference [(18, 223, 70)]

Stub Difference: At pathID 0 In label AV2_EFF_SPEED_LIMIT : ORDER CHANGE DETECTED Unmatched Pairs in Copy are [(18, 223, 70

Stub Difference: At pathID 0 In label AV2_ROAD_CONDITION : profile in Copy but not in Reference [(18, 223, 8)]

Stub Difference: At pathID 0 In label AV2_ROAD_CONDITION : ORDER CHANGE DETECTED Unmatched Pairs in Copy are [(18, 223,

Stub Difference: At pathID 0 In label AV2_SLOPE_LIN : profile in Copy but not in Reference [(18, 223, 513)]

Stub Difference: At pathID 0 In label AV2_SLOPE_LIN : ORDER CHANGE DETECTED Unmatched Pairs in Copy are [(18, 223, 513)]

Stub Difference: At pathID 0 In label AV2_CURVATURE : profile in Copy but not in Reference [(18, 223, 542)]
```

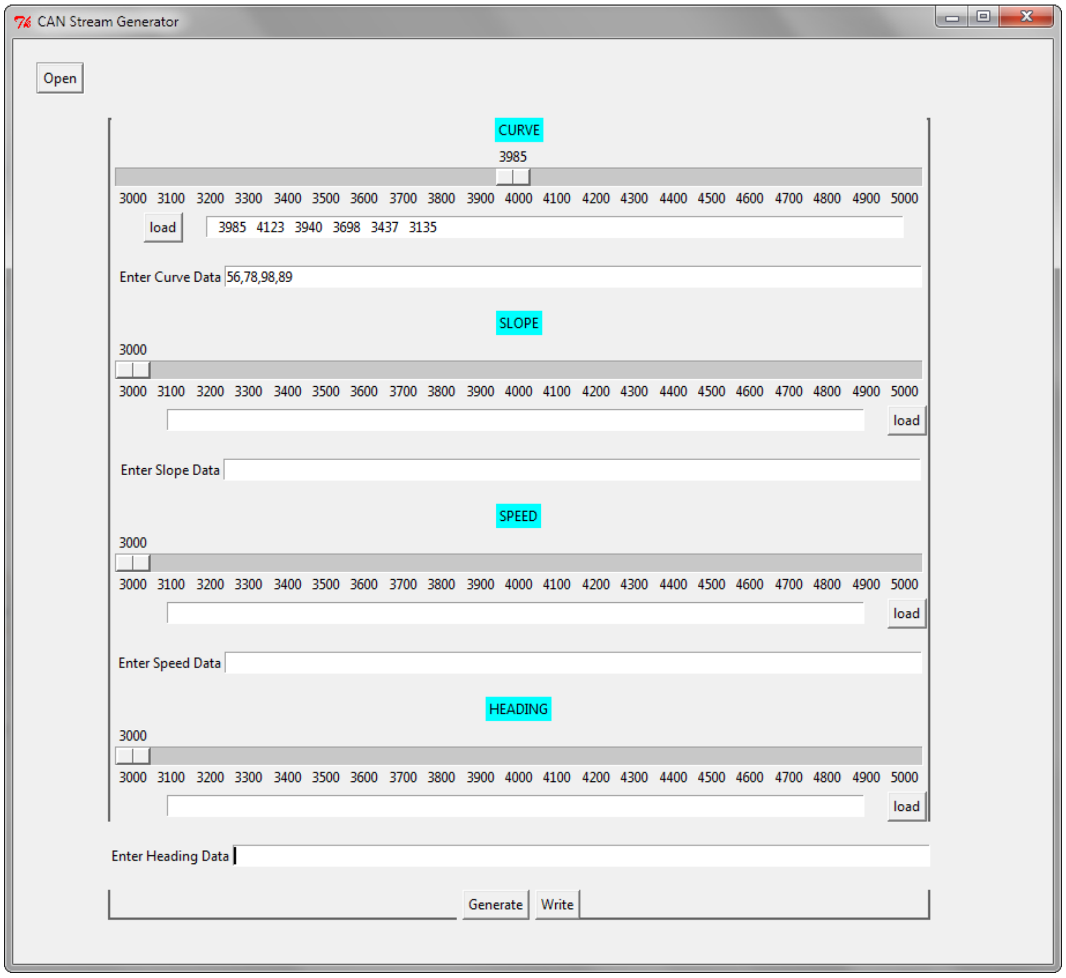
Comparing Window

Figure 7.3: Old XML Comparator Interface

Figure 7.1 presents latest interface design and Figure 7.3 shows the old interface that was used for comparing XML files. The main difference between two is the latest design has only one window and for better usability two text files are showed in vertical format than horizontal. This was done to match the display options of BeyondCompare tool.

Secondly, in previous format for every comparison a comparing window would pop out as shown in figure 7.3. It also had lots of text inside sometimes not easy to read. This was nullified and converted to shorter version in latest format. With highlighting feature there is no need of showing so much text.

7.3. CAN Stream Generator



3985 4123 3940 3698 3437 3135

Enter Curve Data 56,78,98,89

Enter Slope Data

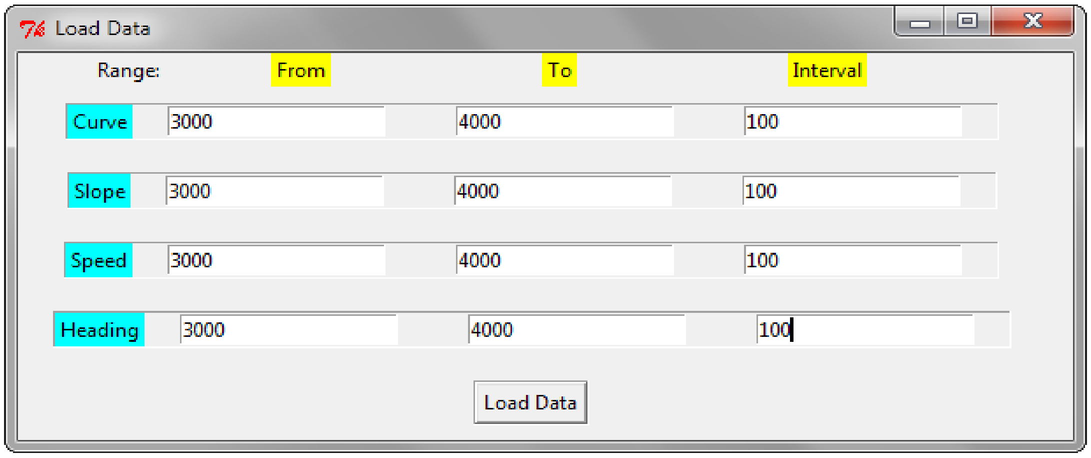
Enter Speed Data

Enter Heading Data

Figure 7.4: CAN Stream Generator

This is how the main window of CAN stream generation looks after getting all the values from the user. It can be observed from the figure that user can use slider over scale to move to different points and select the data points or he can also choose to use values directly using the entry label right below scale widget.

“Write” option located down is used to write values to another python file which is imported into script files. When user presses Generate, CAN stream generated which is shown in Figure 2.2.



Load Data

Range:	From	To	Interval
Curve	3000	4000	100
Slope	3000	4000	100
Speed	3000	4000	100
Heading	3000	4000	100

Figure 7.5: Loading data

Using this dialogue we can load the values of different parameters like curve, slope, speed and heading. Whichever parameters are filled here are only showed on the main screen. With this window one can set needed range with interval which will be then displayed on the slider.

8. Summary

The thesis have been concluded with covering three major topics which consist of Conclusion, Challenges and Limitations that were faced throughout the duration of work.

8.1. Conclusion

In this research main focus was put forward on building XML Comparison tool and CAN Stream Generator tool. Comparing of two XML files is done efficiently taking every scenario into consideration as discussed in implementation chapter.

Similarly the user interface for CAN Stream Generator is able to generate CAN stream just the way scripts used to perform. User interface that was built for both tools is robust and many custom user interface in our firm are built on same Python library Tkinter.

This document covers the concept that was developed to build both of these tools. It also backs research with working implementations and finally we have two software tools that are developed for further use.

8.2. Limitations

Despite of building working software tools, there are some areas which can be improved and some features that can be upgraded.

Initially with XML Comparator though the task of comparison is achieved, the user interface can be slightly improved by introducing a feature of auto spacing. This new feature will implement spaces in XML data which is shown in text box of interface. There can be ways in which data from file saved can be utilized in more efficient manner.

Secondly, in CAN Stream Generator, the functioning should not be only limited till generating CAN stream. But it can also be further developed to take such parameters in consideration which can simulate the data points for a scenario in which car is moving on a path. It can need more time but this feature can add cutting edge facility to the present tool.

9. Appendix

ADAS	Advance Driving Assistance System
ADASIS	Advance Driving Assistance System Interface Specification
AHP	ADASIS Horizon Provider
AHR	ADASIS Horizon Reconstructor
Av2HR	ADASIS Version 2 Horizon Reconstructor
WP	Work Packages
e-Horizon	Event Horizon
Path ID	Path Identifier
OSM	Open Street Map
Ref.xml	Reference.xml
Cop.xml	Copy.xml
MPP	Most Probable Path

Literature

[1] “ADASIS Report” Bosch Engineering GmbH – Internal, BEG-VS/EBI1, 2015

[2] Achim Haegele, “SWS Test Environment ADASIS”, Internal, Bosch Engineering GmbH BEG-VS/EBI1, 2013

[3] Sinisa Durekovic (NAVTEQ), “ADASIS v2 Protocol“, Bosch Engineering GmbH Internal, BEG-VS/EBI1, 2012

[4] Erik T. Ray, Learning XML, 2nd ed. O'Reilly Media, 2003

[5] Mark Lutz, David Ascher, 2nd ed. O'Reilly Media, 2000

[6] EXAMPLE OSM XML FILE http://wiki.openstreetmap.org/wiki/OSM_XML

[7] Haklay M, “OpenStreetMap: User-Generated Street Maps” IEEE, Univ. London, 2008

[8] Christopher A. Jones, Fred L. Drake Jr, Python and XML, 1st ed. O'Reilly Media, 2001

[9] Mark Pilgrim, Dive Into Python, http://www.diveintopython.net/toc/index.html

[10] John E. Grayson, Python and Tkinter Programming, 1st ed. United States, 2000

[11] Wilfried Voss, A Comprehensible Guide to Controller Area Network, Copperhill Technologies, United States, 2005