TECHNISCHE UNIVERSITÄT
CHEMNITZ

Visualization of Crash Channel Assignments in a Tabular Form

Master Thesis

Submitted in Fulfillment of the
Requirements for the Academic Degree
M.Sc. in Automotive Software Engineering

Dept. of Computer Science
Chair of Computer Engineering

Submitted by: Krishna Pooja Parthanarayanasingh
Student ID: 673220
Date: 21.04.2023

Supervising tutors: Prof. Dr. Dr. h. c. Wolfram Hardt, Dr. Ariane Heller
Supervisor from Robert Bosch GmbH: Marcus Lehmann

Acknowledgements

I would like to express my sincere gratitude to the people who have supported me in my journey of this thesis work. I’d like to thank Prof. Dr. Wolfram Hardt for providing me with the opportunity to do my master thesis in the Chair of Computer Engineering at Chemnitz University of Technology. I’d like to thank my supervisor, Dr. Ariane Heller for her support at all times. She always answered my questions in solving organizational issues at each milestone, and guided me with her feedback. I’d like to convey my utmost gratitude to my external supervisor, Mr. Marcus Lehmann for giving me the opportunity to do my thesis at Robert Bosch GmbH. I’d like to thank him for his trust in my abilities, valuable insights, and his constant support from the very beginning to the end. Last but not the least, I would like to express my heart-felt gratitude to my parents who made this path possible for me. I’d like to thank my brother for his support at all times. A big thank you to my family for their confidence in me and all the encouragement which has helped me throughout this journey. Many thanks to each of you for being there at every stage and making all of this possible.

Abstract

Passive safety systems try to lessen the effects of an accident. Airbags are a passive safety feature. They are designed to protect occupants of a vehicle during a crash. These systems have to be configured correctly in order to deploy airbags at the right time in case of a collision. Airbag application tools are used to simulate and interpret crashes. Some factors influence when an airbag should deploy. Based on different parameters, the logic for firing airbags is also different. Under every circumstance, an airbag has to be deployed at the right time in order to prevent injuries and fatalities. During the process of simulation, the data which is simulated is written to a database. During interpretation, this data is extracted from the database. Then, the required information can be analyzed and interpreted for further use. This data contains crash related information. For example, the type of crash, crash code and crash channel assignments. For every crash present in the airbag project, crash channels are assigned to the sensors. Each sensor present has a crash channel assigned to it. This is called the crash channel assignment. An airbag application tool is developed to show the crash channel assignments. This tool should handle the information extraction, and visualization of crash channel assignments. The final output should be in a tabular format, which includes user specific customizations.

Keywords: Passive Safety, Airbags, Application Tools, Crash Channel Assignments

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACC	Active Cruise Control	**HTML**	Hyper Text Markup Language
LDW	Lane Departure Warning	**CSV**	Comma Separated Values
ACU	Airbag Control Unit	**SWC**	Software Component
ECU	Electronic Control Unit	**BSS**	Block Starting Symbol
AB	Airbag	**LINQ**	Language Integrated Query
PAS	Peripheral Acceleration Sensor	**CAN**	Controller Area Network
UFS	Up Front Sensor	**ADAS**	Advanced Driver Assistance Systems
PPS	Peripheral Pressure Sensor	**DB**	Database
PCS	Pedestrian Contact Sensor	**VBA**	Visual Basic for Application
RCS	Rear Crash Sensor	**TSV**	Tab Separated Values
MS	Microsoft	**JSON**	JavaScript Object Notation
AI	Artificial Intelligence	**IDE**	Integrated Development Environment
AUTOSAR	Automotive Open System Architecture	**XSLT**	Extensible StyleSheet Language Transformations
RTE	Run Time Environment	**XSLT**	Structured Query Language
BSW	Basic Software	**CSS**	Cascading Style Sheets
CDD	Complex Device Drivers	**VS**	Visual Studio
XML	Extensible Markup Language		

1 Introduction

In this chapter, vehicle safety systems - active and passive safety systems are discussed in detail. Following that, a detailed overview regarding airbags and crash channel assignments is discussed in the background section. Furthermore, the motivation of the thesis is discussed in the next section. Then, the problem statement is explained. The last section of this chapter consists of the organization of chapters within the thesis, along with a brief overview of the contents of each chapter.

With the popularity of automobiles, the number of accidents increased. There were several reasons for this. As a lot of people started driving vehicles, they were not experienced enough to drive safely. Higher count of automobiles led to traffic. Due to roads which were not maintained, the number of accidents increased. Safety features were not available during the early days of popularity of automobiles. Hence, accidents resulted in serious injuries or even death. Overall, the increase in the use of automobiles led to an increase in the accidents due to lack of safety features. With the introduction of safety features in automobiles, the numbers of accidents have decreased. [1]

Vehicle safety systems were invented in order to address the risks associated with automobiles. There is a rising priority on passenger safety in automobiles. One of the earliest safety system introduced was seatbelts in the 1950s. Later, other safety features were developed to improve vehicle safety. For example, airbags, anti-lock braking systems, electronic stability control, and tire pressure monitoring systems.

These systems are important for several reasons. For example, active safety systems in a vehicle can improve vehicle stability and alert the driver to dangers on the road. These systems can help prevent accidents occur. In case of a car crash, vehicle safety systems can help protect occupants. Airbag systems and seatbelts help lower the risk of injury, and even prevent death. Passive safety systems can help reduce damage to a vehicle, and protect the driver and passengers after an accident. For example, fuel cutoff systems prevent fuel from continuing to flow during a crash, reducing the risk of a fire. [2]

Overall, vehicle safety systems improved the safety of vehicle occupants and in turn reduced the number of accidents, injuries, and deaths associated. They are essential for protecting occupants, preventing crashes, and minimizing damage.

Vehicle safety can be categorized into active safety and passive safety [3]. These terms relate to technologies which are made to avoid or lessen the effects of accidents. Active safety systems help in preventing an accident from happening. They work to avert collisions. Passive safety systems are safety measures which reduce the negative impacts of accidents. These systems in automobiles are for occupant safety.

Active Safety Systems	**Passive Safety Systems**
Anti-lock Braking System	Seatbelt
Traction Control System	Crumple Zones
Stability Control System	Airbags
Collision Avoidance System	Occupant Sensing System
Active Cruise Control	Headrests
Lane Departure Warning	Child Safety Seats

Table 1.1: Types of Vehicle Safety Systems

Some of the active safety systems are anti-lock brakes, traction control systems, stability control systems, and collision avoidance systems.

Anti-lock brakes don’t allow the wheels to lock, hence allowing the driver to have steering control. The braking power applied to each wheel is controlled with the use of sensors and control algorithms in the system.

Traction control systems use sensors to help maintain traction by limiting power to the wheels. Stability control systems use sensors to maintain and evaluate the stability of a vehicle. In case the vehicle begins to lose stability, brakes can be applied or power can be restricted to retain control. If a vehicle turns or enters a curve at high speed, it may skid sideways. The Electronic Stability Control technology automatically applies brakes on the individual wheels to prevent the vehicle from skidding. In turn, the vehicle is in control of the driver.

Collision avoidance systems use sensors like radar, lidar, and cameras. In order to avoid a collision, these systems identify objects in the path of a vehicle and issue warnings to the driver. These systems are also capable of taking action in order to mitigate a collision. For the active cruise control, a radar sensor uses the yaw rate and vehicle speed to determine the safe distance from the vehicle in the front. ACC automatically calculates the safe distance from the vehicle in the front. This helps to maintain safe speed, and distance from the vehicle in the front in some cases, thus providing safety. Sudden or abrupt changing of lanes can cause an accident. So, LDW systems are developed to give the driver a warning if a vehicle leaves the dedicated lane. Therefore, this helps to prevent an accident from happening.

Passive safety systems help in deploying safety measures appropriately in an event of a crash. These systems lower the risk of an injury or death. Hence, they are critical components of the design of a vehicle. Some of the passive safety systems are seatbelts, crumple zones, and airbags.

Seatbelts are mainly used to lessen the risk of injury or harm to a passenger in case of an accident. In order to lower the risk of damage, seatbelts use force limiters and pre-tensioners to absorb the impact energy upon collision. This helps to lower the risk of injuries to the passenger. The chance of the occupant being thrown out of the car upon collision is reduced.

Crumple zones are parts of the vehicle which are designed in order to absorb collision energy by crumpling in a controlled manner. As a result, the collision force which acts on the occupant is reduced. Crumple zones are generally positioned at the front and rear end of the car. They are designed in such a manner that the energy of the impact is absorbed, while the structure of the vehicle is maintained.

Airbags are meant to inflate quickly, providing cushion to occupants in the event of a collision. Airbags are classified into different types. Depending on the type of crash, the airbag is deployed. For example, frontal airbags are to be deployed in the event of a collision in the front, whereas side airbags are to be deployed in the event of a side impact collision. In case of a rollover or a side impact accident, curtain airbags are to be deployed from the roof in order to protect the head and upper torso of the occupant [4].

1.1 Background

In this section, the passive safety system - airbag is discussed in detail. This section is divided into two sub-sections. The first sub-section is about airbags. In this section, the evolution of the airbag system is discussed along with airbag deployment, and the sensors used in an airbag system. The next sub-section is regarding crash channel assignments.

1.1.1 Detailed Overview of Airbags

The first idea of an airbag was created by Bosch in the 1950s. In the 1970s, the technology started to be utilized in cars. The car which was the first with an AB for passengers, was the 1973 Oldsmobile Toronado. It offered a driver side airbag as an addition. In 1980, together with Daimler-Benz, Bosch introduced the world’s first Airbag Control Unit (ACU) to production vehicles. It was patented by Robert Bosch GmbH. The airbag technology was developed and improved upon by Bosch, and by the 1980s, airbags were amongst the common features in automobiles. The ACU is essential to passive safety in the case of a collision. It serves as the command center for the activation of various lifesaving systems, such as airbags. [5][6]

Airbags are now widely used. The different types of airbags are frontal, side, and curtain airbags, amongst others. They are a crucial safety component of automobiles. Airbags have proven to be extremely beneficial by lowering injuries and deaths in case of a collision. Today, they are a standard component in majority of automobiles. More than 90,000 lives have been saved by the airbag system.

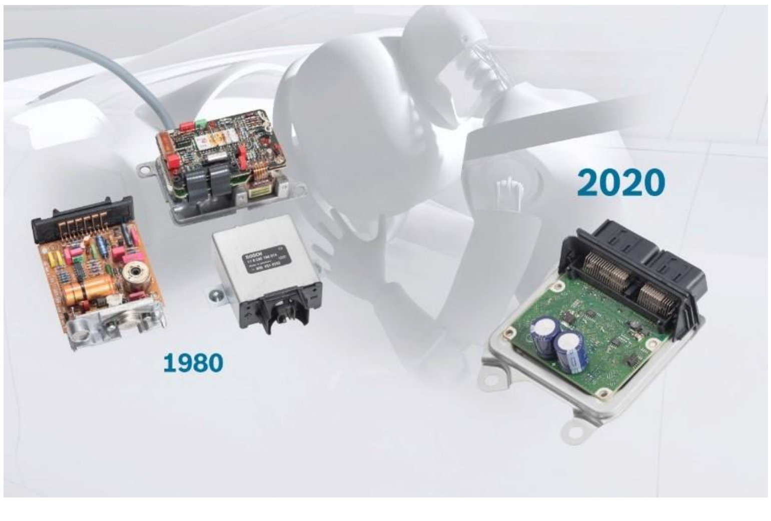
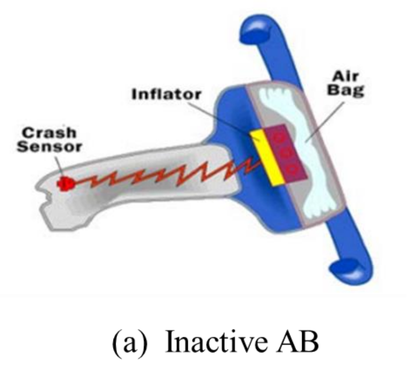


Figure 1.1: Evolution of the Airbag System [6]

The airbag is a passive safety device. It complements the active safety devices. The airbag is made of a flexible material like nylon. The main components in a classic airbag are propellant, inflator assembly, crash sensor, and an airbag. The propellant is a mixture of an oxidizer, along with sodium azide. The inflator assembly consists of a metal canister, filter, and an igniter. The crash sensor initiates the release of the propellant into the airbag. This process happens through an inflator assembly, which results in the inflation of an airbag. The airbag is a soft fabric bag made of woven nylon.

Figure 1.2 shows the parts of an airbag. In (a), an inactive airbag, where the airbag is deflated can be seen. In (b), an active airbag, where the airbag is inflated can be seen.



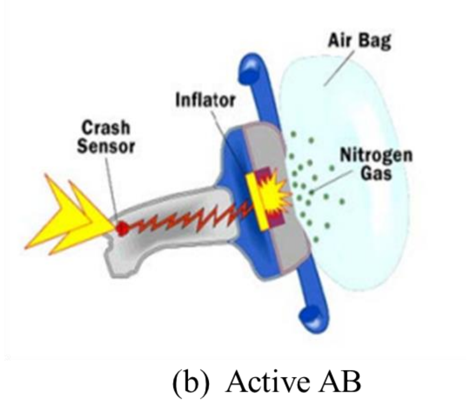


Figure 1.2: Parts of an Airbag [5]

The purpose of the system is to detect an abrupt change of speed in the vehicle. Because of this detection, the airbag can be deployed at the right time in case of a collision. The airbag deploys between the occupant of the vehicle and the interiors. The airbag ECU controls the airbag system.

Figure 1.3: Airbags in a Vehicle [6]

Airbags are a vital safety component in automobiles and help to keep passengers safe. When the airbag inflates, it acts as a cushion for the occupants. Airbags are meant to absorb the shock of a collision and help lower the risk of injuries to passengers. Airbags also help to protect the human body from sharp objects located within the interior of a car. They are basically designed to absorb the shock and reduce the chances of serious damage or death. For the airbags to function properly, seatbelts assist in keeping the occupant in the appropriate position during an accident. Depending on the type of collision and the car, several airbags may be deployed in different locations.

Figure 1.4 shows the position of different airbags in a car. For any standard vehicle model, the front airbag for driver and passenger in the vehicle is common. Depending on other factors like safety requirement and cost, additional airbags may be equipped. The kinds of airbags which can be seen in Figure 1.4 are frontal airbag, side airbag, curtain airbag, knee airbag, and pedestrian airbag.

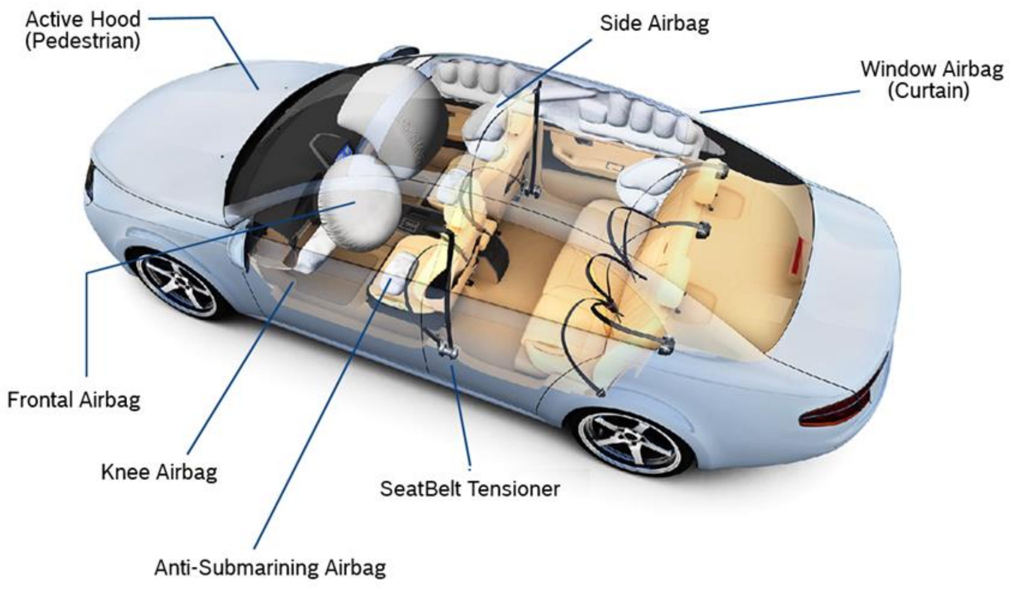


Figure 1.4: Different Types of Airbags in a Vehicle [6]

The rapid deceleration of a vehicle indicates a collision. When there is a collision, sensors are used to detect the impact and transmit the signal to the airbag control module. This process triggers a chemical process which rapidly inflates the airbag. The airbag is then deployed. There are different stages of airbag deployment. These stages can be seen in Figure 1.5, along with the time taken for each stage.

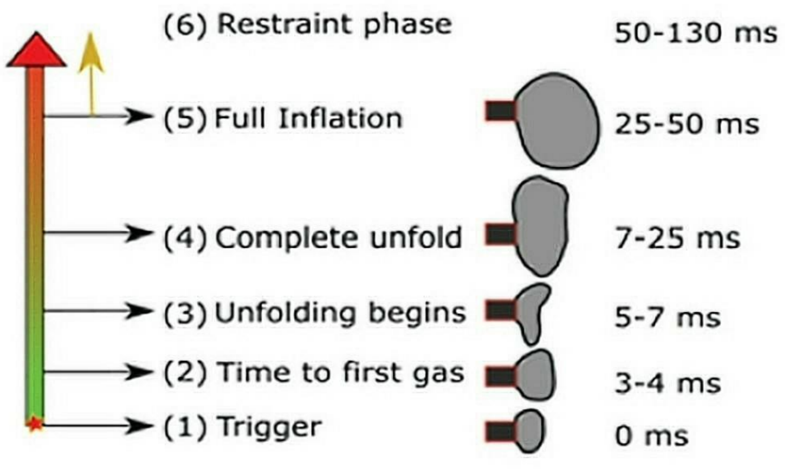


Figure 1.5: Airbag Deployment Phases [7]

The sensors use algorithms to determine when to deploy airbags. This depends on data received like position of the occupant, vehicle speed and impact angle. Sensors transmit a signal to the airbag control module. This sensor data is later analyzed to help decide whether or not to deploy airbags.

The deployment of an airbag is controlled by the airbag control module which is linked to a network of sensors throughout the vehicle. The airbag ECU, which can be seen in Figure 1.6, manages the functioning of the vehicles airbag system. The airbag ECU controls the functioning of airbags and seatbelts. It monitors the numerous sensors and switches needed to detect a collision, and decides whether or not to deploy airbags depending on this data.

The airbag ECU detects and evaluates crash signals depending on the type and severity of the crash. The type and severity of a crash is detected with the help of internal and external sensors. The ECU also detects situations where there can be false alarms. For example, if the road is bumpy or there are potholes, brakes are applied suddenly. This happens so that the safety system can be activated only at the right time. The airbag ECU is necessary as it ensures airbags are only deployed when it is needed.

Figure 1.6: Airbag ECU [6]

The airbag control unit evaluates the data obtained from the sensors to detect the type of crash. There may be different types of crashes. For example, front impact, side impact or rear impact. There are various safety features which are included, depending on the type of crash. Table 1.2 displays the type of safety features activated with respect to the detected crash type.

Crash Type	Safety Features
Front impact	- Single and multi-stage front ABs - Knee AB - Footwell AB - Belt pre-tensioner
Side impact	- Side and head ABs - Roll-over bar - Belt pre-tensioner
Rear impact	- Active headrest - Belt pre-tensioner
Roll-over	- Side and head ABs - Roll-over bar - Belt pre-tensioner

Table 1.2: Crash Types with their Safety Features

Airbag deployment is dependent on sensors. The information from the sensors is sent to the ECU. The data collected by sensors is used to make a decision to deploy airbags or not in the event of a collision.

Airbags are activated by sensors. These sensors monitor and operate a wide range of systems and operations in automobiles. Hence, they are essential. They are used to monitor various systems and subsystems.

There are internal and external sensors. Internal sensors are the sensors which are installed within the vehicle. External sensors are sensors which are placed on the exterior of a vehicle, and are used to monitor the environment around it.

There are a few common sensors used in an airbag system. Figure 1.7 shows the location of these sensors in a vehicle. They are Acceleration Sensor, Peripheral Acceleration Sensor (PAS), Upfront Sensors (UFS), Pressure Sensor, Peripheral Pressure Sensor (PPS), Pedestrian Contact Sensor (PCS), Rear Crash Sensor (RCS), and Central Sensors [8][9].

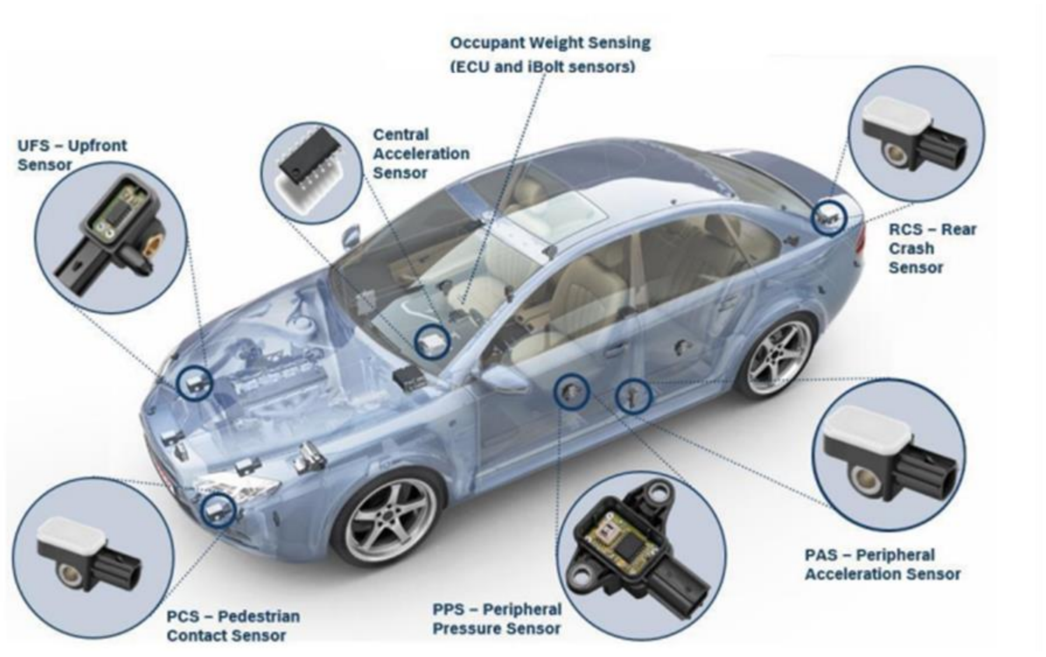

Occupant Weight Sensing
(ECU and iBolt sensors)
UFS – Upfront Sensor
Central Acceleration Sensor
RCS – Rear Crash Sensor
PAS – Peripheral Acceleration Sensor
PPS – Peripheral Pressure Sensor
PCS – Pedestrian Contact Sensor

Figure 1.7: Location of Sensors in a Vehicle [6]

Acceleration Sensors are sensors which assess the vehicles acceleration. They are generally installed in the front of the vehicle and can detect the severity of an impact. During a collision, these sensors measure the acceleration of a vehicle. This in turn helps to decide if airbags should be fired. Acceleration sensors help in detecting side, front and rear-end crashes. They are used in a number of applications like airbag deployment, stability control, and traction control.

Peripheral Acceleration Sensor (PAS) is used to detect abrupt changes in the acceleration of a vehicle. The acceleration of a vehicle is monitored continuously to know if the vehicle is getting into an accident. During a crash, the PAS measure accelerations and provides information on the direction and collision impact. There are different versions of the PAS which can be available. These different versions depend on the location of the sensor in the vehicle. PAS can also detect rear impacts. With the help of special acceleration sensors, it is possible to improve the detection of a front impact collision. Upfront Sensors (UFS) is a special variant of the PAS, which detects a front collision.

Pressure Sensors are used to determine the severity of an impact. This is done by detecting changes in pressure within the vehicle. These sensors also help to detect the location of impact. Pressure sensors help in detecting side crashes. Peripheral Pressure Sensor (PPS) is used to estimate pressure changes in the event of a side crash. It works in combination with acceleration sensors. PPS is usually installed into the door cavity as it provides the detection of a side impact collision. This sensor keeps a check on the changes through door deformation and pressure measurements. The sensor sends signals to the ECU. The decision to distinguish between a dangerous collision and a harmless impact is done in the Airbag ECU.

Pedestrian Contact Sensor (PCS) is a sensor used to detect pedestrian collisions. It works to protect pedestrians in case of a collision. These sensors are installed in the front of the vehicle for detecting pedestrians in front of the vehicle. It helps to detect and protect a pedestrian during an event of a crash.

Rear Crash Sensor (RCS) helps to detect a rear collision. These sensors are mounted in the vehicle's rear end. Seatbelt Sensors are used to detect whether a seatbelt is worn or not by an occupant. Depending on this, the optimum airbag deployment is determined.

Roll Rate Sensors are used to detect the rate of a vehicle roll. This is an indication for a vehicle rollover. If this is detected, then ABs are deployed to provide protection to the occupants. Central Sensors are also known as the ECU sensor cluster. These sensors detect motion in the front, side, rear, and z-direction of a vehicle, including roll rate.

This is a brief overview of the sensors used in airbag systems. Some airbag systems may also have other types of sensors. For example, sensors which are used to identify the position of passengers in a vehicle.

1.1.2 Crash Channel Assignments

In airbag control modules, crash channels are used to monitor the sensors which detect crashes or any impacts. The exact sensors which are assigned to each crash channel in an airbag control module are referred to as crash channel assignment information. The airbag control module uses this crash channel assignment information to specify which crash channel is assigned to the sensor. This is amongst the factors used to decide the deployment of airbags.

Based on the data from crash channels, within the ECU, the deployment algorithm is used to decide whether an airbag device should be fired or not. This helps to guarantee that the airbags are deployed only when it is necessary. This maximizes their effectiveness and lowers the risk of harm. Other factors like force of the hit, location of the car, and other data may also be considered.

The particular sensors that are attached to the crash channel are referred to as crash channel assignments. For example, one crash channel could be assigned to the front impact sensor, and another to the side impact sensor. The configuration of the airbag system will determine the crash channel allocations for a certain car. This differs based on the design of the airbag system and the unique requirements of the vehicle. Based on the number and type of sensors which are employed, some systems may have more crash channels in comparison to others. The kind of sensors might include rollover sensors, front impact sensors, and side impact sensors amongst others.

The airbag application tool used to show these crash channel assignments is called “AssignmentLister”. The airbag project contains sensors. Each of these sensors requires data for simulation. Therefore, crash channels are assigned to the sensors. Every sensor present in the project has a crash channel. Basically, this assignment is done for every crash which can be seen in Figure 1.8.

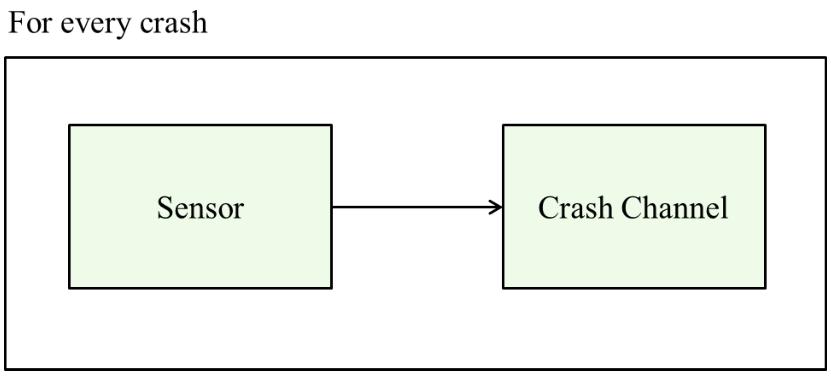


Figure 1.8: Overview of Crash Channel Assignments

1.2 Motivation

It is important to visualize the crash channel assignments in a tabular form. For this reason, an airbag application tool should be developed in order to show these assignments.

A concept should be created in order to get the crash channel assignment information. It is necessary to find a good-fitting file format to handle assignment information in the tool. The target is to transfer the information to another format, such that the output is in a tabular form. It should be as per requirements.

The existing solution using Excel Macros is slow and needs many manual steps. For this reason, the new solution should have a concept in order to visualize crash channel assignments without the use of macros. It should also have fewer manual steps. The tool developed should be able to handle the assignment information in an easier manner. The final output should be generated in a stable way. It is necessary to have a tool which functions in a fast, easy and stable manner with less manual steps. The new solution should mitigate the drawbacks of the existing solution.

To summarize this, the main steps needed to be performed are to get rid of Excel Macros and to have fewer manual steps in comparison to the existing solution. For this reason, a good fitting file format to handle the assignment information is necessary, and the output should be in a tabular format with the crash channel assignments.

The primary goal is to get rid of macros in order to visualize crash channel assignments in a tabular format with the tool. The primary goal is dependent on a few other steps which involves handling the assignment information and transferring it to a tabular format. All of the steps must be successfully performed in order to reach the main objective. The tool should also be integrated into the existing tool chain architecture.

1.3 Problem Statement

The existing solution is quite slow as it is using macros, and writing to Excel directly. The challenge is to find a good fitting format for handling data in the tool. The existing method to get the crash channel assignments is quite complex.

The output from the tool is not generated directly using the existing method. To get the final output, the user must perform many mouse clicks. For the output to be generated following the mouse clicks, the user should wait for a certain period of time. This method used to visualize crash channel assignments is cumbersome. It is also time consuming.

While waiting for the output to be generated, the clipboard cannot be used. If the clipboard is used, the output may not be generated as expected, and the system may even freeze. So, it may take more time than usual to generate the output or the page does not respond at all. Other MS Office Products also cannot be used during that time. Therefore, the tool is quite unstable, and does not work as expected. There are two different use cases available. There is no common solution for the use cases.

To summarize this, the existing problems are that the solution for the tool uses macros. Due to this, the tool is slow and unstable. The assignment information is not handled in an easy manner. Many manual steps are needed to visualize the crash channel assignments. While generating the output, MS Office Products cannot be used. There is no existing common solution for the use case scenarios available.

1.4 Organization of the Report

The thesis is structured in seven chapters. The first chapter consisted of a detailed overview of vehicle safety systems. Then, in the background section, airbags and crash channel assignments were discussed in detail. Following that, the motivation and problem statement of the thesis were discussed in separate sections.

In the second chapter, research papers related to the topic are discussed. Research work and recent technologies related to airbags, airbag deployment, and sensors are discussed.

In the third chapter, an introduction to the airbag application tool is given. Then, the state of the existing AssignmentLister tool is discussed in the following section. The purpose of this chapter is to analyze the old solution, and drawbacks of the existing tool.

In the fourth chapter, the concept developed to visualize crash channel assignments in a tabular form is discussed. The various methods to solving the problem are examined. Then, the chosen approach is discussed in detail, wherein it is broken down into three different parts.

In the fifth chapter, the implementation of the proposed concept is discussed. Different stages of implementation are explained in the sections of this chapter. In the fourth and fifth chapters, the alternative solutions to a particular problem are described, along with the reasoning behind the chosen approach for the solution.

In the sixth chapter, the results and evaluation are described. The crash channel assignments in a tabular form are shown and explained, following which the new solution is evaluated against the old solution.

In the seventh chapter, the conclusion and future scope are described. A summary of the main points of the thesis from earlier chapters are discussed. The conclusion along with ideas for future work is presented in this chapter.

2 State - of - the - Art

In this chapter, recent and ongoing work, technologies, and methodologies related to the topic are discussed. This chapter consists of a section about recent development in airbags. Following that, there is a section about recent and ongoing work in the field. Literature papers are researched upon, and an overview of the findings is discussed. This chapter ends with the summary section.

2.1 Airbags

Airbags reduce the risk of injury to occupants during an accident. This field has progressed in recent years. This has led to improvements in the airbag design and functionality. As a result, airbags are more effective.

Over the years, the development of airbags has been a gradual process of research and innovation. The earliest airbags were very simple designs. They deployed with a single-stage inflation. ABs have evolved into more advanced systems which can adjust the deployment based on the severity of the collision, size and posture of the occupant, and other factors. The history of the development of airbags over the years is discussed further.

During the 1960s -1970s, the earliest airbag designs were developed. These airbags used a single-stage inflation system. The airbag used to inflate quickly, and with a lot of force during a collision. These airbags were not so effective. They were not widely used because of their poorly functional airbag systems. High costs were also amongst the factors. In the 1980s, car manufacturers began to use airbag systems which used sensors to determine the severity of a collision, and adjust airbag deployment accordingly. In the 1990s, airbag technology improved significantly. There was introduction of seatbelt airbags, inflatable seatbelts, and dual-stage inflation technology which allowed for better protection of occupants during an accident. In the 2000s, pedestrian airbags were developed with the aim to protect pedestrians during a collision. During the 2010s, the adaptive and delayed deployment technology became more widespread. This helped to improve occupant safety. In the 2020s, the most recent development in airbags was introduced. This development was the active venting, which adds to the protection of passengers. Technologies involving artificial intelligence and machine learning are also used for airbag deployment, which improves the accuracy of the AB system.

From the 1960s, airbag technology has seen various improvements. Airbags are more advanced now, and they provide higher protection to occupants.

Various kinds of airbags are used in vehicles. For example, frontal airbags, side airbags, and curtain airbags. There are some advanced airbag technologies. Some of the advancements over the years are seatbelt airbags, multi-stage airbags, pedestrian airbags, active steering wheel airbags, and head airbags.

Seatbelt airbags are integrated into the seatbelt system. The seatbelts contain a small airbag as shown in Figure 2.1. During an accident or collision, there is a lot of force exerted on the occupant. These airbags inflate in order to help distribute the force of a collision over a larger area of the body. This aids to lessen injuries, particularly to the chest and torso region during an accident. [10][11]

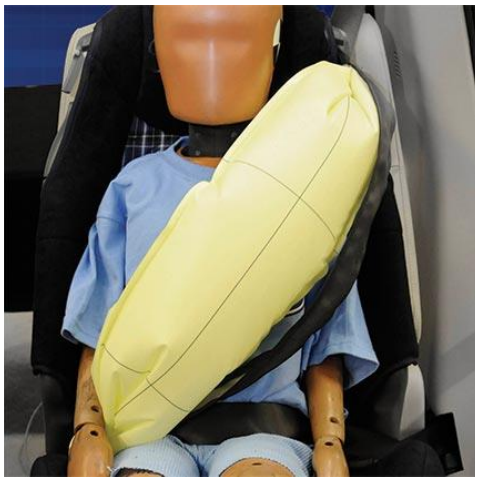


Figure 2.1: Example of a Seatbelt Airbag [6]

Multi-stage airbags are airbags which use sensors in order to get to know the severity of a crash. They adjust the airbag deployment accordingly. Due to this, the risk of injury to passengers in the event of a minor collision is reduced. Airbags are deployed with less force in a minor collision. In more severe collisions, airbags are deployed with more force. This helps to provide maximum protection to passengers.

Pedestrian airbags are located in the front of the vehicle like in Figure 2.2. These airbags deploy from the base of the windshield when a vehicle clashes with a pedestrian. These airbags provide a cushioned surface to help reduce the risk of head injuries to the pedestrian. In the event of a collision, the pedestrian would most probably collide in a particular region on the vehicle. This region is protected by airbags. Hence, the severity of injuries is reduced because of the pedestrian airbag.

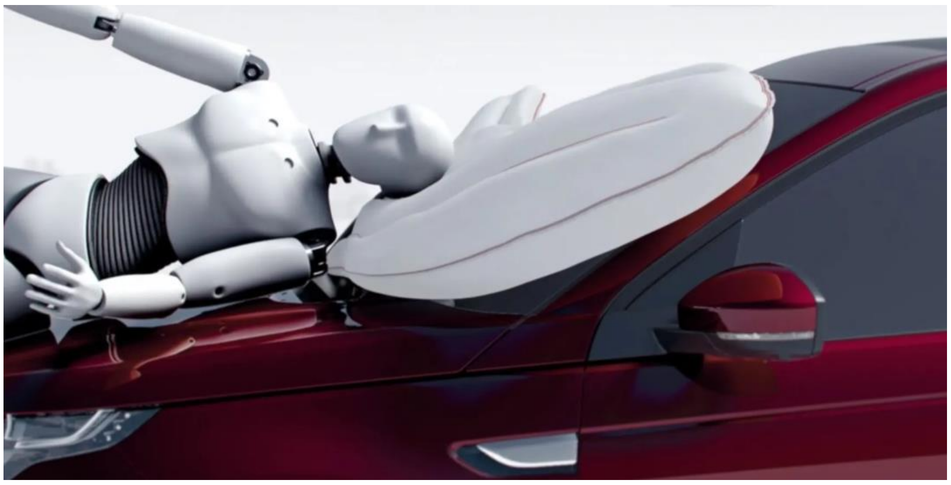


Figure 2.2: Pedestrian Protection Airbag [12]

Active steering wheel airbags detect the location of the driver's hands on the wheel of the car using sensors. The deployment of airbag is adjusted according to this position. These airbags help to lower the risk of harm to the driver’s hands and arms in the event of a crash. Hence, the airbag system is more effective. Head airbags deploy from the roof or the seat, and provide additional cushioning in order to reduce injuries. Non-pyrotechnic airbags use compressed gas or foam to deploy the airbag instead of explosive pyrotechnic materials. This lessens the chances of any injury from AB deployment. They also help to reduce the environmental impact of the airbag system. These are some of the improvements in the type of airbags used in vehicles for occupant protection.

During a collision, airbags are designed to deploy quickly and with some force to protect passengers. Advancements have made airbag deployment more controlled and precise. Some of the notable improvements in airbag deployment are discussed further.

Dual-stage deployment is a technology which deploys the airbag in two stages. The first stage is the deployment of airbags with a lower force. The following stage is the deployment of airbags with a higher force if necessary. For example, if a lower force is enough to protect the occupants during a crash, then the second stage of deployment doesn’t happen. Multi-stage airbags are used for this purpose.

Adaptive deployment is a technology which uses sensors to determine the position and size of occupants in a vehicle. The airbag deployment is adjusted accordingly. This kind of deployment helps in distinguishing passengers of different size and hence, their position in a vehicle. This in turn helps to protect passengers in a better way. For example, if a child is in the front seat of a vehicle, the airbag may be deployed with less force to protect the child from injury.

Delayed deployment is a technology which delays airbag deployment during an accident. After a crash, the airbag is deployed a few milliseconds later. The purpose of this kind of deployment is to allow the occupant’s body to move into a safer position before the inflation of the airbag. This in turn helps to minimize the probability of an injury.

Active venting is the most recent technology which uses vents in the airbag. This helps to control the rate of inflation, and release gas from the airbag quickly. In case of a collision, there is a forward movement or jerk which the passengers experience. The flexible vent allows the driver’s forward momentum to push out the air. This working principle of this process can be seen in Figure 2.3. The risk of injury to passengers from the airbag itself is reduced. The amount of dust or debris in the vehicle in an accident may also be reduced because of this technology. This can improve visibility for the driver in the vehicle. As a result, having flexible venting airbags can protect and save passengers' lives in the scenario of an accident.

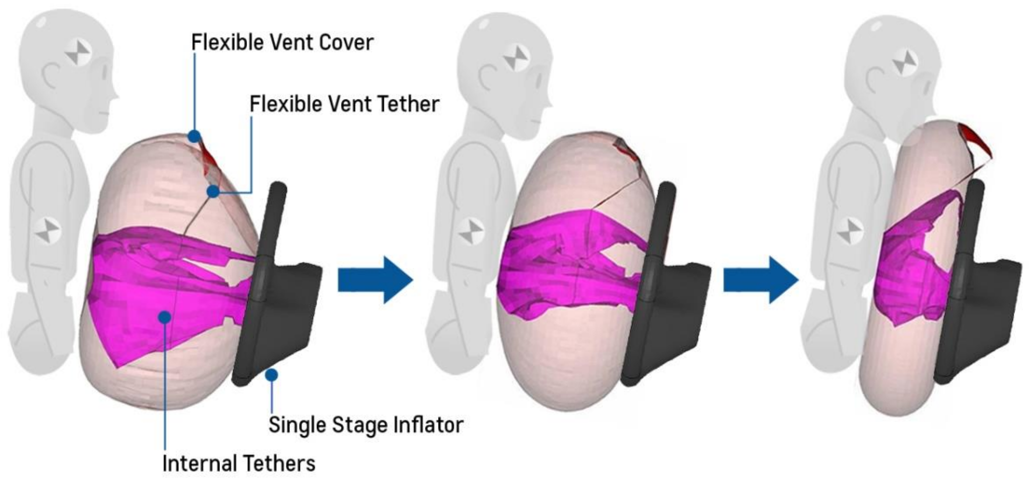


Figure 2.3: Working Principle of the Flexible Venting Airbag [13]

Smart airbags use sensors to detect the position of the occupant. Many factors influence airbag deployment. Smart airbags help minimize the risk of injury to passengers, considering all these factors. For example, smart airbags can detect whether the passenger is wearing a seatbelt or not, and then adjust airbag deployment accordingly. The improvements in airbag deployment control help to make airbags even safer in the event of an accident.

Technologies with artificial intelligence and machine learning enhance airbag deployment. Information from the sensors is first analyzed. Then, the way in which the airbag can be deployed is determined. For this purpose, factors like speed of the vehicle, position of occupants, and the severity of the collision are taken into consideration. Using advanced technologies like this helps to reduce the risk of injury to passengers, and also unnecessary airbag deployment.

Advanced materials are used to improve the protection from airbags. Airbag manufacturers are developing new materials to make airbags smaller, lighter, and more effective. For example, some manufacturers use graphene, which is a light-weight, flexible and robust material. They are used to reduce the size and weight of the airbag system, hence making airbags more effective to protect occupants. Other airbag materials being researched include carbon nanotubes.

An important trend in the automotive sector has become sustainability. Airbag manufacturers are working towards reducing the environmental impact of airbag systems. Airbags which use eco-friendly materials are being developed, along with designing airbags which can be easily recycled or disposed. The use of renewable energy sources, like solar power to deploy airbags is also being explored by some manufacturers. The latest trends in airbags and airbag systems are focused on improving airbag effectiveness. The size, weight, cost, and sustainability factors are also being considered.

The key component which manages the deployment of airbags and other safety features is the ECU. Some of the latest trends with airbag system ECUs are discussed further.

Advanced sensors and detectors are used in modern airbag systems to detect a collision and assess the severity of a crash. This allows the ECU to determine airbag deployment accordingly to protect passengers. Airbag systems have integrated safety features [14]. One such safety feature is the seatbelt pre-tensioner. A double pre-tensioner is mainly used in the seatbelt system. There is some space between the passenger and seatbelt. During a crash, the body of the passenger slides rapidly forward. The seatbelt restrains the passenger from moving further. When this happens, the passenger’s body is compressed against the seatbelt. A large force is exerted on the passenger.

The seatbelt pre-tensioner helps to reduce the impact on the passenger. Hence, the seatbelt works more effectively. The tightening and latching process of the seatbelt is controlled properly, in a timely manner. This ensures that the occupant in the vehicle is seated properly. It reduces the injury that may be caused to the occupant. A load limiter is used together with the pre-tensioner. It limits the amount of force which is transmitted through the seatbelt in the event of an accident. It assists in the controlled release of seatbelt tension. As an outcome, the risk of injury is lowered further as the energy of the collision is absorbed.

The double pre-tensioner and load limiter are safety features which work together. The working of this can be seen in Figure 2.4. They help in improving occupant safety, hence preventing serious injury or even death. These safety features are managed by the ECU. They provide a more reliable safety system for the vehicle.

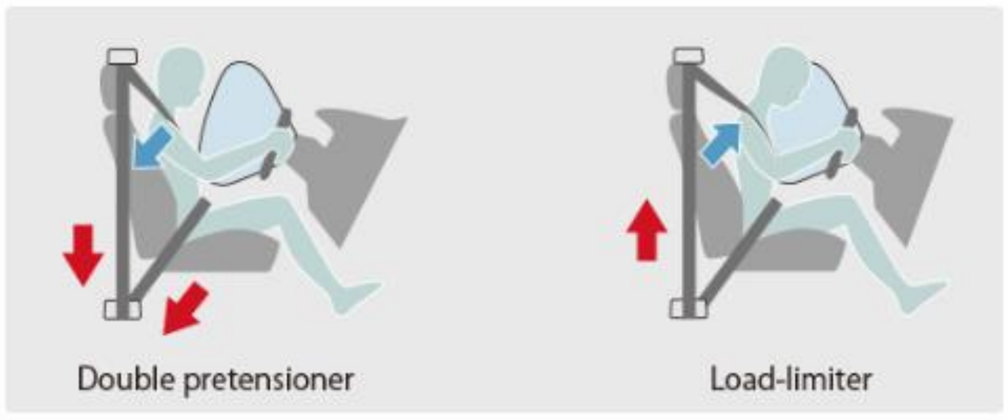


Figure 2.4: Seatbelt Pre-Tensioner [6]

Artificial intelligence integration is the use of AI to improve the accuracy of the ECUs crash detection and airbag deployment algorithms [15]. This allows the system to adapt and improve over a certain period of time, hence providing better protection for vehicle occupants. These advancements are focused on enhancing the airbag safety feature for vehicles. Therefore, the risk of injuries to occupants in the vehicle is reduced.

Airbag sensors are important components in airbag systems. They detect a collision and trigger the deployment of airbags. There have been some advancements and ongoing research in the field of airbag sensors. Some of them are multi-axis sensors, inflatable sensors, and optical sensors. [16][17]

Multi-axis sensors are designed to detect impacts from multiple directions. This helps the airbag system to detect collisions from any direction. Hence, the system responds more quickly in the event of an accident, and is more effective. Inflatable sensors use a flexible, inflatable material which can detect a collision and trigger airbag deployment. They are more responsive than traditional sensors. Hence, they provide better protection for occupants. Optical Sensors use light to monitor changes in the vehicle's surroundings. For example, optical sensors can detect any obstacle ahead of a vehicle. They can help provide more precise information about the surroundings of a vehicle.

Over the years, airbags and airbag systems have evolved. They have become more advanced with more features for protection. The latest trends are mainly focused on improving airbag functionality. These advancements to enhance the airbag system have provided more protection to occupants in the case of an accident. As a result, the probability of injury or even death to passengers is minimized.

2.2 Recent Developments

Recent developments and ongoing research in the automotive field has resulted in significant improvements in various areas. The advancements in the automotive field include the development of ECUs, implementation of AUTOSAR, and scientific development on airbag and airbag systems amongst others. Together, these developments have created more advanced, efficient, and safer vehicles.

In this section, some of the recent and ongoing research conducted by the Professorship for Computer Engineering at Chemnitz University of Technology is discussed in detail. Following that, a few research papers from other publications are briefly discussed.

AUTOSAR is a standardized architecture for the development of software. The introduction of AUTOSAR has made the development of automotive software platform independent. It is composed of three layers, the Application Software, Run-Time Environment (RTE), and Basic Software (BSW). [18]

There are numerous components that interact with one another. This complicates the integration of software on the ECU. There are some errors which can happen while integrating AUTOSAR software on an ECU. These errors occur as there are a large number of modules involved in an automotive ECU. They may also happen due to improper timing. AUTOSAR has a layered architecture. Because of this, the errors complicate the test and verification process.

Testing automotive software is challenging. Issues are most likely to happen during the integration phase. Some of the main challenges to test automotive software are their complexity, safety concerns, and inter-connectedness.

There are numerous components. These components are interconnected. Hence, it’s difficult to verify their behavior. Automotive systems also need to function under different environmental conditions. This further increases their complexity. Errors in the automotive systems may lead to accidents, or even death. These systems require proper testing to ensure that the system behaves as expected under all conditions. Hence, safety is a critical issue. Automotive systems are highly connected amongst each other. Hence, testing is important to ensure that all components function correctly independently, and in combination with other components. Integration phase is the phase where all software components are integrated on the ECU. Testing is required to confirm that the system is functioning properly.

In this research paper [19], a method is proposed in order to check the functionality and timing of AUTOSAR modules. For this purpose, a semi-automated method is used, where the test cases are based on interface descriptions. This technique helps to identify the affected modules in a reliable manner. Hence, the test and verification process is simplified.

An overview of the proposed method can be seen in Figure 2.5. The proposed technique provides a solution for testing AUTOSAR software modules. Test cases are generated based on interface descriptions of the application and BSW modules. For an external test program, a configuration file is generated. It communicates with the test engine, and controls the test execution on the ECU. This particular method can be implemented using standard development tools. This proposed approach to test the BSW covers all modules, except the Complex Device Driver (CDD). This is because of its hardware dependency. In order to test CDD modules, a virtual integration method is proposed. This uses hardware driver simulation on a desktop computer. The main focus in this research paper is on the testing of inter-ECU communication between the application and BSW layers.

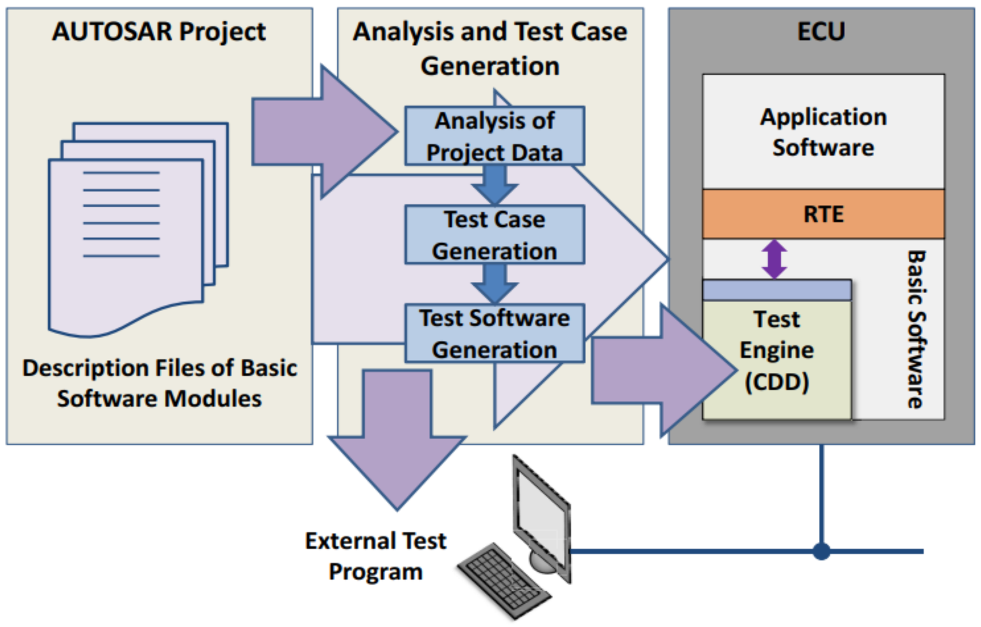


Figure 2.5: Method to Test AUTOSAR Software Modules [19]

This process is divided into three phases, which are the analysis of project data, test case generation, and test software generation.

In the first phase, which is the analysis of project data, the necessary data is extracted from the AUTOSAR project. This project contains the application software. The data extracted consists of information from the BSW modules, their configurations, and interfaces. This data is extracted from the configuration files using an XML parser [20]. The data is then processed. As a result of the processing, the input and output signals necessary for the test cases are generated.

In the second phase, which is the test case generation, test cases are generated based on the information got in the first phase. The test cases are designed such that individual and combined modules can be checked sequentially. This is done so that the functional and timing properties can be checked correctly.

In the third phase, which is the test software generation, the test engine is generated using pre-defined templates. These templates are in C code. This is then integrated into the BSW as the CDD module. The test engine contains a program with the tests to be performed. This needs to be synchronized with the external test program. The external program controls the test process and evaluates results. The result can be in these formats – tabular HTML pages or CSV files. The process of test software generation can be seen in Figure 2.6.

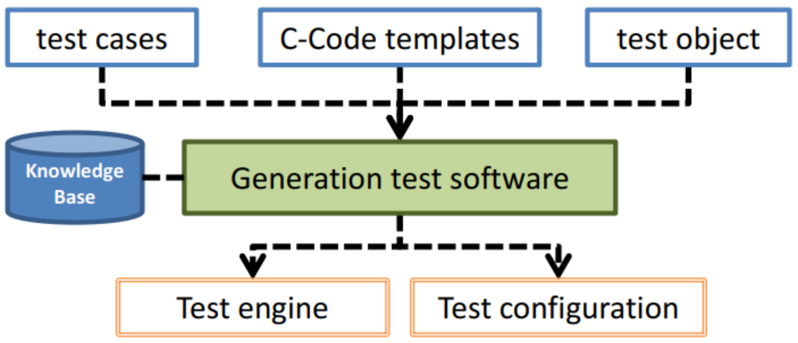


Figure 2.6: Workflow of the Generation of Test Software [19]

This approach is semi-automated. This is because it requires manual effort in extracting the necessary data and synchronizing the test sequence. Since it is automated to a very good extent, faster integration and testing of AUTOSAR software on an ECU can be done. It has a wide range of use-cases from testing BSW modules to integrating software components. Since the creation of test cases is automated, this tool is valuable for developers working with AUTOSAR software.

Airbag control systems require accurate timing and reliable functionality. This technique used for testing and verifying software on an ECU can be relevant for the development and integration of airbag control systems.

There are several approaches for prototyping. These include Model-in-Loop, Hardware-in-Loop, and Processor-in-Loop. In this research paper [21], a concept is presented for automating the process of rapid prototyping of AUTOSAR-based systems. The proposed method uses AUTOSAR tools to automate the process of generating system architecture, and modules which are specific to the application provided. Non-AUTOSAR compliant code is integrated into an AUTOSAR-compliant SWC. This process involves parsing the code in order to identify its structural and behavioral aspects. Then, it is modified to fit into the specifications of AUTOSAR.

So, a tool was implemented based on this concept. This tool performed the automated process. This was then used to generate an AUTOSAR system for a sensors fusion algorithm. The generated code was checked for errors. Two methods were used for the evaluation, and memory usage between them was compared. The comparison between these two methods can be seen in Figure 2.7.

- Method 1: 🟦 (Manual implementation by altering and adjusting code)
- Method 2: 🟩 (Proposed concept using the tool)

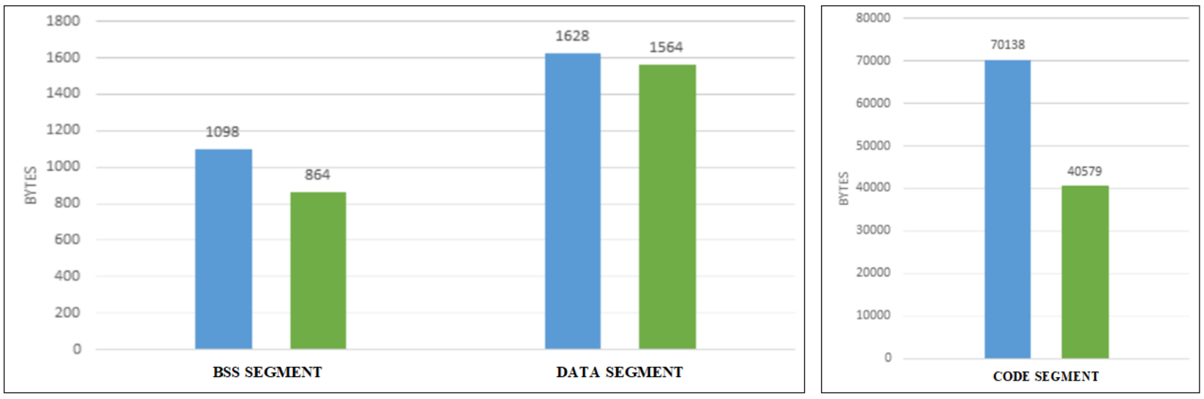


Figure 2.7: Comparison of Memory Usage of the Two Methods [21]

Compared to the manual adaptation of the code, the proposed concept with the automated method was able to save more space in all the three memory segments - BSS, data, and code segments. The conclusion from this was that the automated concept saved more space. It also helped reduce time and effort. This concept enables running a prototype code on a real-world ECU. Hence, this method is effective for the rapid prototyping of AUTOSAR systems.

AUTOSAR provides a standard framework for software development. This can help develop software components for airbag systems. The use of automated tools and processes, like the tool presented in this research paper, can help the development and testing process. With the use of a standard framework and automated tools, it is possible to ensure that SWCs for airbag systems follow the specified industry standards and regulations. Hence, the risk of errors is reduced, and safety is improved. Overall, it can help speed up the process of development and testing of airbag systems.

This research paper [22] focuses on the significance of storing structured data. This is mainly to support the development process for AUTOSAR projects. The development of parsers to extract structured data from various sources is described. This is done in order to fill a database for the AUTOSAR knowledge base.

Two methods are explained to parse data in this paper. They are tresos insallation parser and project parser. The tresos parser automatically fills the AUTOSAR knowledge base. The tresos installation parser extracts structured information from its installation files of existing

AUTOSAR versions. These existing items are updated, and not overwritten. The project parser enables the analysis of AUTOSAR projects. The project parser extracts information from an AUTOSAR project. This project consists of a subset of information from the tresos installation files.

The implementation details of both the parsers are described. They are written in C# and use the .NET Framework to parse the data. LINQ to XML is used to parse the configuration XML files. LINQ to XML is a set of .NET Framework libraries, using which data can be extracted from the XML files. The parsers enhance the development process for ASTAS (Automotive Software Test and Analysis Systems) test modules. ASTAS are software tools which are used to test and analyze automotive software. The parsers provide valuable support for data analysis for the development of ASTAS test modules. The parsers can be used for analyzing and testing software in various automotive systems, such as airbag and airbag systems. For example, the parsers can be used to identify if there are any dependencies between software modules in an airbag system. Then, this could be analyzed to ensure the system works correctly.

This research paper [23] discusses the development of a Multi-ECU demonstrator platform. It is called YellowCar.

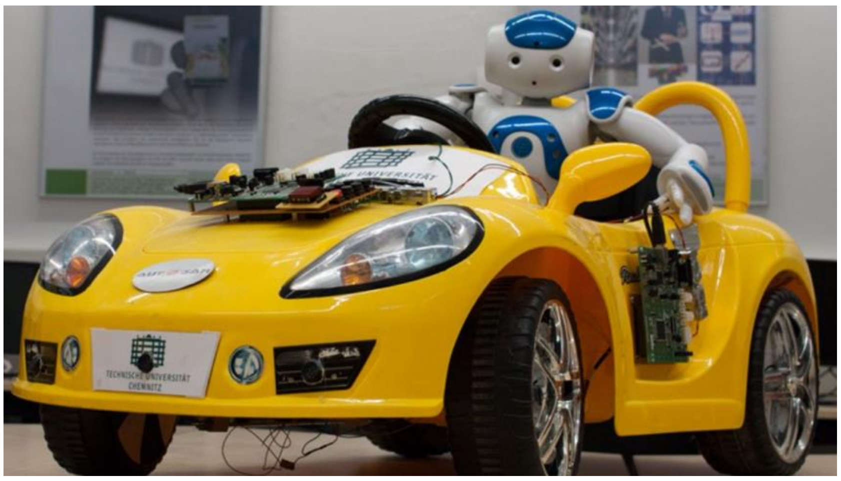


Figure 2.8: Multi-ECU Demonstrator Platform – YellowCar [24]

It is based on an automobile model. It consists of a network with three integrated ECUs based on AUTOSAR standards. The three integrated ECUs are Assistant, Feature, and Processing ECU. The front view of YellowCar with ECUs can be seen in Figure 2.9. The ECUs have different functions to perform. The functionality of the ECUs and server is depicted in Table 2.1.

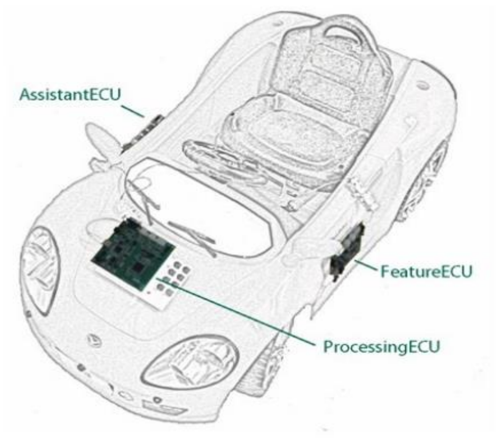


Figure 2.9: Front View of YellowCar with ECUs [23]

ECU and Server	Functionality
Assistant ECU	Engine, Steering
Feature ECU	Light Control
Processing ECU	Reading Sensor Data
Server	Remote Control, Monitoring

Table 2.1: ECUs and Server with their Functions

These ECUs are connected via a CAN communication system. The platform consists of many sensors, an electric motor, a steering unit, and lights. The sensors include ultrasonic sensors and cameras. All the sensors are connected to a single ECU. This ECU acts as a sensor ECU node. This sensor data can be accessed by other ECUs by reading the signals from the communication bus. This architecture supports sensor data fusion. The YellowCar is a platform for functional testing, performance evaluation, and optimization of software architecture.

A server, written in C++ connects to a network via the communication bus. A webpage is provided by the server. This webpage shows sensor and control data values. The server also provides data to C# software that simulates the tool in 3D. Based on the delivered CAN messages, this depicts the behavior of a real car. This system enables remote controlling and monitoring of the YellowCar in real time. This can be seen in Figure 2.10.

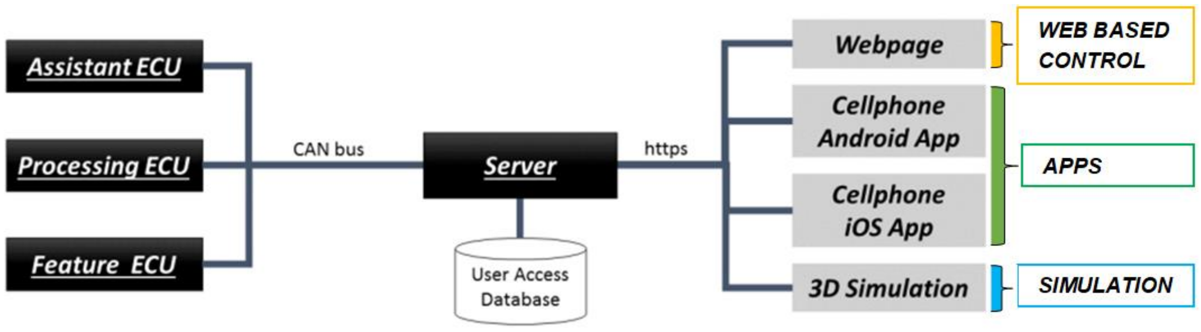


Figure 2.10: Web Based Control, Apps and 3D Simulation of YellowCar [23]

YellowCar provides a real-world demonstrator for testing and validating applications before implementing them on actual cars. Many applications have successfully demonstrated the suitability of the platform. YellowCar has implemented applications for light control, traffic sign detection, and obstacle detection successfully.

This open concept helps to easily add new functionalities and additional ECUs. It also offers a secure setting for evaluating automotive applications. The development of the platform, its architecture, and potential applications in the automotive industry are discussed in this research paper. On-going research includes evaluating approaches for test case generation, analysis of AUTOSAR configurations and source code. The platform allows researchers to experiment with new automotive technologies. YellowCar is an important tool for industry research in the automotive field.

The YellowCar platform can be used to test and validate airbag systems in the automotive industry. The effectiveness of various airbag designs can be checked under different conditions. These conditions may be the type of collision, impact angle or vehicle speed. YellowCar supports the integration of new ECUs and functionalities. This, along with its modular design makes it a possible platform for developing and testing airbag systems. Other safety critical applications in the automotive industry may also be developed and tested.

An extension of YellowCar is BlackPearl. It is an extended automotive multi-ECU demonstrator platform. This research paper [25] presents the development of the BlackPearl. It consists of separate boards for sensor units, image processing units, and display modules. All of these are interconnected by a CAN bus. The platform supports up to three ECUs and additionally, three image processing units. It is designed to meet ADAS needs, which rely on image-based sensors for pre-processing, feature detection and recognition.

The platform has been used to implement image processing applications like High Way Traffic Analysis successfully. The platform may be used for testing and validating a range of applications, including airbag systems [26]. It can be equipped with sensors and image processing units. This can help evaluate the performance of airbag systems under various scenarios.

This aim of this research paper [27] is to increase the reliability of airbag deployment in automobiles. There are some problems with airbag deployment. For example, even when there is no accident, an airbag may deploy due to a strong impact on the sensors in the ACU or while driving in off-road conditions. This research aims to mitigate these limitations. This is done by creating an algorithm which predicts a collision in a precise manner. A pre-crash algorithm is proposed, which estimates the behavior of objects before a collision. This information is then communicated to the crash algorithm for correct recognition of crash scenarios. This algorithm consists of information about a collision before it happens, and a crash algorithm based on crash-related sensors. Simulations using crash test data showed the pre-crash algorithm to be more dependable than the crash algorithm alone [28]. Figure 2.11 shows the overall crash algorithm, along with the estimation of pre-crash information.

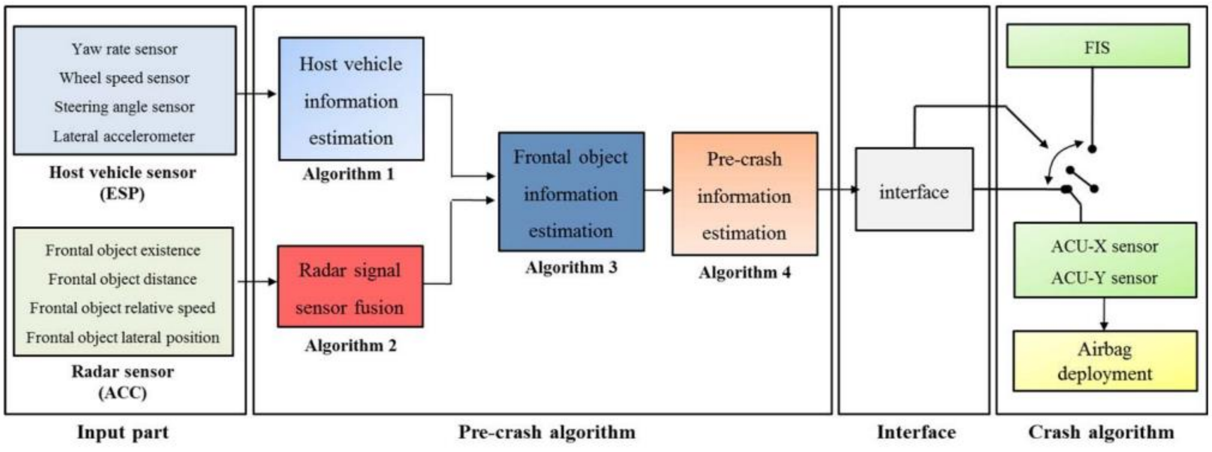


Figure 2.11: Crash Algorithm Based on Pre-Crash Information [27]

An airbag must always deploy at the appropriate time. When an airbag is deployed with extreme force, the risk of damage increases. If an airbag is not deployed at the right time during a crash, occupants in the vehicle may suffer critical injuries or even death. Hence, it is important to detect when the airbag is to be deployed properly. [29]

The automotive industry is constantly advancing with the latest trends and developments in vehicle technology. AUTOSAR has improved software development processes. Due to advancements in the field, ECUs are more efficient. Crash information and pre-crash information systems have become a focus area for vehicle safety.

2.3 Summary

Airbags are required because they give an extra layer of safety to passengers during a collision. Airbags and airbag systems have evolved over the years. The latest trends are focused on improving their effectiveness. Hence, the risk of injuries is reduced, or even death is prevented. Airbag technology improvements include enhanced airbag deployment systems, airbag sensors, and new materials for ABs. These advancements were discussed in Section 2.1.

In Section 2.2, various trends in automotive development were discussed. Recent and on-going research conducted by the Professorship for Computer Engineering at Chemnitz University of Technology was discussed. Along with that, research papers from other publications were also discussed. The latest trends in AUTOSAR are improving communication between ECUs, and providing more efficient and flexible software solutions. These trends will help the automotive industry meet its growing needs. More efficient ECUs have improved vehicle performance. Crash algorithms help to mitigate the impact of collisions. They provide vital information. To prevent the negative effects of wrong airbag deployment, it is critical that the airbag is deployed at the correct time. Airbag systems are evolving continuously to provide better protection to occupants and save lives.

Furthermore, continued research and development in airbag technology, as well as automotive development trends are significant. Overall, this will eventually lead to better systems. The goal of these systems is to improve vehicle safety, avoid fatalities, and minimize the frequency of accidents on the road.

3 Airbag Application Tools

In this chapter, there are three sections. The first section gives a brief overview of airbag application tools. In the second section, the existing solution for the AB application tool which is used to visualize crash channel assignments is discussed. In addition to the concept of the old solution, the disadvantages are also highlighted. This chapter ends with the summary section. This section gives an overview of the key points covered in this chapter.

3.1 General Overview

Software tools for airbag application are important in the development, testing, and deployment of airbag systems in vehicles.

Software tools enable engineers to design and simulate airbag systems. Different designs and configurations can be tested. This helps to ensure the performance and safety of airbag systems. There are various software tools which are used to calibrate and test the airbag system. The system is tested in various scenarios to ensure it deploys correctly and protects occupants. This also helps to make sure the system meets its safety standards and regulations. There are different software tools used to analyze data from airbag deployments, based on varying requirements. This collected information helps to improve the design and performance of airbag systems, making them safer for passengers.

For example, airbag systems rely on sensors to detect a crash and trigger airbag deployment. There can be different kinds of data available from the sensors. This data can be extracted by the AB tool as per requirements. By analyzing the extracted information, it can be checked if the sensors are working correctly. Hence, the airbag system can be improved.

Overall, airbag application tools are essential. They are used for the development and deployment of airbag systems. AB tools allow engineers to design, simulate, test, and improve the airbag system. This helps to ensure the performance of airbag systems, hence providing occupant safety in the event of an accident.

During a crash, the sensors on a vehicle send a quick response to the ECU. When this happens, details of the crash are stored. This data which is stored is then converted into machine-readable format. Then, it can be analyzed with the customers based on their requirements. The whole set of possible outcomes that can cause different airbags to deploy in varying environments is checked.

A brief overview of the airbag application tool chain can be seen in Figure 3.1. Crash tests are performed by placing crash test dummies in the vehicle. From these crash tests, the crash signals are recorded by the customers. Then, customers provide raw crash data to the company. This data may include different kinds of information, such as the model of the vehicle, and severity of damage caused to the vehicles. This crash data is basically used to improve their vehicles, which results in better safety features.

An airbag application tool (Tool 1) is used to edit this data which is provided by the customers. It is then stored into a database. This database is called the Crash DB. This database stores all the input data recorded during crash tests. It consists of many single crashes, which is also referred to as a crash set. In a crash set, every crash has a specific crash name.

During the calibration phase for airbag projects, there is another application tool (Tool 2) used for simulation purposes. It is used for the simulation of airbag ECU software on computer systems. This tool provides a generic simulation platform. It has graphical editors to view and modify the simulated system. Along with single control units, the tool also simulates systems of control units. For example, a central control unit with peripheral sensors.

Along with the crash data, some other files are also needed as input for simulation. For example, a configuration file, a specific algorithm, and a file consisting of different parameters. Customers have different specifications. Depending on these specifications, there are different parameters for calibration. These parameters consist of information regarding different conditions. Based on these conditions, the basic logic for firing devices such as airbags is different.

There are two phases during a simulation. During the first stage, data is pre-processed. Then, the actual simulation is run in the next stage. The output from the simulation is particular result values for each simulation device. This data is written into the Result DB. So, the simulation tool is used to generate the result database. This is a general overview of some airbag application tools in the toolchain.

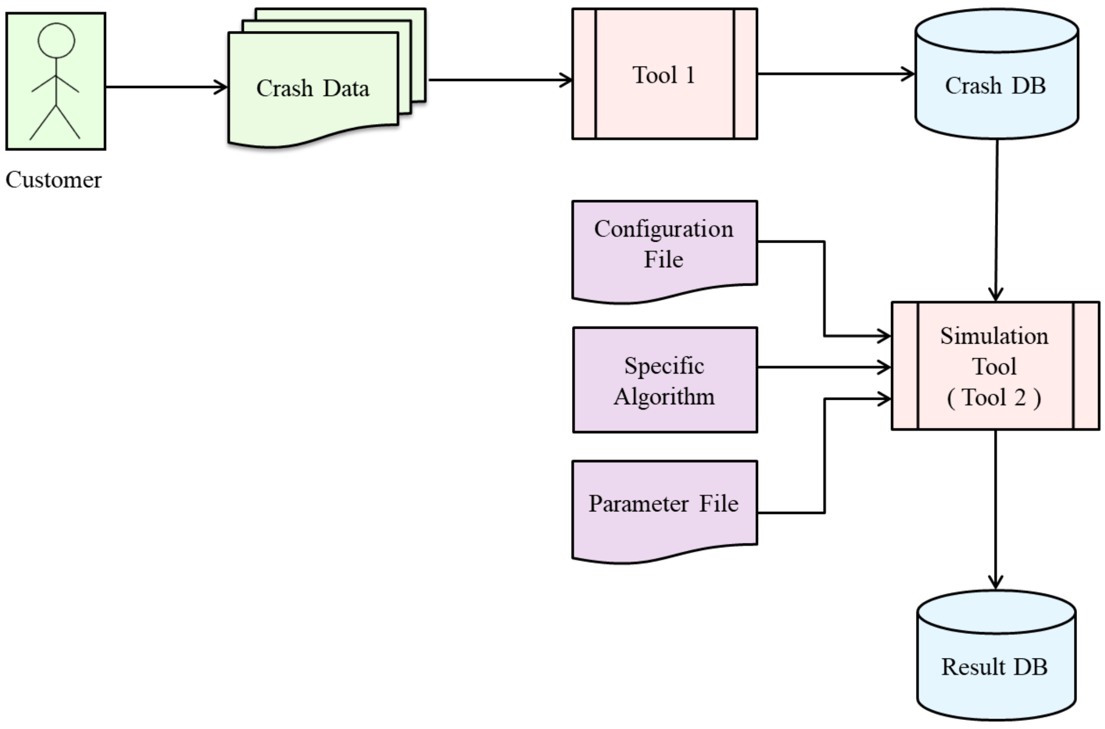


Figure 3.1: General Overview of the Airbag Application Toolchain

Color	Depiction
🟩	Customers providing raw crash data to the company
🟥	Airbag Application Tools in the toolchain
🟦	Databases (Crash DB, Result DB)
🟪	Input files for simulation

Table 3.1: Color Depiction of the Airbag Application Toolchain

There are many airbag application tools which help in the simulation, calibration, and crash-data management for the airbag projects. There are a few main tools, as well as many smaller helpful tools.

Amongst the airbag application tools, “AssignmentLister” is the name of the tool which is used to visualize crash channel assignments. The airbag project has sensors, where in each of them needs data for simulation. For this very purpose, crash channels are assigned to the sensors. Each of the sensors present has a crash channel assigned to it. This assignment is done for every crash present in the project. These assignments are known as crash channel assignments. The assignments should be shown in a tabular form. This is a requirement as it is necessary to see if the assignments are correct.

The AssignmentLister tool shows for each crash of a project, the assigned channels for each sensor in a tabular form. So, basically the tool is developed in order to show the crash channel assignment information.

A brief overview of the application tools used for airbags was discussed in this section. The need for these tools was also explained. Following that, some of the airbag tools developed by Bosch were discussed. This section ends with an explanation about the AssignmentLister tool. This tool displays crash channel assignments in a tabular form. In the following section, the existing AssignmentLister tool, along with its drawbacks will be discussed.

3.2 Existing Methodology to Visualize Crash Channel Assignments

In this section, the old solution used to visualize crash channel assignments is discussed. Following that, the limitations of the approach are analyzed and explained in detail.

The AssignmentLister tool is running as a macro within the Excel file. The existing methodology can be seen in Figure 3.2. As a first step, the tool runs a macro within the Excel file. Then, it reads the input files for the channel assignment information. The crash channel assignment information, also known as mapping information is got. As a last step, this information is added to the Excel file. Excel Macros are used for this purpose in the tool.

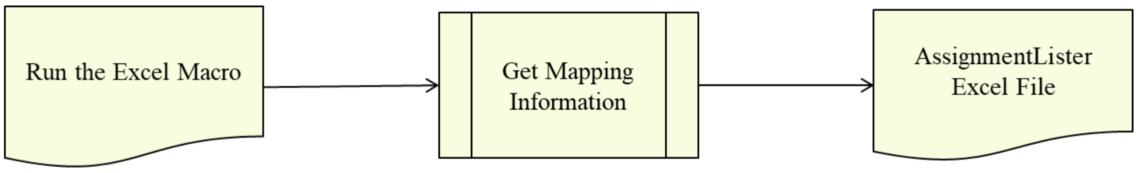


Figure 3.2: Existing Approach to Visualize Crash Channel Assignments

The AssignmentLister tool shows the assigned channels for each sensor in an Excel table. In this Excel file, the assignments are shown for each crash of a project. All channel assignments are listed in a tabular format. There is also a list of available channels in the Excel file. The list of available channels is taken from the Crash DB.

When the macro is finished running, the tool directly shows the Excel workbook with some information about the crash, and the channel assignments. The Excel workbook is then saved in the same folder as the project file, or outside the project folder.

Some drawbacks are analyzed from the existing tool. The method used is quite complex. The user has to perform many mouse clicks to get the output in Excel. The output from the tool is not directly available using the existing method.

The process in which the Excel file with crash channel assignments is created is depicted in Figure 3.3. The macro is run through an Excel file. First, the macro should be installed separately. Then, the “Add-ins” button is clicked in the file. Following that, “Macros” button should be selected. In the options displayed following the mouse click, the tool name, “AssignmentLister” should be selected. When this step is done, a dialog box appears. There is a field in the dialog box where the input file location can be entered. After entering this, the “Start” button should be clicked. After this step, the user has to wait for some time. Then, the tool writes the crash channel assignment information to Excel. These are the steps which the user has to perform in order to get the final output from the tool. This procedure is quite complex and tedious for the user. It also consumes a lot of time to do these steps, and then get the final output, which is the Excel file.

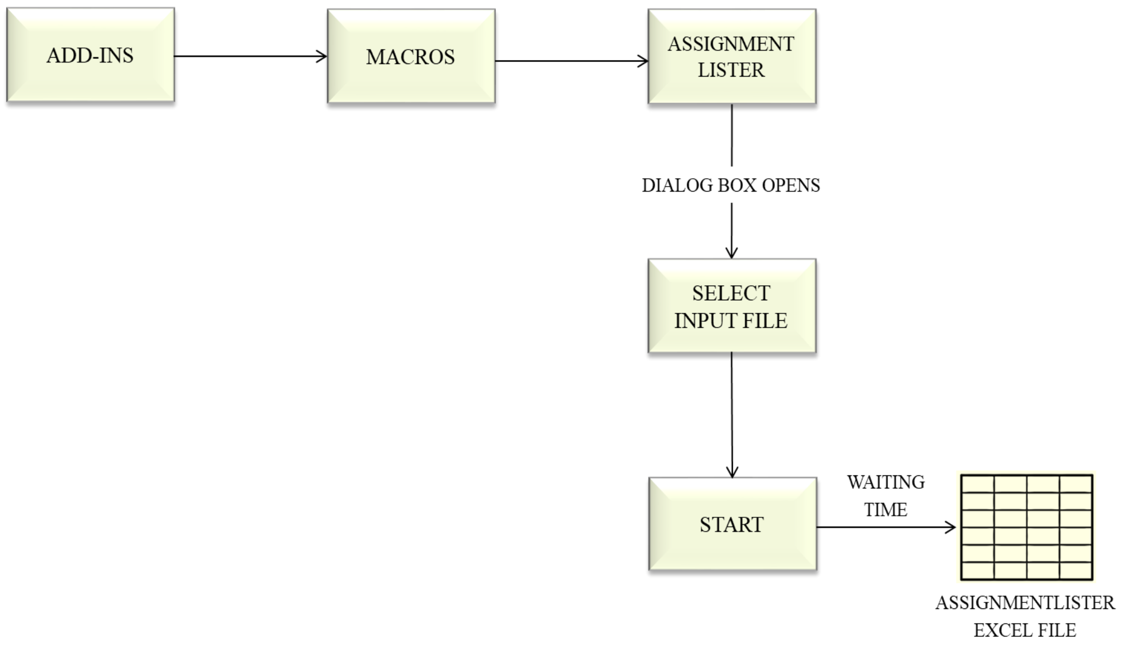


Figure 3.3: Steps to get Crash Channel Assignments in a Tabular Form

There are a few limitations of using Excel macros to run the tool. Since the AssignmentLister tool runs as an Excel macro, separate macro installation is necessary. Updating the macro is a lot of effort. This is because the macro is combined with other macros. Instead of updating a macro, it is a less tedious process to create a new macro even. In addition, using macros to generate the Excel file slows down the overall process. Hence, using macros to get the crash channel assignments in a tabular form consumes a lot of time.

While running the tool, the clipboard cannot be used. If the clipboard is used, the Excel file with the crash channel assignment information is not generated. The system may even freeze, leaving the screen black for an extended length of time. The user will just have to wait for the system to restart. In the meantime, the system also cannot be used when this happens. So, it may take more time than expected to generate the output. Sometimes, the page doesn’t respond at all. Other MS Office Products also cannot be used when the tool is run using a macro. The tool is quite unstable. It does not work as expected.

There are two different use cases possible with the tool. A complete airbag simulation project is available. The tool runs as a macro within the complete simulation project file. In this use case, assignments from the complete simulation project are checked. The simulation tool file or folder path is used as an input parameter by the tool. The files inside the project folder with the channel assignment information are read. The mapping information is got, and then added to the Excel file. From the complete project, only the simulation results are available. The tool runs as a macro within the simulation result file. This use case checks the assignments from only the simulation results. The simulation must be completed first. Only then the tool can be run successfully. The result file from the simulations should be available. If this is available, then the channel assignment information is added to the Excel file. The programming language is in Excel VBA. There is no common solution for these available use cases in the tool.

To summarize the drawbacks discussed, the existing solution uses Excel macros. The tool is slow and unstable because of the use of macros. Assignment information is not handled in a well manner. Many manual steps are needed to get the final output with the crash channel assignment information. This process is quite cumbersome. Using macros to generate the output has negative effects on MS Office Products. There is no common approach for the different use case scenarios. The aim is to eliminate these limitations. As a consequence, the AssignmentLister tool's overall performance shall be enhanced.

3.3 Summary

Airbag application tools are critical in the development, test, and deployment of airbags in vehicles. In Section 3.1, an overview of airbag application tools was discussed. The significance of AB application tools was emphasized. These tools help simulate, calibrate and manage crash data for airbag projects. Overall, these tools improve the airbag system, ensuring the correct functionality of airbag systems. This helps to provide better occupant protection. This section ends with an introduction to the AssignmentLister tool, which is the application tool used to visualize crash channel assignments in a tabular form.

In Section 3.2, the existing method to display crash channel assignments using the AssignmentLister tool is explained. Further, the drawbacks of the old method were analyzed. These limitations reduce the overall performance of the AssignmentLister tool. The aim is to mitigate these drawbacks in order to have a better performing tool.

4 New Methodology to Visualize Crash Channel Assignments

This chapter is divided into four sections. Each section covers different aspects of the methodology to display crash channel assignments. In the first section, different approaches which could be considered for the solution are explored. An overview of the new methodology to visualize crash channel assignments is given in Section 4.2. In Section 4.3, this methodology is covered in detail. It is further split into three sections. Each sub-section is a stage of the concept. For example, the first sub-section is the first stage, and so on. This is done for easier understanding of the overall concept which is developed. The fourth section consists of the summary of this chapter.

4.1 Different Approaches

There are various approaches to be evaluated for the solution. The chosen methodology, along with the reasons for which it is chosen is discussed in the following sections. This section considers and explains different methodologies.

The airbag application tool, AssignmentLister is important as it is necessary to visualize crash channel assignments in a tabular form. The assignment information should be handled in the tool. It is required to find a suitable file format for this purpose. The target is to transfer the information to another format, such that the output is in a tabular form.

There are some drawbacks with the old solution. Macros are used, which have a lot of disadvantages on the tool. They make the tool slow, unstable and complex. The drawbacks are discussed earlier in more detail, in Section 3.2. The different approaches explained will be evaluated to achieve the aim of developing the new tool, without these drawbacks. The idea is to have a methodology without the use of macros. For this purpose, a good fitting file format should be found to handle the assignment information. Therefore, various available file formats will be discussed. The various methods to have the output in a tabular form shall also be explored. The objective of this section is to explore different file formats, and methods to output the assignment information in a tabular form.

For storing assignment information, different file formats are available. Some of the file formats which are discussed further are text file, CSV, JSON, and XML.

A text file is a simple format, having the extension of .txt. Text files are good to store data which is not complex. It is difficult to organize data within text files, as this type of file format does not have a well-defined structure. Since text files are not structured, it is hard to extract any specific information needed from the file. There are also no format options, such as font styles or colors available. To sum it up, text files can be a good choice for storing simple data. They are incompatible with more complex information requiring a structured format.

CSV is a file format where data is stored in plain text. This plain text is separated by commas. This file format has limited support for different data types. Important data may be lost during conversion as a result of this. Complex data structures like arrays are not well-suited to be stored using this format. There is no standard format used to store data. This makes maintaining data consistency complicated. Checking if the data is complete or correct might be time consuming, and possibly error prone.

TSV is a file format where tabs are used to separate the information. TSV has an added advantage compared to CSV. Tabs are usually not used in text so often. This makes TSV easier to parse than CSV. In case the value in the data contains a tab, then it is difficult to parse the file.

JSON is generally used for web applications. JSON files can be complex, as the syntax is difficult to understand. JSON files do not have a built-in way for a specific structure. Wrong data structure is also accepted using this file format. This can result in errors. In some cases, the data may lose its consistency.

XML can handle complex data structures. Tags and attributes are used in XML. Data is stored in a hierarchical way. It is easy to organize information using XML. It has a built-in way for a specific structure. This can be useful in many cases. For example, it is easy to parse the XML file in order to get any particular information needed from the file. It is widely supported by numerous apps and platforms. As a result, the XML format is versatile. XML can be more complex than other file formats. This makes learning it more difficult. Due to the additional overhead with the tags and attributes used in XML, the file size may become larger in comparison to other file formats.

The different file formats available are discussed. Table 4.1 shows a summary of some of the main points discussed about the different file formats.

File Format	**Advantages**	**Disadvantages**
Text	- Suitable to store simple data	- No structure
CSV	- Stores data in plain text	- Limited support for data types - Not so easy to parse the file and get information
TSV	- Easier to parse than CSV	- Difficult to parse if values have tabs
JSON	- Widely used for web apps	- Syntax is difficult to understand - No built-in structure
XML	- Can handle complex data structures - Easy to parse and get information	- File size may be larger in comparison to other formats

Table 4.1: Comparison of Various File Formats

The pros and cons of the file formats discussed are highlighted. These are some of the file formats which could be used to store crash channel assignment information. The suitable file format is chosen based on the requirements of the project.

First, the information is stored in a good-fitting file format. Then, it can be converted to a tabular form. There are different formats available to store the data as a table. Some common formats which are used for this purpose are HTML and Excel. Examples of these file formats which are used to visualize data in a tabular form can be seen in Figure 4.1 and Figure 4.2 respectively. These file formats are explained further.

HTML is a markup language used to show data by creating web pages. Figure 4.1 depicts an example of an HTML web page.

HTML pages are styled using CSS. HTML tables are static. They do not provide dynamic features. This limits the functionality of the tool used to display the assignment information. Hence, the formatting possibilities are limited. It is complicated for users to edit the tables. For example, adding or deleting information from the table is complex or even not possible at all. It may be challenging to present the data in an orderly manner. HTML tables may not appear correctly on some devices and web browsers. This is due to its incompatibility. This can cause issues with the formatting of the table. These issues in turn make it hard for the information to be visible clearly for users. Using HTML to display the information can also be risky. Cross-site scripting attacks are a security threat. These incidents occur when malicious software is put in a web page. The intruder can then steal sensitive data.

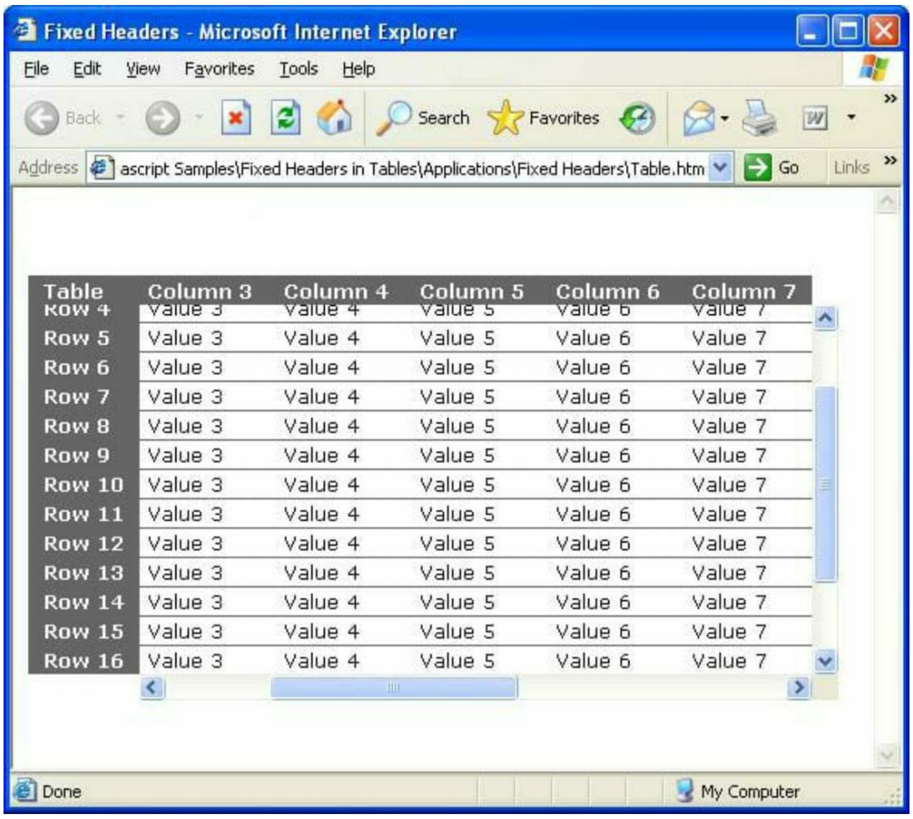

Fixed Headers - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
Address
ascript Samples\Fixed Headers in Tables\Applications\Fixed Headers\Table.htm
Go
Links
Done
My Computer

Table	Column 3	Column 4	Column 5	Column 6	Column 7
Row 4	Value 3	Value 4	Value 5	Value 6	Value 7
Row 5	Value 3	Value 4	Value 5	Value 6	Value 7
Row 6	Value 3	Value 4	Value 5	Value 6	Value 7
Row 7	Value 3	Value 4	Value 5	Value 6	Value 7
Row 8	Value 3	Value 4	Value 5	Value 6	Value 7
Row 9	Value 3	Value 4	Value 5	Value 6	Value 7
Row 10	Value 3	Value 4	Value 5	Value 6	Value 7
Row 11	Value 3	Value 4	Value 5	Value 6	Value 7
Row 12	Value 3	Value 4	Value 5	Value 6	Value 7
Row 13	Value 3	Value 4	Value 5	Value 6	Value 7
Row 14	Value 3	Value 4	Value 5	Value 6	Value 7
Row 15	Value 3	Value 4	Value 5	Value 6	Value 7
Row 16	Value 3	Value 4	Value 5	Value 6	Value 7

Figure 4.1: Example of an HTML Web-Page [30]

Another file format which can be used is the Microsoft Excel. Excel is a spreadsheet program. The example of an Excel sheet can be seen in Figure 4.2.

Excel stores data in a tabular structure using columns and rows. It provides a simple user interface. Excel files are not static. They provide dynamic features. It has a variety of formatting and calculation possibilities. Excel makes data handling and modifications simple. For example, adding or deleting information from the table is easy. It is simple to customize the data which is displayed in Excel. There are many options for customizing and formatting data to improve readability. Excel files might be vulnerable to errors while handling special characters.

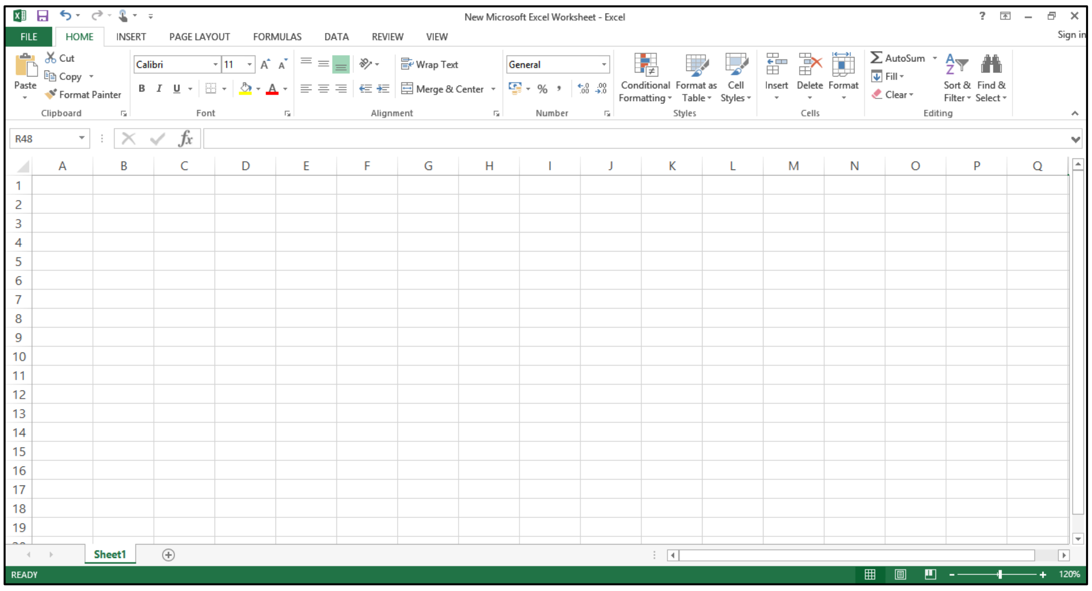


Figure 4.2: Example of an Excel Spreadsheet

The choice of file format to store data, and visualize the assignment information in a tabular form depends on the specific requirements of the project. Other factors also impact this selection. These factors are the type of data, and the way in which the assignment information is extracted. Each of the file types considered has benefits, as well as limitations. It is crucial to choose the right format to store assignment information according to the requirements. The specific use case and its requirements should be considered when choosing between different file formats.

4.2 Overview of the Methodology

This section introduces the idea for displaying crash channel assignments. Section 4.3 will cover the further details about this methodology.

The aim is to develop an airbag application tool in order to show the assignments. The visualization of crash channel assignments should be possible without the use of macros. The new solution should lessen the limitations analyzed from the old solution too. For this, it is required to find a good file format to handle information in the AssignmentLister tool. The different approaches to store the assignment information are discussed in Section 4.1. The target is to transfer all the data from this file format to a table. The final result should meet the specifications. The methods possible to store data in a tabular form are also discussed in Section 4.1.

A basic overview of the proposed methodology is depicted in Figure 4.3. The database consists of the required information, which are the crash channel assignments. This information is extracted from the database. It is then saved to an appropriate file format. Then, from this file, the data is transformed to a tabular format. This is the output file from the AssignmentLister tool. The crash channel assignment information should be available in a table as the final outcome. The concept to develop the tool is divided into three main stages, the assignment information extraction, intermediate steps and the final step. Extracting assignment information from the database is defined as the first step in the developed concept. Storing this information to a good-fitting file format is the next step. This step is also the intermediate step in the solution. The conversion to the AssignmentLister output file is the final step.

- AssignmentLister Tool: ⬚ (Depicted in Figure 4.3)

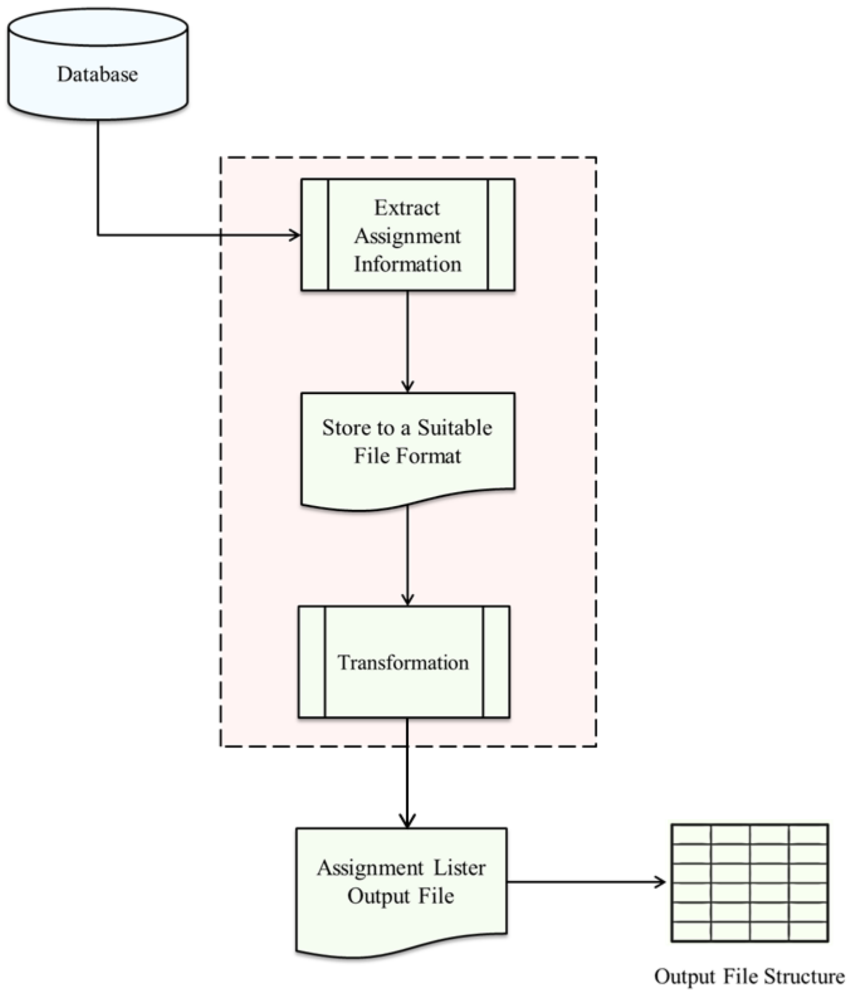

Database
Extract Assignment Information
Store to a Suitable File Format
Transformation
Assignment Lister Output File
Output File Structure

Figure 4.3: Overview of the Proposed Methodology

This is a simplified illustration of the new solution for displaying crash channel assignments. The following section explains the reason for choosing this methodology in detail. The suitable file formats for the intermediate result requirements and the final result are also discussed further.

4.3 Concept

In Section 4.2, an overview of the proposed methodology is given. This section discusses the concept developed to visualize crash channel assignments in detail. In order to achieve this, the section is split into three sub-sections for better comprehension. Each part in this section is a stage of the concept. The first sub-section describes the information extraction. The second sub-section explains the intermediate steps. The third part is the final stage. It describes the final step of the concept. The AssignmentLister tool handles all these steps. The idea to develop the tool is using C# programming language with Visual Studio IDE, as shown in Figure 4.4.

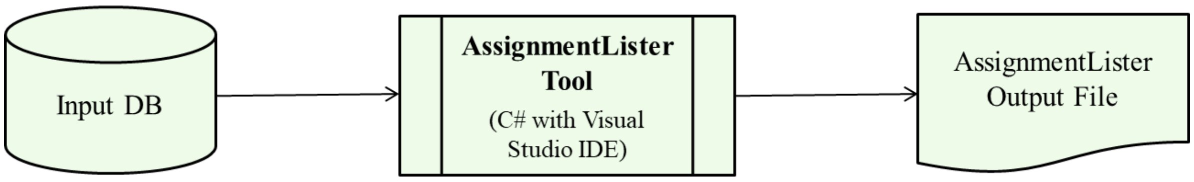


Figure 4.4: Basic Concept of the AssignmentLister Tool

4.3.1 Information Extraction

The information extraction is the first step in the developed concept. To be able to view the crash channel assignment information in the final step, the assignment information should be extracted first. The airbag project is simulated using the simulation tool. This simulated data is stored in the result database.

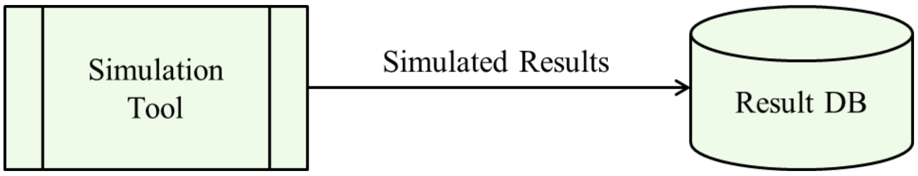


Figure 4.5: Storing AB Simulation Results

The simulated results consist of all information from a crash, including the channel assignments. From this step onwards, there are two possibilities to get the channel assignments. Both the possibilities will be discussed, and an approach will be chosen from the two. The possibilities are the following:

a) The first option is to extract the channel assignments directly from the Result DB.

b) There is another airbag application tool called RSDBnext. This tool uses the Result DB as input. It generates two result files. These result files also consist of the channel assignment information. Extracting the channel assignment information from this result file is the second option.

The assignment information is also called the mapping information. In both the possible options, the assignment information is available for the AssignmentLister tool. Using the first option, the mapping information should be extracted by the AssignmentLister tool from the Result DB. This can be done using SQL queries. Using the second option, the mapping information should be extracted from the result files generated from the RSDBnext tool. This tool uses the Result DB as input.

Option (a) is using SQL queries to get information from the database. This can be complex. Databases may have a large amount of data. Hence, extracting information from the database can consume more time. In this case, the database contains various crash related information. The queries must be written correctly to extract the right information. If the database is modified or changed, the SQL queries also need to be updated accordingly. This can get complex. There are rows, columns, and relationships between them. In order to extract information from the DB successfully, these relationships have to be understood. For example, the relation between tables in the database or the hierarchy in which the data is stored. DBs may also have security restrictions. Additional access permissions may be needed to get information from the tables. This can add to the complexity of extracting information from the database. However, extracting information from the DB directly is a feasible method, but it also has some cons. These should also be considered while choosing the most effective solution to extract crash channel assignment information.

In the second option (b), result files generated from the RSDBnext tool consists of assignment information. There are two result files generated from the RSDBnext tool. The format of these files is Excel and XML respectively. Extracting information from Excel is not such a good idea as the approach is not so easy. The formatting and structure of the information may change when exporting data from Excel to another format. In the XML file available, the crash channel assignment information is available in a hierarchical structure. It is easy to parse the data in XML than Excel.

In the first option, the mapping information should be extracted using SQL queries directly from the database itself. Using this approach is more complex to get the information compared to the other option available. The aim of developing the AssignmentLister tool is to be able to handle the assignment information in an easy manner. Therefore, option (b) is more suitable in this case. Hence, the crash channel assignments are extracted from the result file in XML format. Though both the options are feasible to do in this case, the first option (a) is more complex. Hence, the second option (b) is the better one.

Hence, in the AssignmentLister tool, the mapping information is extracted from the result XML file of the RSDBnext tool. This is shown in Figure 4.6.

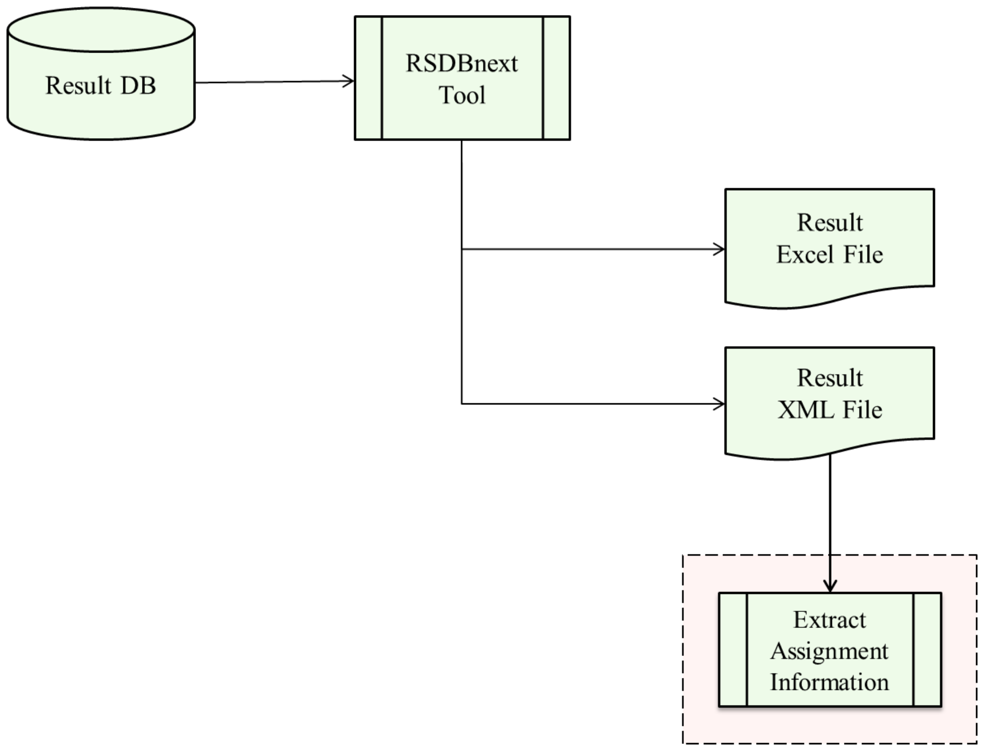


Figure 4.6: Extraction of Crash Channel Assignment Information

The reason this option is chosen amongst the other possibilities is discussed in this section. At the end of this step, the assignment information is extracted from the Result XML file. This is the first stage in the concept.

The idea is to extract the assignment information by choosing the less complex and more feasible option in this case. This will in turn result in a tool which is not complex in the end. Hence, the handling of assignment information becomes easier for the next steps.

4.3.2 Intermediate Steps

This is the second step in the developed concept. The assignment information is collected. Now, this information should be handled further in the AssignmentLister tool. After choosing the suitable option in the first step, the assignment information is extracted from the result file of the RSDBnext tool.

The assignment information is extracted from the Result XML file of RSDBnext tool. There are different possibilities for further steps. These possibilities are discussed and the most suitable method is chosen.

a) There is a possibility of storing the assignment information to a tabular format directly, and skipping the intermediate steps.

b) Storing the assignment information to a file, and then converting this to a tabular format.

Option (a), which is converting the assignment information directly, has a few disadvantages. Directly converting the assignment information may lead to formatting issues. When there are formatting issues during conversion, there may be information lost in the process. Some elements may not be interpreted correctly from the input file. This leads to errors in the output. Directly transferring data to a tabular format also leads to performance issues.

Option (b), which is converting the assignment information to another file, then converting it to a tabular format provides better organization of the file. It also provides good compatibility with various software and tools working with the output file. Using an intermediate file ensures the data is not lost, and necessary formatting specifications are met. Hence, option (b) is better than option (a) in this case.

Therefore, it is better to have the intermediate steps for the reasons discussed. The performance of the overall AssignmentLister tool also increases by this methodology.

In this intermediate step, it is important to put all the assignment information which is extracted together. This is done by creating an internal data structure which is appropriate according to the information available. The assignment information should then be stored to a file. The different type of file formats which may be used to store this information is discussed in Section 4.1.

The requirements are that the assignment information should be stored in a structured manner. It should be easily readable. Since the data is transformed to a tabular format later, it should be easy to read from this file type. The file formats discussed in the earlier section were text files, CSV, TSV, JSON, and XML. It is important to identify the good-fitting file format amongst these file types. The requirements of this specific use case should be considered.

Text files cannot store complex data well. They are unstructured. It is important to be able to find the necessary information easily from the file. This is not possible using text files. A structured format is required to store the assignment information. Hence, text files are not suitable for this use case. CSV is difficult to read and understand. This file format does not support all data types, which may result in data loss. Assignment information should be displayed correctly without any information being lost. Hence, using this file format is not so appropriate for this case. TSV format is better than CSV as the data is separated by tabs, and not commas. It is more readable in comparison to CSV. But overall, it does not provide a good structure for the channel assignment information. JSON files don’t have a specified structure. The disadvantage of using JSON is that the wrong data structure is also accepted using this file format. This can lead to errors, or data inconsistency in some cases. As a result, using this file type is not suited for this purpose. Therefore, text files, CSV, TSV, and JSON are not fitting for this usage.

Since tags and attributes are used in XML, it is easier to identify the necessary information successfully in the later steps. XML data is simple to retrieve and parse. The built-in way for a specific structure in XML makes the assignment information easy to read. Since the next step is converting the XML to a tabular format, it is important to be able to parse the XML file, and get the assignment information easily from it. It is appropriate to store the information in XML file format for this particular use case. It is more beneficial to maintain information in a hierarchical structure for the available data. Hence, XML is the most suitable file format amongst the others for this use case with handling assignment information.

So, the idea is to extract assignment information as a first step. Next, this information should be handled in the AssignmentLister tool. Following that, the strategy is to store the assignment information to an XML file. From this XML file, the assignment information shall be transformed to another file having a tabular format in the final step.

For this purpose, the XML file is transformed to another XML file with styles. The first XML file created shall have the crash channel assignment information in a hierarchical structure. The following XML file which is created will have the assignment information, as well as the respective styles applied to it in the file. Having the assignment information in XML format gives a good overview of the stored information. This information is then present with all of its styles. For example, for each value displayed in the table, the respective style can be seen in XML format. These steps are depicted in Figure 4.7. The necessity to apply styles to the XML file, type of StyleSheet used, and the purpose of creating two XML files is discussed further.

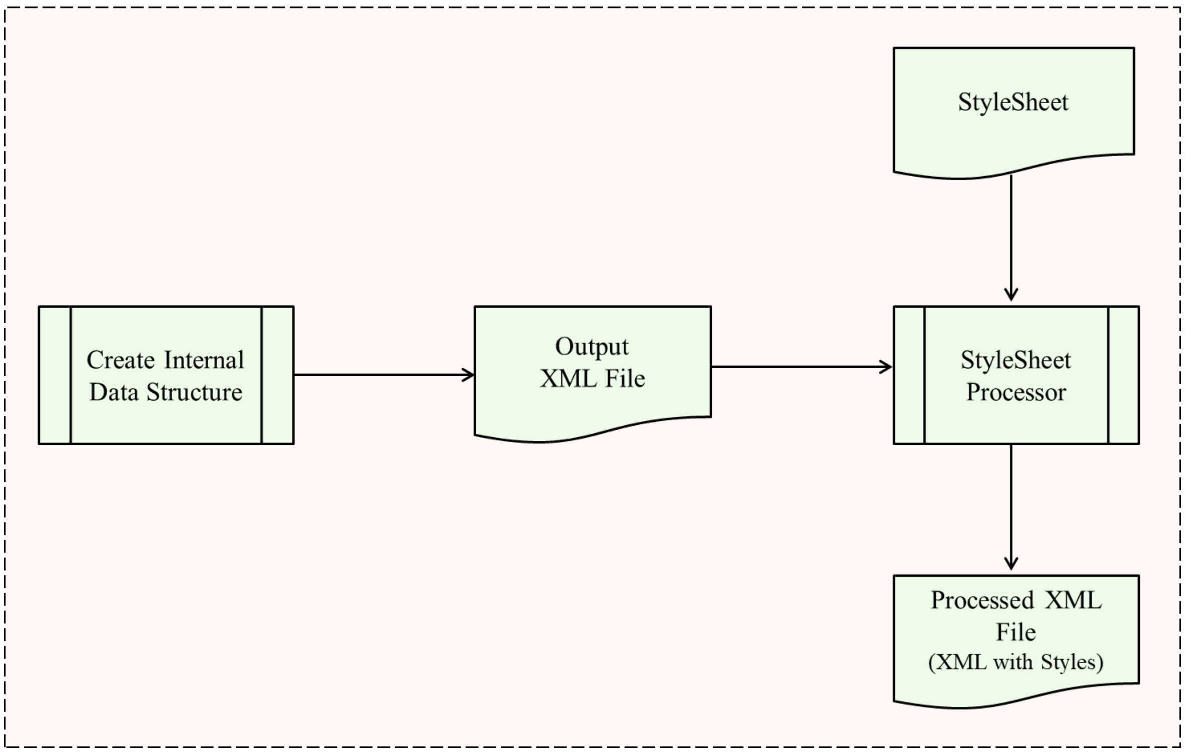

StyleSheet
Create Internal Data Structure
Output XML File
StyleSheet Processor
Processed XML File (XML with Styles)

Figure 4.7: Intermediate Results

Styles help in defining the structural elements of a file. By adding some styles to a document, the information is displayed in a structured and clear manner. In addition, styles help in automating formatting tasks. This helps to save the manual time taken to do it. Furthermore, it helps in maintaining consistency of the styles throughout the document, for all the information available. It also makes the file look visually better.

In this case, styles are required to present the assignment information in a tabular format. Hence, styles are applied to the XML file. XML describes data in a structured format. It does not have an inherent tabular structure. Hence, data is converted from XML to a tabular format, where the information available is structured into rows and columns. For example, styles define the XML elements and their corresponding columns in the tabular format, or specify formatting like having bold for the text present in some cells in the tabular format. If some XML elements should be displayed in a specific font or color, styles help to make sure this format option is applied to those elements throughout the table. This helps to ensure that the information, along with the styling options is consistent in the entire document. Overall, styles have a crucial role in ensuring that the assignment information present in the XML file is represented effectively in a tabular format. Having a clear structure with styles for all the information in the tabular format makes it easier to visualize the crash channel assignments.

Some of the styles which can be used are table styles, cell styles, character styles and conditional formatting styles. Table styles are the styles which add formatting to tables. For example, borders and coloring of the table. Cell styles are the styles which add formatting to individual cells in the table. For example, font style, background color, and text alignment. Character styles add formatting to specific characters. Conditional formatting styles are used to apply formatting to cells depending on certain criteria. For example, highlighting cells with a certain background color to indicate an error. The type of style needed depends on the requirements of the final output.

Styles can be applied to the XML file using various StyleSheets. The possible styling ways considered to display assignment information are CSS and XSLT.

a) Using CSS
b) Using XSLT

CSS and XSLT are used to style and transform XML data. They have different strengths.

CSS can be used to format documents written in XML. In order to do this, styles are defined in a separate CSS file. Then, they are applied to the XML file. CSS generally controls the presentation of a document or web page. It can be used to add formatting to XML documents. But, CSS has limited features to add styling options. It is not well-suited for complex transformations. It is easier to do simple transformations using CSS.

XSLT uses its own stylesheet language in order to apply styles to the XML document. It is used to convert an XML document to the same or a different file format. XSLT is designed specifically for XML. It provides options to extract, edit, alter and transform data from XML documents to different output formats. XSLT is capable of performing complex transformations. It can combine multiple XML documents, filter data, or even apply conditional formatting styles. Using XSLT, different formatting specifications can be applied to specific parts of the information in the XML file. Due to this, different customizations are possible. Some of the cons of XSLT are that it requires a deeper understanding of XML. It may be more complex than CSS.

Feature	**CSS**	**XSLT**
Styling	Limited options	More options
Suitability	Simple tasks	Complex tasks
Strength	Styling HTML	Styling and transforming XML documents to an output file
Output	Visual display in a browser	New document

Table 4.2: Comparison between CSS and XSLT

XSLT is designed particularly to handle XML documents. XSLT is a more powerful tool to transform XML data to different outputs. Hence, converting the XML document using XSLT is better [31]. XSLT has more styling and formatting features compared to CSS. The assignment information in the XML file shall consist of different crash information in a hierarchical manner. Different styles are to be applied on this information. Also, conditional formatting options may be needed for some information. For simpler transformations or when only basic styles are needed in the XML document, CSS may be a better option. However, in this case, using XSLT is preferable for the previously mentioned reasons. Therefore, Option (b), which is using XSLT to style XML documents, is more suitable than using CSS in this case.

To summarize the intermediate steps in a simpler manner, first, the assignment information is stored in an internal data structure. This information is then put to an XML file. This XML file consists of the assignment information. It is depicted as the Output XML File in Figure 4.7. Styles are applied using XSLT. Then, another XML file is created. This XML file consists of the assignment information with its respective styles. It is depicted as the Processed XML File in Figure 4.7.

By creating two XML files, the raw assignment information is separated from the styling information. This separation of data and the styling makes it easier to manage the assignment information in the next steps. It is helpful to have the second XML file in order to transform the assignment information to a tabular format. If there is a need to modify styling, it can be done in the second XML. This does not affect the assignment information stored in the first XML file. Providing users with the information regarding the styles applied to the data can also help them get an overview of the styling information. Furthermore, having two separate XML files makes it easier to integrate the assignment data with other systems which may require different styling or formatting options. The assignment information stored in the first XML file created can be reused. If the same information is required, but different styling options, then the XML file can be reused. The styles applied to this file do not have to be taken care of in this case, as it is present in a different XML file. This can help in saving time in developing different output formats with the same data. Hence, this strategy to handle assignment information in the tool gives more flexibility, reusability, and interoperability. Therefore, this strategy shall be beneficial to the AssignmentLister tool.

In the intermediate step, there are two XML files which shall be created. The first XML contains the assignment information. Styles are applied to the XML file using XSLT. The second XML file stores the assignment information along with the respective styles. This file can be used by users of the AssignmentLister tool to get an overview of the crash channel assignments, along with information on the styles applied. The assignment information in the XML file is to be transformed to an output having a tabular format in the end. The next step, which is also the final step, is to transform this XML file with styles to a table.

4.3.3 Final Step

This is the third step in the concept. In the first step, assignment information is got. Then, this information is handled in the tool. As a next step, the crash channel assignment information is stored in an XML file. Then, styles are applied to this assignment information. Another XML file is created which consists of all the assignment information with its styles accordingly. The last step in the concept is explained in this section. The generated XML file needs to be translated to a different format. This file format should store the information in a tabular form.

Factor	HTML	Excel
Data	Unstructured	Structured
Orderly Presentation (in a tabular format)	Challenging	Easy
Formatting Possibilities	Limited features	Extensive features
Customization Options	Limited options	More options
Ability to Edit	Complicated	Easy
Dynamic Features	Static	Dynamic
User-Friendly Nature	Not so user friendly, difficult to use	User friendly interface, easy to use
Compatibility Issues	Not compatible on some devices	Generally compatible
Security Risks	Vulnerable to attacks	Generally secure

Table 4.3: Comparison between HTML and Excel for Data Visualization

In Section 4.1, the different file formats which can be considered to store tabular data were discussed. The possibilities which are considered as file formats to store data in a tabular form are HTML and Excel. As explained, the two formats have pros and cons. The use case and its requirements are considered to be able to choose amongst the two options. The first factor to be considered is for the file formats to output the crash channel assignment information in a tabular form. Both HTML and Excel satisfy this point. The next factor is for the user to be able to format the file as per requirements. It is necessary to be able to style the file as per user specifications. In HTML, there aren't as many formatting and styling features.

In Excel, there are more options for formatting and styling. Excel can be used to filter data. The requirement to filter assignment data is necessary. The final result should also be editable or modifiable by the user. For example, if the user tries to delete a row or add an additional column, this should be possible. Using HTML, it is not possible to modify the data once the final output is available. Using Excel, this is possible. The final output can be modified as needed. Hence, for the stated reasons, Excel is a more suitable file format to be used for this use case with crash channel assignments. For showing crash channel assignments, the requirement is also to have the output file in Excel. Therefore, the final output is in Excel format.

The Processed XML file consists of the assignment information, along with the styles. This is available from the intermediate step. In the final step, this file is converted to an Excel file having a tabular format. To summarize this, the XML file with styles is transformed to an Excel file in the final step of the developed concept. This is depicted in Figure 4.8.

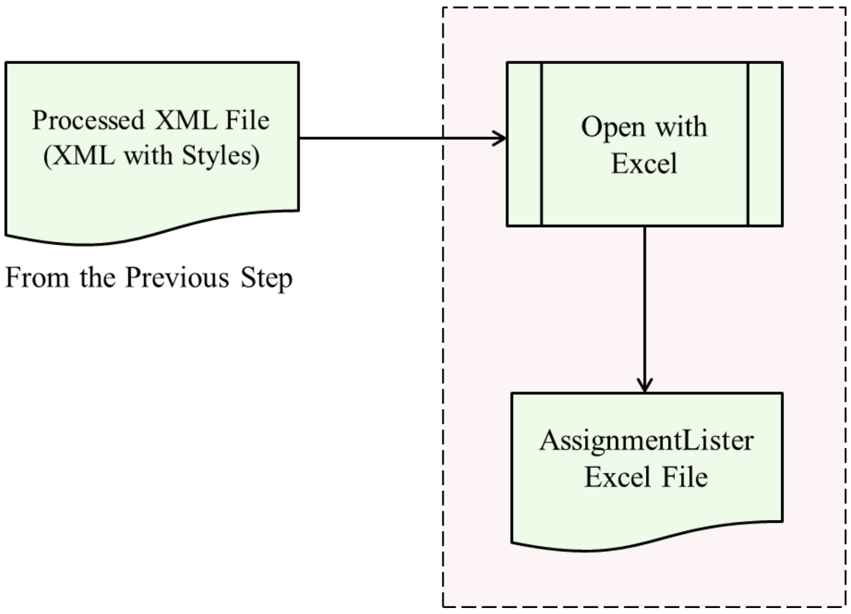


Figure 4.8: Final Step to Visualize Crash Channel Assignments

The final result, which is the AssignmentLister Excel file, consists of the crash channel assignment information in a tabular form.

4.4 Summary

This chapter consists of the new methodology to display crash channel assignments. In Section 4.1, the different approaches for the solution were discussed. The ways in which the assignment information can be handled in the tool were explored. In Section 4.2, an overview of the methodology is given. A basic illustration of the methodology is also shown.

In Section 4.3, the proposed methodology is discussed in detail. This section has sub-sections. Each sub-section represents a stage in the development of concept for showing crash channel assignments. In these sub-sections, the different possibilities for the solution are discussed. Then, the most suitable approach is chosen. The requirements of the AssignmentLister tool is considered while choosing the approach. As a first step, the assignment information is extracted from the result file of the RSDBnext tool. This extracted information is stored to an XML file. Styles are applied using XSLT to this file, hence generating another XML file with styles. The first XML file consists of the assignment information. The second XML file consists of the assignment information, along with the styles which are applied accordingly. The reason for having two XML files in the intermediate step is also discussed. Then, this file is transformed to an Excel output. The Excel file consists of the crash channel assignments in the form of a table. The new methodology includes the use of XML, XSLT, and Excel. The reason for choosing these formats are also discussed in Section 4.3.

In this chapter, the different approaches, and the new methodology to visualize crash channel assignments are discussed. The following chapter consists of details regarding implementation of the AssignmentLister tool according to the proposed concept.

5 Implementation

This chapter consists of the implementation of the AssignmentLister tool. This tool is implemented based on the proposed concept. This chapter consists of three sections. The first section gives a general overview of the implementation, along with details regarding the programming language used to develop the tool. The second section gives a more detailed insight into the implementation. Different solutions for a certain problem are discussed. Following that, the most appropriate approach is chosen based on the requirements of the tool. The third section consists of a brief summary of the implementation chapter.

5.1 Overview of the Implementation

This section consists of a brief overview regarding the implementation of AssignmentLister tool. This tool basically shows crash channel assignments in a tabular form. The assignment information is extracted from the Result XML file of the RSDBnext tool. After extraction of the information, it is stored to an XML file. Then, styles are applied to this XML file using StyleSheet. The StyleSheet used for styling contains rules which define the formatting of the output Excel file. For example, the StyleSheet can be used to set the font, background color, cell borders, and alignment. For this, XSLT is used. The assignment information and the styles are then combined to create an XML file. This file is then converted to an Excel file. The Excel file is the final result from the AssignmentLister tool. This file consists of crash channel assignments in a tabular form.

The AssignmentLister tool is programmed in C#. There are some benefits of choosing C# with Visual Studio. Visual Studio has features like code completion, syntax highlighting, and automatic formatting options. This makes coding in C# fast and efficient. There is a significant developer group that uses C# with VS. As a result, there are many resources available. This makes it easier to identify solutions to issues. VS supports cross-platform development using the .NET Core framework. VS integrates with other development tools. This makes managing C# apps simple. Developers can add customized features and functionalities using extensions and plug-ins. The developer can modify the IDE to suit their specific requirements. [32]

Using C# is also beneficial while developing a tool which processes XML data, and generates Excel files through XSLT transformations. Along with the already mentioned reasons, the particular benefits of choosing C# for the development of the AssignmentLister tool are explained further.

C# provides type safety. This helps to reduce errors in the code. C# has built-in support to parse XML data. Since the assignment information is extracted by parsing the Result XML file, this helps to get the information in an easy manner. C# provides built-in support for XSLT transformations. This is useful while converting XML data to Excel format. The .NET framework library provides a set of libraries for working with XML data and Excel files. For example, Microsoft.Office.Interop.Excel namespaces can be used to access functionalities to work with XML and Excel data respectively. This is useful while working with complex XML data and StyleSheet (XSLT) transformations. These are some of the benefits of implementation using C# with VS. Overall, C# is a robust language for developing tools. It is suitable to use for the AssignmentLister tool as it has built-in support for parsing XML files and XSLT transformations.

For the development of the AssignmentLister tool in C#, various classes are created. Each class corresponds to a functionality. Table 5.1 provides an overview of the classes which are present in C#, as well as the functionality of each class.

Class (.cs)	Overview of the Functionality
Program	AssignmentLister tool is steered by the inputs
RSDB Program	Acts as a main method for the tool with input RSDB (.mdb file)
Start RSDB	RSDBnext tool is started, result.xml file is generated
Xml to List	Values are copied from the result.xml file to a list data structure
Channels in DB	Channels in the local database are got
Write to Xml	All the assignment information is written to the XML file
StyleSheet Processor	StyleSheet.xslt is opened, and saved to an Excel file

Table 5.1: Class Names with their Functionality

In the following section, details regarding the stages of implementation are discussed in three different sub-sections.

5.2 Stages of Implementation

The AB project contains sensors, which require data for simulation. Hence, crash channels are assigned to them. Every sensor in the project has a crash channel. This assignment is done for every crash present. The AssignmentLister tool shall show these crash channel assignments in a tabular form. The channel names are mapped to sensor locations. This is basically called the mapping information. Along with this information, the crash code, assignment comment, and channels present in the database are also important to be displayed in the final output.

The AssignmentLister tool is developed using C# with Visual Studio IDE. The reason for choosing C# with VS is discussed in Section 5.1. There are different stages during the implementation of the tool. This section consists of three sub-sections. These three sub-sections are divided according to the different stages in implementation. The stages are extracting assignment information, storing the assignments to a file and applying styles, and the conversion of the file to a tabular form.

In the first stage, the RSDBnext tool is started with the AssignmentLister tool. This generates the result files from the tool. The assignment information is then extracted from the result file of the tool. The second stage is the intermediate step. In this stage, the extracted information is stored to an internal data structure. Then, this information is written to an XML file. Styles are applied to this file using XSLT. Then, another XML file is created with information and styles. All of the necessary data for crash channel assignments is included in this XML file. The third stage is the last step. In this stage, the XML file is converted to an Excel file. All the information from the XML file with the styles is transformed to a tabular format. This Excel file consists of the assignment information as per requirements.

Figure 5.1 depicts these processes in detail. The three stages are marked with different colors in order to distinguish between them.

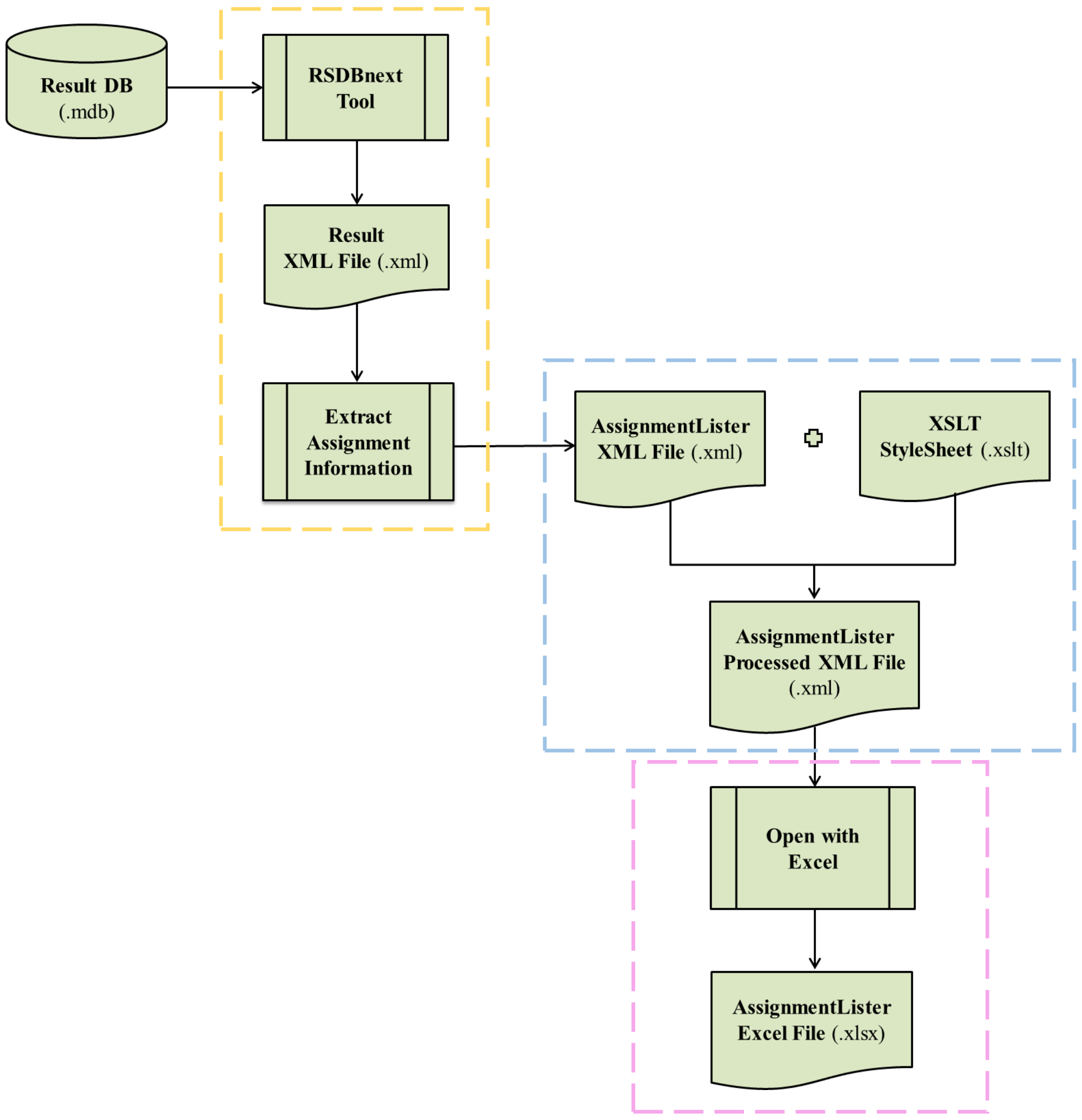

Result DB (.mdb)
RSDBnext Tool
Result XML File (.xml)
Extract Assignment Information
AssignmentLister XML File (.xml)
XSLT StyleSheet (.xslt)
AssignmentLister Processed XML File (.xml)
Open with Excel
AssignmentLister Excel File (.xlsx)

Figure 5.1: Stages of Implementation of the AssignmentLister Tool

Color	Depiction
🟨	Stage 1 - Extracting Assignment Information
🟦	Stage 2 - Storing the Assignments to a File & Applying Styles
🟪	Stage 3 - Conversion of the File to a Tabular Form

Table 5.2: Color Depiction of the Stages of Implementation

These are the three stages involved in the implementation of the AssignmentLister tool. Each stage corresponds to the following sub-sections.

In each sub-section, the colors used for the outlined boxes in the figures are based on Table 5.2. For example, in Section 5.2.1, the first stage is explained. Hence, yellow color is used to highlight some code parts in the figures. In this way, blue and lilac colors are used to highlight code parts for the second and third stages, in Sections 5.2.2 and 5.2.3 respectively.

5.2.1 Extracting Assignment Information

This is the first stage of implementation of the AssignmentLister tool. In this stage, the crash channel assignment information is extracted from the result XML file of RSDBnext tool.

There are some user requirements. According to these requirements, the AssignmentLister tool is implemented. The main requirement is for the output from the tool to have the information depicted in Table 5.3. So, the information should be extracted by the AssignmentLister tool in this stage. Some other formatting and styling options required are discussed further.

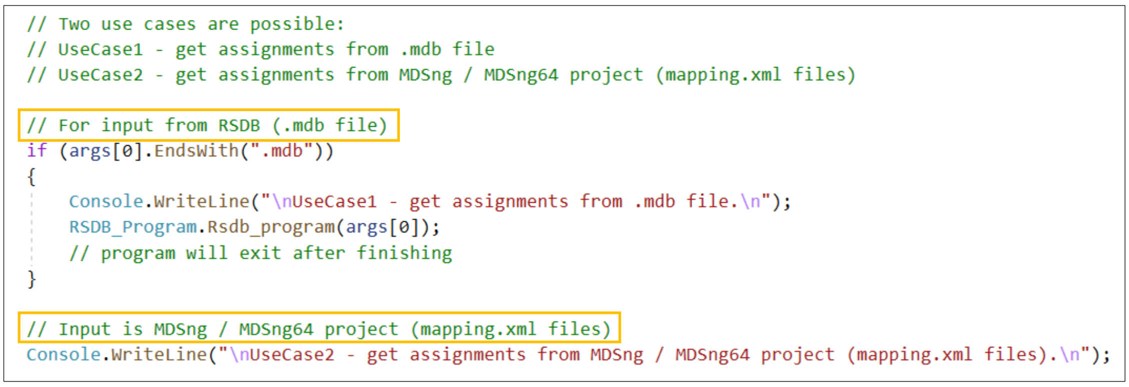
Information (for each crash)	**Meaning**
Crash ID	Unique crash identifier
Assignment Comment	Comment regarding the assignment
Sensor Location	Placement of a sensor in a vehicle
Channel Name	Identification name for a channel
Channels in the Database	All the channels which are present in the local DB

Table 5.3: Information to be Extracted by the AssignmentLister Tool

RSDB is the input for the RSDBnext tool. In order to generate the result files, the tool is started. For this, the .mdb file is given as input. Doing so generates the result files. Two result files are generated – result.xlsx and result.xml file. To get the crash channel assignment information, the result.xml file is used. The assignment information is extracted from this file in the AssignmentLister tool. For this, in the command line arguments in VS, the .mdb file path is given as input. The RSDBnext tool is run with this input in order to generate the result files.

Using the AssignmentLister tool, two different use cases are available. In one of the use cases, the assignment information is extracted from the complete simulation project. For this, assignments are got from the simulation project (MDSng). In the other use case, assignments are extracted from the simulation results. For this, assignments are got from the input, which is the RSDB (.mdb file).

To have a common solution for both the available use cases, the AssignmentLister tool is steered by the inputs accordingly, like in Figure 5.2. If the input is from RSDB, then the use case for input being an .mdb file is run. If the input is not an .mdb file, then it means it is the other use case. So, assignments are then got from the simulation project (MDSng). The use case which this thesis deals with is extracting assignment information from RSDB (.mdb file), to visualize crash channel assignments.

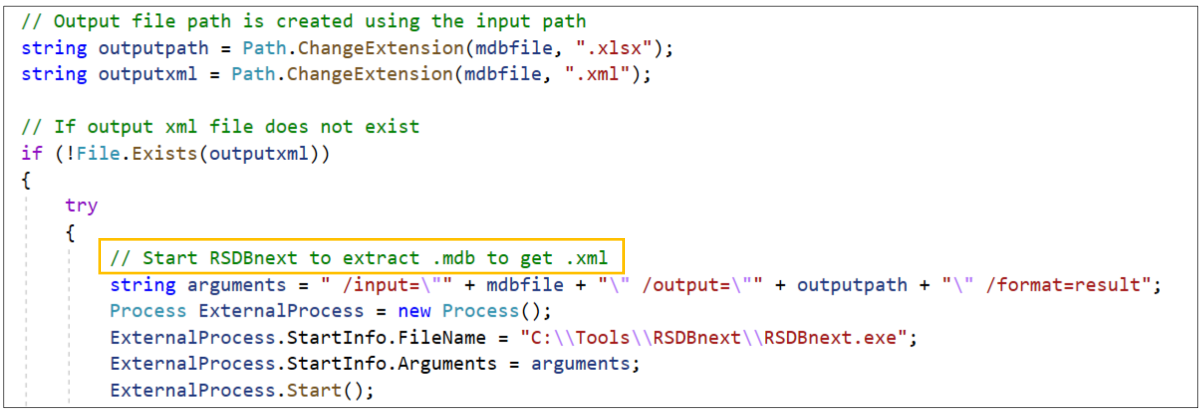


```
// Two use cases are possible:
// UseCase1 - get assignments from .mdb file
// UseCase2 - get assignments from MDSng / MDSng64 project (mapping.xml files)

// For input from RSDB (.mdb file)
if (args[0].EndsWith(".mdb"))
{
    Console.WriteLine("\nUseCase1 - get assignments from .mdb file.\n");
    RSDB_Program.Rsdb_program(args[0]);
    // program will exit after finishing
}

// Input is MDSng / MDSng64 project (mapping.xml files)
Console.WriteLine("\nUseCase2 - get assignments from MDSng / MDSng64 project (mapping.xml files).\n");
```

Figure 5.2: Code Snippet to Steer the AssignmentLister Tool by the Inputs

So, as a next step, considering the input is an .mdb file, the tool is started. As a result of this, two files are available in Excel and XML format respectively.

In Figure 5.3, the code snippet from the method to start the tool can be seen. First, the result file paths are created. Then, the tool is started to get the result.xml file. If the necessary result file already exists, then the tool is not started again. This step is then skipped.



```
// Output file path is created using the input path
string outputpath = Path.ChangeExtension(mdbfile, ".xlsx");
string outputxml = Path.ChangeExtension(mdbfile, ".xml");

// If output xml file does not exist
if (!File.Exists(outputxml))
{
    try
    {
        // Start RSDBnext to extract .mdb to get .xml
        string arguments = " /input=\"" + mdbfile + "\" /output=\"" + outputpath + "\" /format=result";
        Process ExternalProcess = new Process();
        ExternalProcess.StartInfo.FileName = "C:\\Tools\\RSDBnext\\RSDBnext.exe";
        ExternalProcess.StartInfo.Arguments = arguments;
        ExternalProcess.Start();
```

Figure 5.3: Starting the RSDBnext Tool to get the Result.xml File

The result files from the tool are available at the end of this step. The necessary assignment data must then be extracted from the result.xml file. The result.xml file consists of various crash related information in a hierarchical manner. To get the required information for the AssignmentLister tool, the result.xml file is first loaded. The file is then read in this stage to obtain the necessary data. For this, the hierarchy of the result.xml file has to be understood. Figure 5.4 shows the hierarchical structure of elements in the result.xml file.

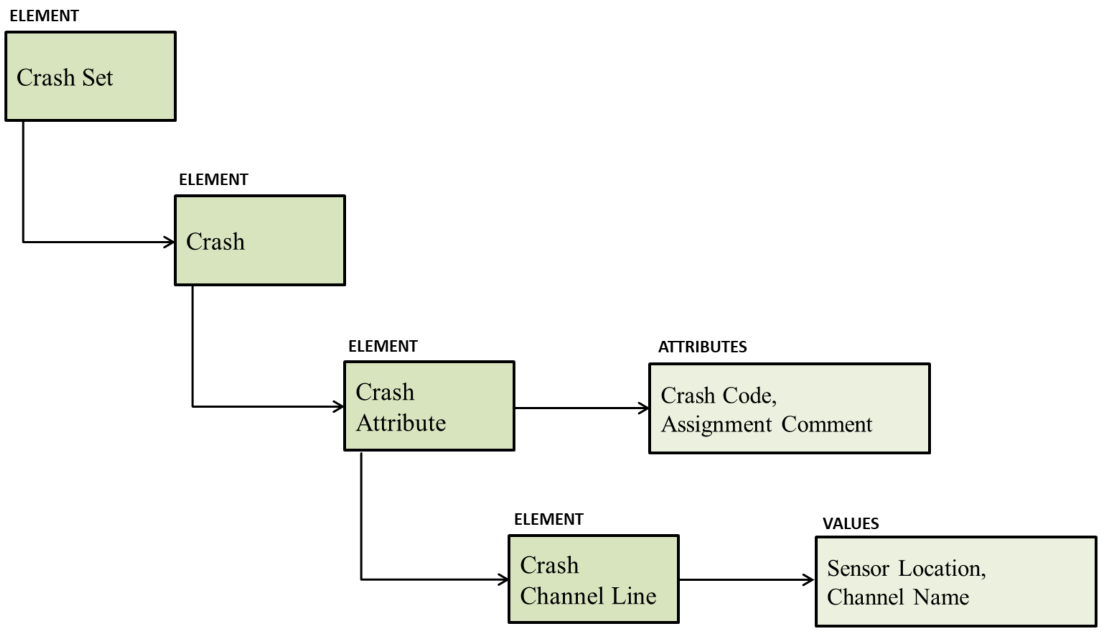
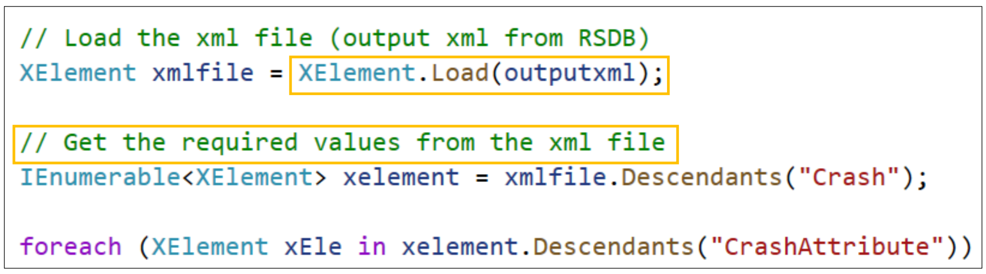


Figure 5.4: Hierarchy of Elements in the Input XML File for AssignmentLister

In Figure 5.5, the “outputxml” refers to the result.xml file, which is the output from the RSDB tool. This is the input file for the AssignmentLister tool. This file is loaded. Then, the file is read in order to get the required information.



```
// Load the xml file (output xml from RSDB)
XElement xmlfile = XElement.Load(outputxml);

// Get the required values from the xml file
IEnumerable<XElement> xelement = xmlfile.Descendants("Crash");

foreach (XElement xEle in xelement.Descendants("CrashAttribute"))
```

Figure 5.5: Loading the Result.xml File and Extracting Information

As a requirement, for information regarding channels in the database, the channels are got from the local database.

The steps which are implemented in the AssignmentLister tool to extract assignment information are summarized. The .mdb file is copied in order to start the RSDBnext tool. According to the input being used, the use cases are steered accordingly. As the input is an .mdb file, the RSDBnext tool is run. As a result of this, the result.xml file is available. Then, this result.xml file is read in order to get the necessary information for the AssignmentLister tool. For channels in the database information, the channels are got from the local database.

In the following step, the assignment information which is extracted is stored to a file. Then, styles are applied to it. The implementation of this stage is discussed in the next section.

5.2.2 Storing the Assignments to a File and Applying Styles

This is the second stage of implementation of the AssignmentLister tool. From the first stage, the assignment information is extracted. The assignment information should be saved to a data structure in this step. Then, from this data structure, the information is written to an XML file. Then, styles are applied using XSLT StyleSheet. As a result, another XML file is created. The way in which these intermediate steps to develop the AssignmentLister tool are implemented is explained further.

As a first step in this stage, the assignment information is stored in an internal structure. Some of the structures in which the information can be stored are stacks, queues, arrays and lists. Each data structure has its own characteristics with advantages and disadvantages. The better data structure depends on the use case and requirements. The available data structures which are possible options are briefly discussed. Among these, the most suited data structure is selected. The data structures are:

a) Stacks, Queues
b) Arrays
c) Lists

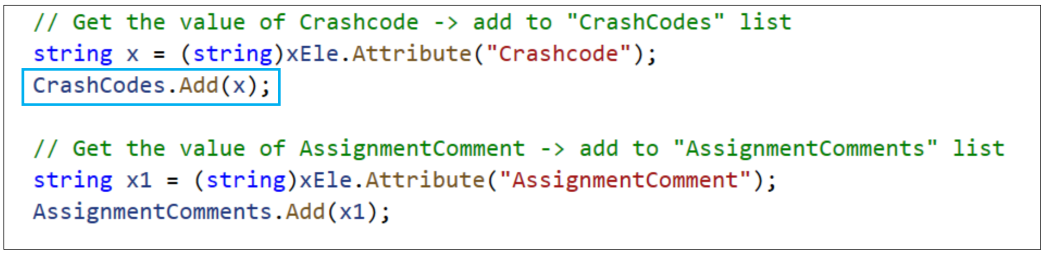
Option (a): If a specific order of elements is to be maintained or elements are to be accessed in a FIFO (First In, First Out) manner, stacks and queues can be suitable.

Option (b): Arrays have a fixed size. The number of elements an array can hold should be known when it is created. Element addition and deletion are difficult since each element must be moved. This makes the process complex. Hence, performance of the tool is affected in this case. In C#, arrays can store elements of a specific data type only.

Option (c): Using list data structure gives flexibility to add or remove elements. Lists can adjust its internal structure accordingly and have built-in methods to add, remove and sort elements. This helps to lower the lines of code needed in order to manipulate the list. Lists can store elements of different data types. They are simple to implement. Also, data in the list can easily be sorted.

For the stated reasons, lists are more appropriate to use in this case. The number of elements is not fixed and changes with each project. It is necessary to be able to access the elements and manipulate them according to the requirements in the next steps. Hence, option (c) is the data structure which is chosen to store the required information amongst the other two structures.

The chosen internal structure to do this is a list. The assignment information which is needed for the final output consists of crash codes, assignment comment, and the crash channel assignment mapping information. So, different lists are created in order to store this data. Having different lists for each of the different information needed makes implementation easier for the next steps. The attribute value of the element is got, and this is added to a list. For example, the attribute value of all crash codes is got. This is then added to the "CrashCodes" list. This can be seen in Figure 5.6.



```
// Get the value of Crashcode -> add to "CrashCodes" list
string x = (string)xEle.Attribute("Crashcode");
CrashCodes.Add(x);

// Get the value of AssignmentComment -> add to "AssignmentComments" list
string x1 = (string)xEle.Attribute("AssignmentComment");
AssignmentComments.Add(x1);
```

Figure 5.6: Code Snippet to Add the Required Values to a List Data Structure

At the end of this step, all the required information is stored in the list data structure. This information will now be written to an XML file as the following step in this stage. For this, the structure of the XML file is first defined. The XML structure is kept the same for both the available use cases of the AssignmentLister tool. Then, all the information in the lists is written to the XML file in the defined structure.

In order to write to XML, first a path is created for the AssignmentLister.xml file. There are a few requirements for the way the information appears in the XML file. In order to have the data in the required form, certain string manipulations are done. In the result.xml file, the sensor locations and channel names are available as a single string value. This is split into two different values for sensor location and channel name respectively. This is then added to two separate lists (one list for sensor locations, the other for channel names). This helps to keep the information in the required format for later steps.

Zero emulation is when a crash does not have a channel assigned for a specific sensor. Sometimes, the crash channel may not be mapped to a sensor. In this case, the channel name is marked as “Zero Emulation” in the result.xml file. This indicates that no channels have been assigned. When this is the case, the requirement is for the zero emulation to be shown as “---:-” in the AssignmentLister tool. So, the channels with the name “Zero Emulation” are rewritten to “---:-”.

These requirements are implemented. Then, the AssignmentLister.xml should be created. For this, all the information in the lists is written to the .xml file in a structured way. To write to XML in C#, there are different possibilities. These possibilities are discussed, and a suitable option is selected. The options to write to XML are:

a) XmlDocument
b) XmlWriter

Option (a): XmlDocument represents a whole XML document. This option to write to XML is used to create a new .xml document, or to load to an existing .xml file. XmlDocument requires many lines of code to write to an XML file. This makes the code more complex. XmlDocument uses a lot of memory for large XML files.

Option (b): XmlWriter writes directly to a stream. This method provides an option to easily set attributes, write elements, and define namespaces. XmlWriter can handle small and large XML files in an efficient way. XmlWriter is simpler than the other option. Additionally, fewer lines of code are needed.

Hence, XmlWriter, which is option (b), is a better option in this case amongst the possible methods. XmlWriter is used to write all the information needed to the AssignmentLister.xml file. The elements, along with their attributes are written to the .xml file according to the requirements.

The hierarchy of elements defined in the AssignmentLister.xml file can be seen in Figure 5.7.

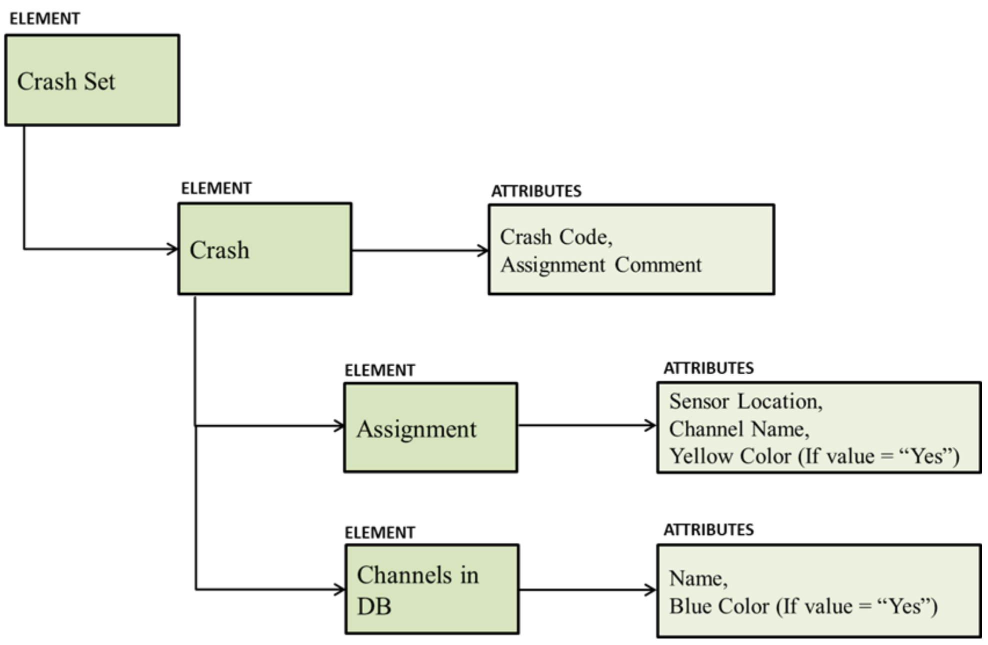
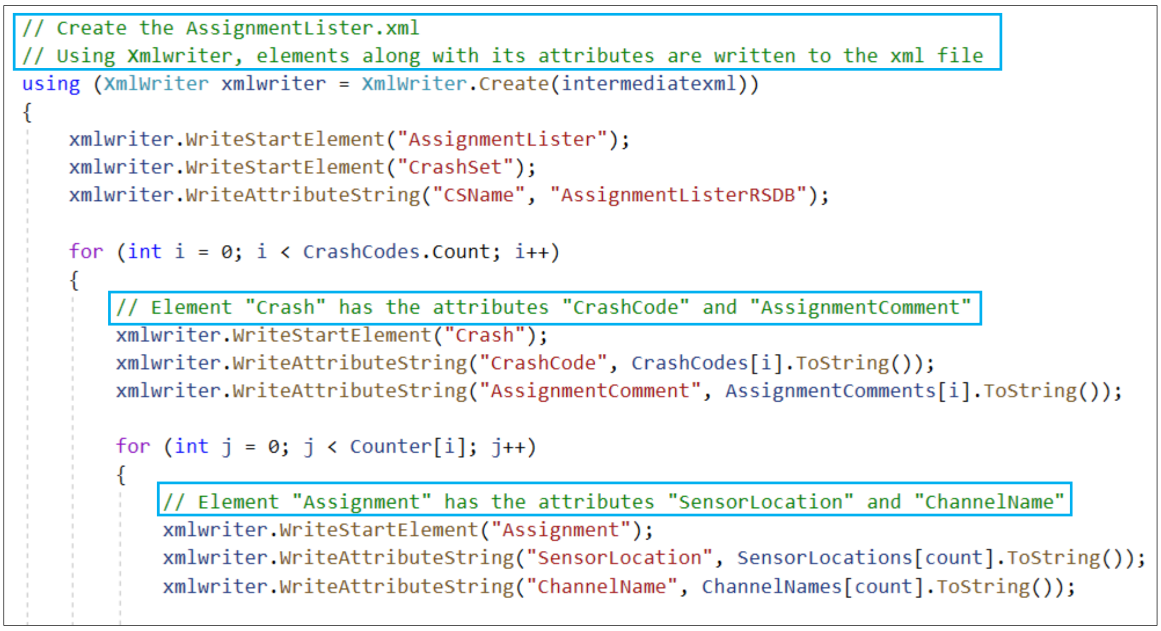


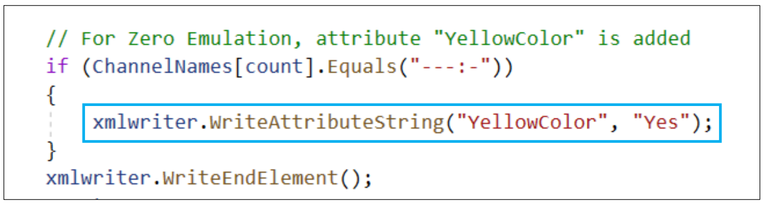
Figure 5.7: Hierarchy of Elements in the AssignmentLister.xml File



```
// Create the AssignmentLister.xml
// Using Xmlwriter, elements along with its attributes are written to the xml file
using (XmlWriter xmlwriter = XmlWriter.Create(intermediatexml))
{
    xmlwriter.WriteStartElement("AssignmentLister");
    xmlwriter.WriteStartElement("CrashSet");
    xmlwriter.WriteAttributeString("CSName", "AssignmentListerRSDB");

    for (int i = 0; i < CrashCodes.Count; i++)
    {
        // Element "Crash" has the attributes "CrashCode" and "AssignmentComment"
        xmlwriter.WriteStartElement("Crash");
        xmlwriter.WriteAttributeString("CrashCode", CrashCodes[i].ToString());
        xmlwriter.WriteAttributeString("AssignmentComment", AssignmentComments[i].ToString());

        for (int j = 0; j < Counter[i]; j++)
        {
            // Element "Assignment" has the attributes "SensorLocation" and "ChannelName"
            xmlwriter.WriteStartElement("Assignment");
            xmlwriter.WriteAttributeString("SensorLocation", SensorLocations[count].ToString());
            xmlwriter.WriteAttributeString("ChannelName", ChannelNames[count].ToString());
```

Figure 5.8: Writing to the AssignmentLister.xml File

A few additional attributes are also added for styling purpose. The requirement is for the channel names with zero emulation to be marked with yellow background color. So, an attribute "YellowColor" is added for zero emulation as depicted in Figure 5.9.



```
// For Zero Emulation, attribute "YellowColor" is added
if (ChannelNames[count].Equals("---:-"))
{
    xmlwriter.WriteAttributeString("YellowColor", "Yes");
}
xmlwriter.WriteEndElement();
```

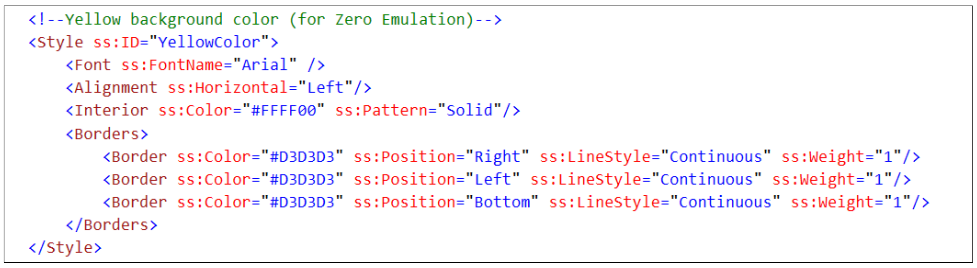
Figure 5.9: Code Snippet to Add the Color Attribute

If a crash has a particular channel mapped to a sensor, then this is called a used channel in the database. The requirement is for these used channels to be marked with blue color. So, an attribute “BlueColor” is added for the used channels. This is implemented in a similar way like the yellow background color.

The reason for adding these attributes is to help style the document. This is explained in detail further. So, at the end of this step, all of this information is written to the AssignmentLister.xml file. The user requirements are considered and implemented too. The next step in this stage is to style the document using XSLT StyleSheets.

There are many formatting and styling options which are required in the final tabular form with the assignment information. For this reason, XSLT StyleSheets are used for the purpose of styling. Styles are used for various options. Some of the styles used as per requirements are font names in Arial, alignment of text in the cells to the left (cell styles), yellow background color for zero emulation, and blue text color for used channels (conditional formatting styles).

An example of the styles in StyleSheet.xslt class is shown in Figure 5.10.

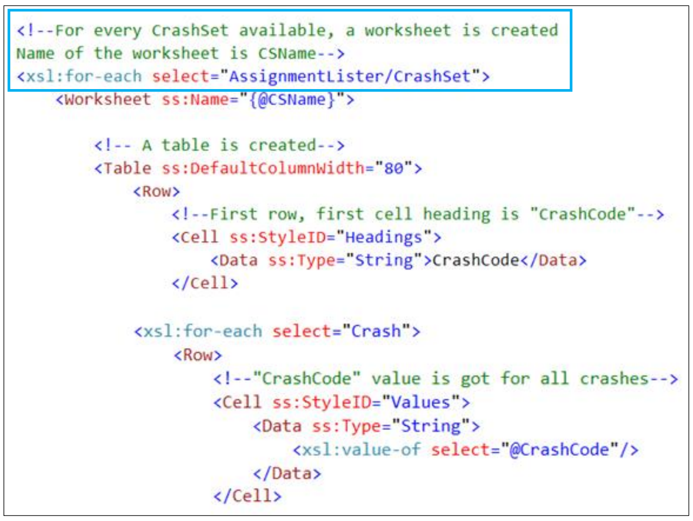


```
<!--Yellow background color (for Zero Emulation)-->
<Style ss:ID="YellowColor">
    <Font ss:FontName="Arial" />
    <Alignment ss:Horizontal="Left"/>
    <Interior ss:Color="#FFFF00" ss:Pattern="Solid"/>
    <Borders>
        <Border ss:Color="#D3D3D3" ss:Position="Right" ss:LineStyle="Continuous" ss:Weight="1"/>
        <Border ss:Color="#D3D3D3" ss:Position="Left" ss:LineStyle="Continuous" ss:Weight="1"/>
        <Border ss:Color="#D3D3D3" ss:Position="Bottom" ss:LineStyle="Continuous" ss:Weight="1"/>
    </Borders>
</Style>
```

Figure 5.10: Example of Styles in StyleSheet.xslt

For the use case with input as .mdb file, there is only one crash set available. This crash set consists of many crashes. In the other use case for the AssignmentLister tool, there are many crash sets available. Each crash set consists of many crashes. The number of crashes depends on the airbag project.

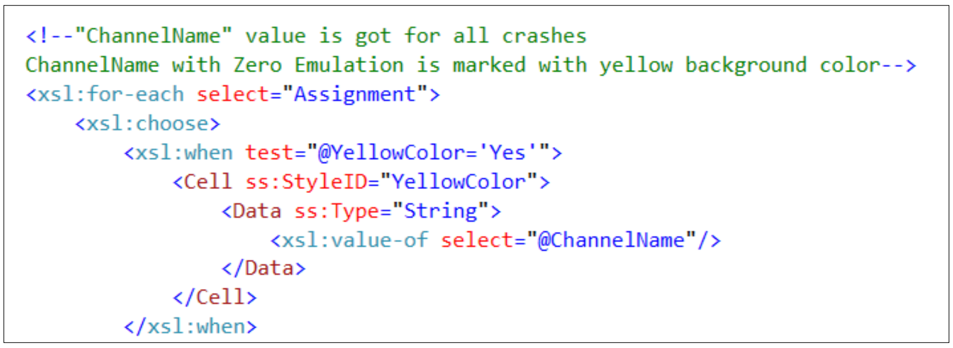
For the StyleSheet to be used for both the use cases, a “for-each” loop is used in the .xslt class. The number of worksheets in the final Excel file corresponds to the number of crash sets available for the project. For example, for the use case with .mdb file as input, as there is only one crash set, there is only one worksheet available in the final Excel output. The headings and the corresponding values for the cells are written to the .xml file. An example for this is shown in Figure 5.11.



```
<!--For every CrashSet available, a worksheet is created
Name of the worksheet is CSName-->
<xsl:for-each select="AssignmentLister/CrashSet">
    <Worksheet ss:Name="{@CSName}">

        <!-- A table is created-->
        <Table ss:DefaultColumnWidth="80">
            <Row>
                <!--First row, first cell heading is "CrashCode"-->
                <Cell ss:StyleID="Headings">
                    <Data ss:Type="String">CrashCode</Data>
                </Cell>

            <xsl:for-each select="Crash">
                <Row>
                    <!--"CrashCode" value is got for all crashes-->
                    <Cell ss:StyleID="Values">
                        <Data ss:Type="String">
                            <xsl:value-of select="@CrashCode"/>
                        </Data>
                    </Cell>
```

Figure 5.11: Creating a Worksheet and Table in .xslt

If the channel name has the attribute “YellowColor” in the .xml file, then the Style ID for YellowColor is applied. This is done for channel names having zero emulation. Yellow background color is applied to the channel name in this case. If there is no zero emulation, then yellow background color is not applied to the cell in this case. An example of the way this is implemented using .xslt is shown in Figure 5.12.

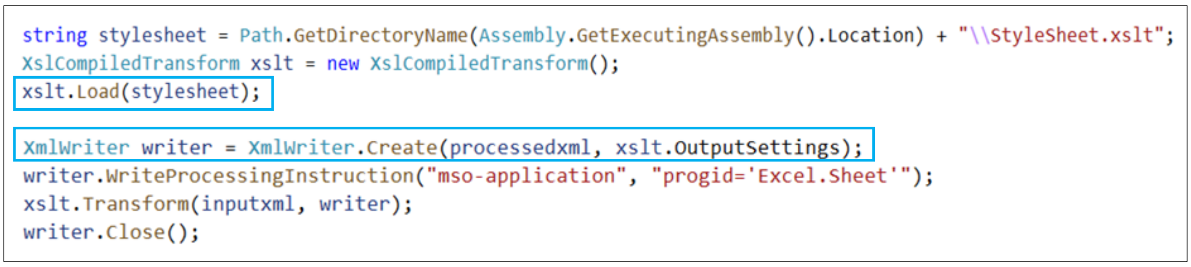


```
<!--"ChannelName" value is got for all crashes
ChannelName with Zero Emulation is marked with yellow background color-->
<xsl:for-each select="Assignment">
    <xsl:choose>
        <xsl:when test="@YellowColor='Yes'">
            <Cell ss:StyleID="YellowColor">
                <Data ss:Type="String">
                    <xsl:value-of select="@ChannelName"/>
                </Data>
            </Cell>
        </xsl:when>
```

Figure 5.12: Code Snippet to Mark a Cell with Background Color in .xslt

If the channel name has the attribute “BlueColor” in the .xml file, then the Style ID for BlueColor is applied. This is done for used channels in the database. If the channel is not used, then it is not marked with blue color in this case. This is implemented in a similar way like the yellow background color. In this way, styles are applied using StyleSheet.xslt as per requirements.

The AssignmentLister.xml consists of the required information. Styles are applied using StyleSheet.xslt on this file. This creates another XML file, called the Processed.xml. To do this, the StyleSheet path is got, and the .xslt file is loaded. Then, the Processed.xml file is created with values from AssignmentLister.xml, and styles from StyleSheet.xslt. The Processed.xml file consists of the required information along with the styles applied for each value respectively.



```
string stylesheet = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location) + "\\StyleSheet.xslt";
XslCompiledTransform xslt = new XslCompiledTransform();
xslt.Load(stylesheet);

XmlWriter writer = XmlWriter.Create(processedxml, xslt.OutputSettings);
writer.WriteProcessingInstruction("mso-application", "progid='Excel.Sheet'");
xslt.Transform(inputxml, writer);
writer.Close();
```

Figure 5.13: Code Snippet for XSLT Transformation in C#

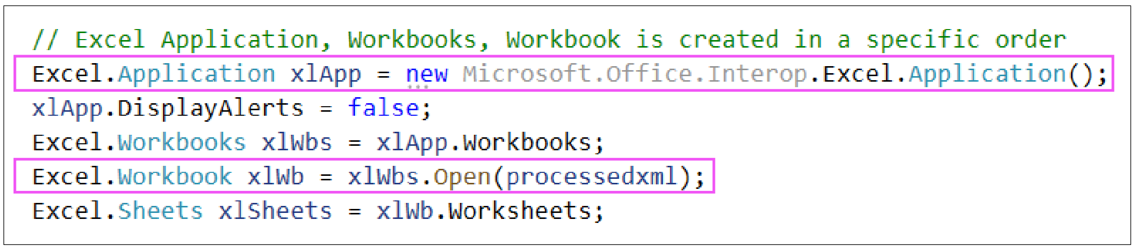
The steps which are implemented in the AssignmentLister tool to store assignments to a file and apply styles are summarized. The extracted assignment information is stored to different lists. This information is written from the lists to an XML file (AssignmentLister.xml) after defining the XML structure. The document is styled using XSLT (StyleSheet.xslt). Another XML file (Processed.xml) is created with the information and styles together.

In the following step, the assignment information with styles (Processed.xml) is converted to a tabular form. The implementation of this stage is explained in sub-section 5.2.3.

5.2.3 Conversion of the File to a Tabular Form

This is the last stage of implementation of the AssignmentLister tool. From the intermediate stage, the Processed.xml file is available with the information and styles. In this stage, the assignment information is converted to an Excel file. The Processed.xml file is read, and saved as AssignmentLister.xlsx.

As a first step, the Excel application is created using “Microsoft.Office.Interop.Excel”, which is a namespace in C#. The namespace requires that MS Excel is installed on the machine. It provides a number of tools for working with Excel files. It gives access to the Excel application's properties. Using this, developers can create and manipulate Excel workbooks. A wide range of operations like transforming files to Excel, and formatting can be done within C# applications. So, the Excel application is created in a specific order. Following that, workbooks, workbook and worksheets is created. The processed.xml file is also opened here. In the next steps, this file is saved as the AssignmentLister.xlsx file.



```
// Excel Application, Workbooks, Workbook is created in a specific order
Excel.Application xlApp = new Microsoft.Office.Interop.Excel.Application();
xlApp.DisplayAlerts = false;
Excel.Workbooks xlWbs = xlApp.Workbooks;
Excel.Workbook xlWb = xlWbs.Open(processedxml);
Excel.Sheets xlSheets = xlWb.Worksheets;
```

Figure 5.14: Creating an Excel Application and Loading the Processed.xml File

There are some more formatting and styling features which are to be implemented according to the requirements. There is an option of implementing these features using XSLT StyleSheet or C# code. Both the options are considered. Then, the better option for the additional styling required is selected. The options being considered to implement the additional styles are:

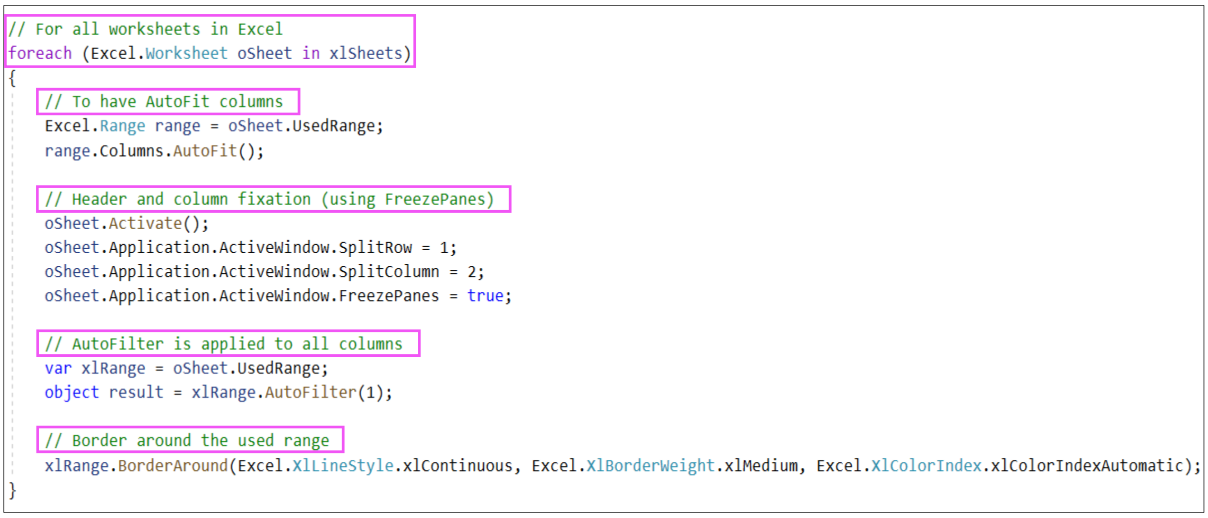
a) In StyleSheet
b) In C# code

Both XSLT and C# have their pros and cons. The selection of the method to be used depends on various factors. The complexity of the styling requirements is one of the key elements in this issue.

XSLT is capable of handling complex transformations. It is also used for many of the styling features for the AssignmentLister tool. Implementing complex styling requirements is possible using XSLT. Though the styles can be implemented in XSLT, the code will get more complex using this method. Many lines of code are needed to add the specific styles. This leads to a complex code. C# is also capable of handling some of the styling requirements. Implementing the specific styling requirements is not so complex using C#. Using C#, the implementation for the styles is quite simple, requiring very few lines of code in comparison to XSLT. C# provides more flexibility to create customized styling features in this case. It is also easier to apply the desired formatting options to multiple worksheets in Excel automatically. Hence, it is the better option, considering the styling requirements and their complexity.

In Figure 5.15, the “foreach” loop is used. This means that these options are available for each worksheet in Excel. With the use case having .mdb file as input, there is only one worksheet. For the other use case, there are multiple worksheets available in the Excel output. Using this loop makes the features available for both the use cases easily. The styling features are implemented based on the specifications. They are auto-fit columns, header and column fixation, auto-filter option, and borders.

For the used range of the column in Excel, there is an auto-fit width option implemented. The size of the column is modified based on the size of the values in it when this option is selected. Header and column fixation is achieved using the “FreezePanes” option. One row and two columns should be fixed as per requirements. The remaining information should not be fixed. An option to apply filters to each column should be available. This is achieved using the auto-filter option. A border is available around the entire Excel table. The way these options are implemented can be seen in Figure 5.15.



```
// For all worksheets in Excel
foreach (Excel.Worksheet oSheet in xlSheets)
{
    // To have AutoFit columns
    Excel.Range range = oSheet.UsedRange;
    range.Columns.AutoFit();

    // Header and column fixation (using FreezePanes)
    oSheet.Activate();
    oSheet.Application.ActiveWindow.SplitRow = 1;
    oSheet.Application.ActiveWindow.SplitColumn = 2;
    oSheet.Application.ActiveWindow.FreezePanes = true;

    // AutoFilter is applied to all columns
    var xlRange = oSheet.UsedRange;
    object result = xlRange.AutoFilter(1);

    // Border around the used range
    xlRange.BorderAround(Excel.XlLineStyle.xlContinuous, Excel.XlBorderWeight.xlMedium, Excel.XlColorIndex.xlColorIndexAutomatic);
}
```

Figure 5.15: Screenshot of Code for Formatting and Styling Feature in C#

After this, the processed.xml file, which is read for the assignment information and styles is saved as an Excel file. The formatting and styling options implemented are also added to the Excel file. This file, which is the final output, is available in the form of a table. Then, the Excel objects which are created are released in a specific order.

The steps which are implemented in the AssignmentLister tool to convert the Processed.xml file to a tabular form are summarized. The Processed.xml file is read to get the assignment information along with styles. Some formatting and styling options like auto-fit column and auto-filter are implemented using C# code. The Processed.xml file is saved as an Excel file (AssignmentLister.xlsx). The AssignmentLister.xlsx consists of crash channel assignment information in a tabular form as per user requirements.

This is the last step in the AssignmentLister tool's development. The tool is implemented according to the proposed concept.

5.3 Summary

This chapter consists of details about the implementation of the AssignmentLister tool. It starts with an overview of the implementation in the first section. The next parts cover the various stages of the tool's implementation. The stages are described across three sub-sections. Implementation of the tool to get crash channel assignments is explained in the first sub-section. In the next sub-section, the way in which assignments are stored to the XML file is explained. This XML file is called the AssignmentLister.xml. Then, styles are added to this file using StyleSheet.xslt. The AssignmentLister.xml file in combination with StyleSheet.xslt creates another XML file. This XML file is the Processed.xml. The final sub-section describes the way Processed.xml is converted to AssignmentLister.xlsx. In the AssignmentLister.xlsx, the crash channel assignments are displayed in the form of a table as per requirements. This Excel file is the final output from the AssignmentLister tool. The next chapter discusses the results and evaluation of the AssignmentLister tool.

6 Results and Evaluation

This chapter discusses the results of the AssignmentLister tool, along with its evaluation. The chapter is divided into four sections, the first of which provides an overview in general. The second section discusses the intermediate results from the tool, and the final result. The final result displays crash channel assignments in a tabular form. The tool is evaluated using various testing methods. A comparison of the old and new AssignmentLister tool is presented in the third section. This chapter ends with a summary.

6.1 Overview of the Results

The AssignmentLister tool is implemented to show crash channel assignments. The .mdb file is the input to the AssignmentLister tool. This input starts the RSDBnext tool, which generates the Result.xml and Result.xlsx files. The Result.xml file is used as an input file for the AssignmentLister tool. First, the XML file (AssignmentLister.xml) with all the assignment information is created. Then, another XML file (Processed.xml) with the information and styles is created. Following this, the Excel file (AssignmentLister.xlsx) is created. This Excel file consists of the crash channel assignments which are displayed in a tabular form.

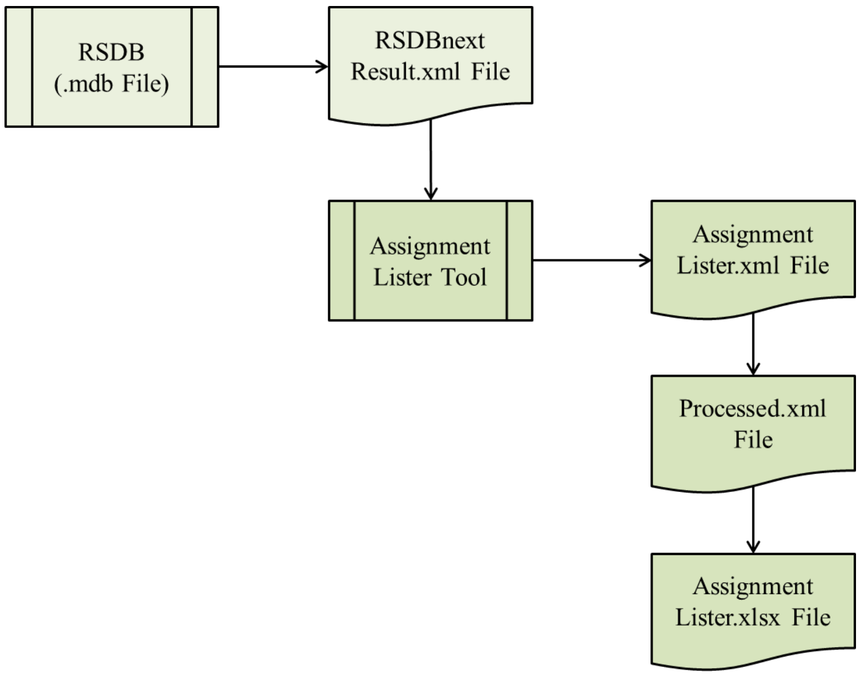
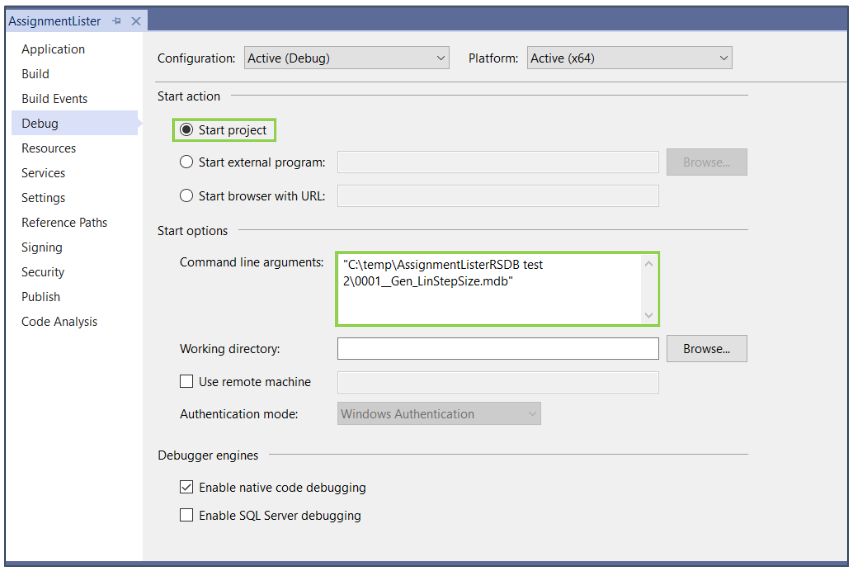


Figure 6.1: Overview of the Results from AssignmentLister Tool

To start the AssignmentLister tool, the input .mdb file path should be given. This is depicted in Figure 6.2.



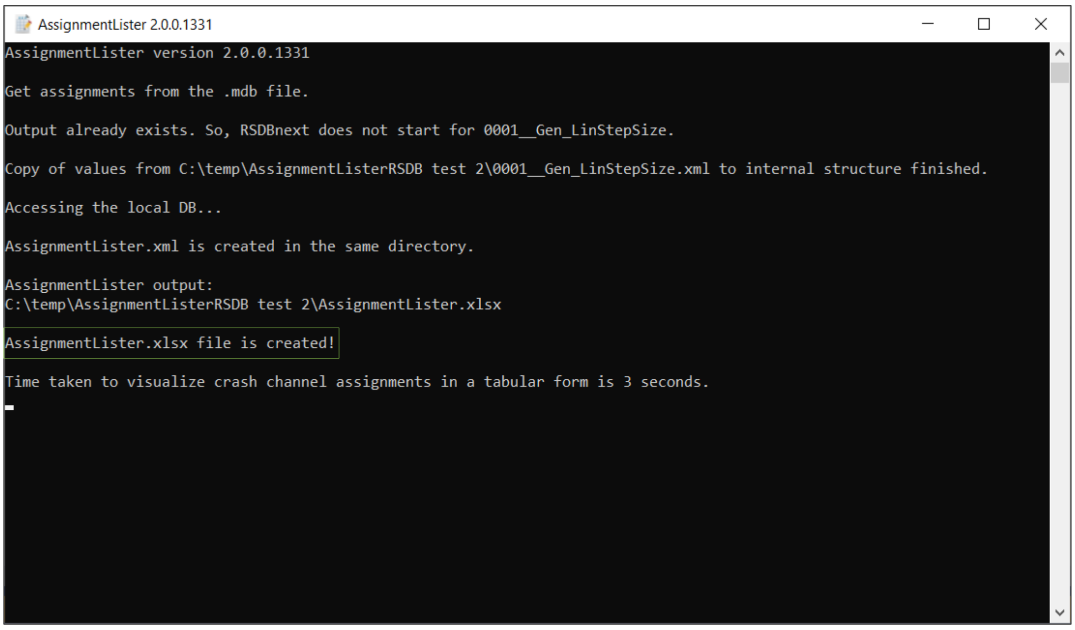
Start action: [✓ Start project]

Command line arguments: "C:\temp\AssignmentListerRSDB test 2\0001__Gen_LinStepSize.mdb"

[✓] Enable native code debugging

Figure 6.2: Input File Path to Start the Tool in Visual Studio

When the AssignmentLister tool is run with the .mdb file, a console window appears. Figure 6.3 shows this console window.



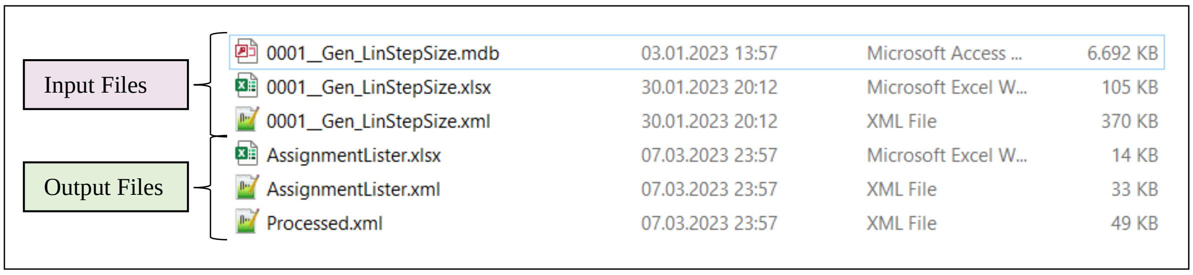
```
AssignmentLister version 2.0.0.1331

Get assignments from the .mdb file.

Output already exists. So, RSDBnext does not start for 0001__Gen_LinStepSize.

Copy of values from C:\temp\AssignmentListerRSDB test 2\0001__Gen_LinStepSize.xml to internal structure finished.

Accessing the local DB...

AssignmentLister.xml is created in the same directory.

AssignmentLister output:
C:\temp\AssignmentListerRSDB test 2\AssignmentLister.xlsx

AssignmentLister.xlsx file is created!

Time taken to visualize crash channel assignments in a tabular form is 3 seconds.
```

Figure 6.3: Screenshot of the Console Window for AssignmentLister Tool

The steps which are performed to visualize crash channel assignments are summarized. The AssignmentLister tool starts the RSDBnext tool. The .mdb file path is the input to start the

RSDBnext tool. This generates the result.xlsx and result.xml files. If the result.xml file is already present from the last run, this step is omitted. The values from the result.xml file of RSDBnext are copied to a list data structure. The local database is accessed to get the information about channels which are in the database. This is a necessity for the final output. Then, an XML file (AssignmentLister.xml) is created in the same directory, with all the required data. Following this step, styles are applied to the AssignmentLister.xml file. This generates another XML file (Processed.xml) in the same directory. The final step is to create the AssignmentLister.xlsx file. This Excel file displays crash channel assignments in a tabular form. The Excel file is in the same location as the XML files. All the files generated are present in the same directory. The time taken to generate these files, in order to visualize the crash channel assignments is 3 seconds (for the example of the airbag project shown in Figure 6.3).

After running the AssignmentLister tool, the output files are available in the same directory. An example of the files available at the same location can be seen in Figure 6.4. The time of creation of the files, along with the type and size of the file for the particular airbag project can also be seen.



0001__Gen_LinStepSize.mdb	03.01.2023 13:57	Microsoft Access ...	6.692 KB
0001__Gen_LinStepSize.xlsx	30.01.2023 20:12	Microsoft Excel W...	105 KB
0001__Gen_LinStepSize.xml	30.01.2023 20:12	XML File	370 KB
AssignmentLister.xlsx	07.03.2023 23:57	Microsoft Excel W...	14 KB
AssignmentLister.xml	07.03.2023 23:57	XML File	33 KB
Processed.xml	07.03.2023 23:57	XML File	49 KB

Figure 6.4: Example of Files in the Directory

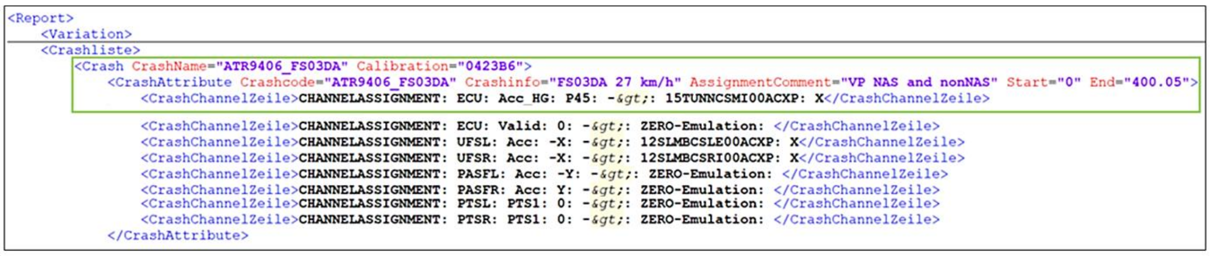
The next section shows the intermediate XML results, and the final Excel res ` in a tabular form.

6.2 Visualization of Crash Channel Assignments in a Tabular Form

The AssignmentLister tool is run with an airbag project to visualize the crash channel assignments. The generation of the interim results from the AssignmentLister tool is included in this section, along with the final outcome. After the extraction of assignment information, all the data is stored to an XML file. This XML file is called the AssignmentLister.xml. When styles are applied to this XML file using XSLT Stylesheets, another XML file is created. This

XML file is called the Processed.xml. These two XML files are shown in this section. The final Excel output is then displayed and evaluated.

When the RSDBnext tool is started, the result files from the tool are generated. An example consisting of a part of this file can be seen in Figure 6.5. For a particular "**Crash**", there are "**CrashName**" and "**Calibration**" attributes. These attributes have values. For the parent element "**Crash**", the child element is "**CrashAttribute**". For this element, there are attributes like "**Crashcode**", "**Crashinfo**", and "**AssignmentComment**". The "**CrashChannelZeile**" is another element, consisting of the channel assignments for a crash. There are many channel assignments for a crash. In each crash channel line, the first part is the sensor information. The latter part in the line is the assigned channel for that particular sensor. For example, for the first channel assignment, "**ECU: Acc_HG: P45**" is information about the sensor. It is basically the sensor name, direction, and value, which is separated by colons. "**15TUNNCSMI00ACXP: X**" is the channel name. This format follows for every crash channel line.



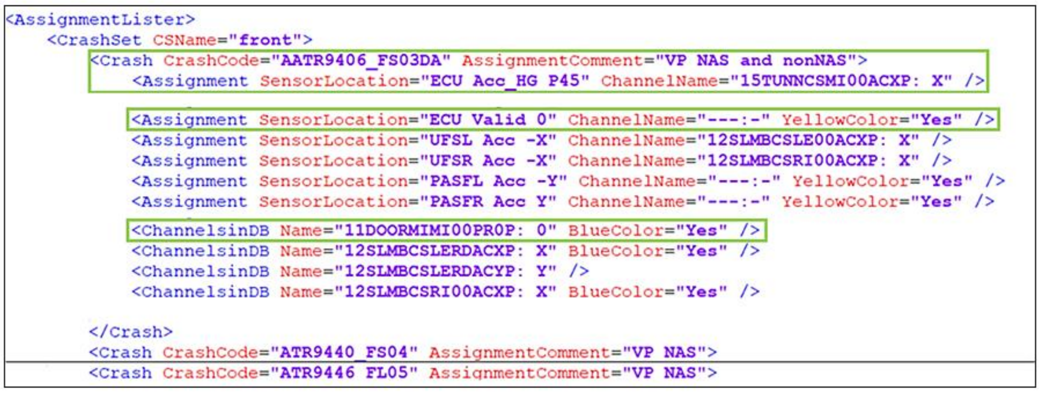
```
<Report>
    <Variation>
    <Crashliste>
        <Crash CrashName="ATR9406_FS03DA" Calibration="0423B6">
            <CrashAttribute Crashcode="ATR9406_FS03DA" Crashinfo="FS03DA 27 km/h" AssignmentComment="VP NAS and nonNAS" Start="0" End="400.05">
                <CrashChannelZeile>CHANNELASSIGNMENT: ECU: Acc_HG: P45: -&gt;: 15TUNNCSMI00ACXP: X</CrashChannelZeile>

                <CrashChannelZeile>CHANNELASSIGNMENT: ECU: Valid: 0: -&gt;: ZERO-Emulation: </CrashChannelZeile>
                <CrashChannelZeile>CHANNELASSIGNMENT: UFSL: Acc: -X: -&gt;: 12SLMBCSLE00ACXP: X</CrashChannelZeile>
                <CrashChannelZeile>CHANNELASSIGNMENT: UFSR: Acc: -X: -&gt;: 12SLMBCSRI00ACXP: X</CrashChannelZeile>
                <CrashChannelZeile>CHANNELASSIGNMENT: PASFL: Acc: -Y: -&gt;: ZERO-Emulation: </CrashChannelZeile>
                <CrashChannelZeile>CHANNELASSIGNMENT: PASFR: Acc: Y: -&gt;: ZERO-Emulation: </CrashChannelZeile>
                <CrashChannelZeile>CHANNELASSIGNMENT: PTSL: PTS1: 0: -&gt;: ZERO-Emulation: </CrashChannelZeile>
                <CrashChannelZeile>CHANNELASSIGNMENT: PTSR: PTS1: 0: -&gt;: ZERO-Emulation: </CrashChannelZeile>
            </CrashAttribute>
```

Figure 6.5: Screenshot of the Result.xml File

The result.xml file consists of the information required for the AssignmentLister tool. This information is extracted, and stored in the AssignmentLister.xml file in the required format. An example consisting of a part of this AssignmentLister.xml file can be seen in Figure 6.6.

A crash set consists of many crashes. Each crash consists of assignment information. For a particular "**Crash**", the attributes are "**CrashCode**" and "**AssignmentComment**". These attributes have values. For the parent element "**Crash**", the child element is "**Assignment**". For this element, there two attributes for "**SensorLocation**", "**ChannelName**". There are many channel assignments for a crash. This information is extracted from the result.xml file, and is stored in a defined structure. This structure is defined based on the requirement for the next steps. For example, the sensor name, direction, and value, which were separated by colons in the result.xml file is separated by just a space in the AssignmentLister.xml file. This structure is the same for every assignment line. Some assignments also have the attribute

"**YellowColor**". This indicates that the channel name for that particular assignment is "**---:-**", which is Zero-Emulation. In this case, the cell color should be marked with yellow. After the assignment information, an element "**ChannelsinDB**" can be seen. Channels in the database is the list of channels extracted from the local database. This element has the atrribute "**Name**". Some channels in db have the attribute "**BlueColor**". This shows that the channel is used. In this case, the text color in the cell should be marked with blue as per requirements.



```xml
<AssignmentLister>
    <CrashSet CSName="front">
        <Crash CrashCode="AATR9406_FS03DA" AssignmentComment="VP NAS and nonNAS">
            <Assignment SensorLocation="ECU Acc_HG P45" ChannelName="15TUNNCSMI00ACXP: X" />

            <Assignment SensorLocation="ECU Valid 0" ChannelName="---:-" YellowColor="Yes" />
            <Assignment SensorLocation="UFSL Acc -X" ChannelName="12SLMBCSLE00ACXP: X" />
            <Assignment SensorLocation="UFSR Acc -X" ChannelName="12SLMBCSRI00ACXP: X" />
            <Assignment SensorLocation="PASFL Acc -Y" ChannelName="---:-" YellowColor="Yes" />
            <Assignment SensorLocation="PASFR Acc Y" ChannelName="---:-" YellowColor="Yes" />
            <ChannelsinDB Name="11DOORMIMI00PROP: 0" BlueColor="Yes" />
            <ChannelsinDB Name="12SLMBCSLERDACXP: X" BlueColor="Yes" />
            <ChannelsinDB Name="12SLMBCSLERDACYP: Y" />
            <ChannelsinDB Name="12SLMBCSRI00ACXP: X" BlueColor="Yes" />

        </Crash>
        <Crash CrashCode="ATR9440_FS04" AssignmentComment="VP NAS">
        <Crash CrashCode="ATR9446 FL05" AssignmentComment="VP NAS">
```

Figure 6.6: Screenshot of the AssignmentLister.xml File

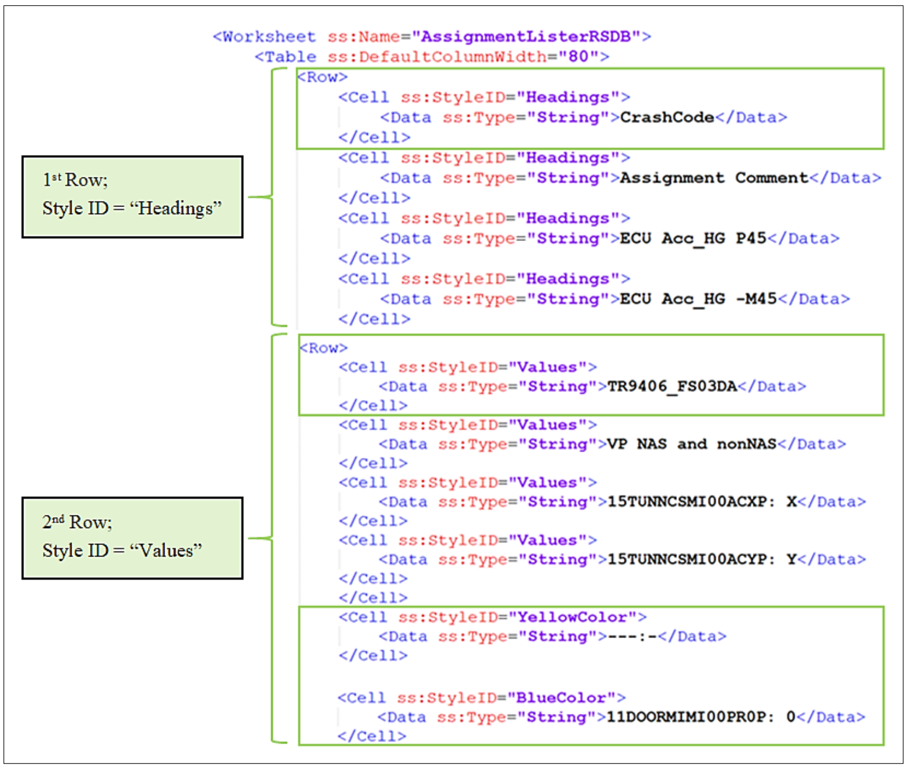
The AssignmentLister.xml file consists of all the required information defined in a particular structure for the AssignmentLister tool. Styles are applied to this file using StyleSheet.xslt. This creates the Processed.xml file. The Processed.xml file consists of the values with the styles. Figure 6.7 depicts an example of this file.

The Processed.xml file consists of the styles for the Excel table. The row and column styles are specified for the particular values accordingly. For example, "**Headings**" is the Style ID in the first row. This Style ID is defined in the beginning of the StyleSheet.xslt file. It consists of style settings like alignment, font, borders, and colors. Based on the necessary requirements, these settings are defined.

The Style ID "Headings" is applied for the heading "CrashCode". The other headings in the Excel table appear in the first row itself. They also have the same Style ID. All Style IDs are defined in the StyleSheet.xslt file. The values which are filled up below the headings in the rest of the rows have the Style ID "**Values**". These values are filled up for each row in the Excel table one after the other, after the first row.

The tag **<Row>** depicts the start of a new row in the table. **<Cell>** stands for the column in the table. After the <Row> tag, the first <Cell> tag stands for the first column, the next

<Cell> tag stands for the second column, and so on. Each row stands for a single crash and its assignment information. For the values which should have yellow background color and blue text color, the Style ID “**YellowColor**” and “**BlueColor**” are assigned respectively. The Style ID for yellow color has the style to add background color to the cell. The Style ID for blue color has the style to add blue text color to the cell. These styles are applied based on the tags present in the AssignmentLister.xml file. For example, if a particular value has the attribute yellow color to it in the AssignmentLister.xml file, then the Style ID “YellowColor” is applied to that value. Each value which appears in the table has a style assigned to it in the Processed.xml file.



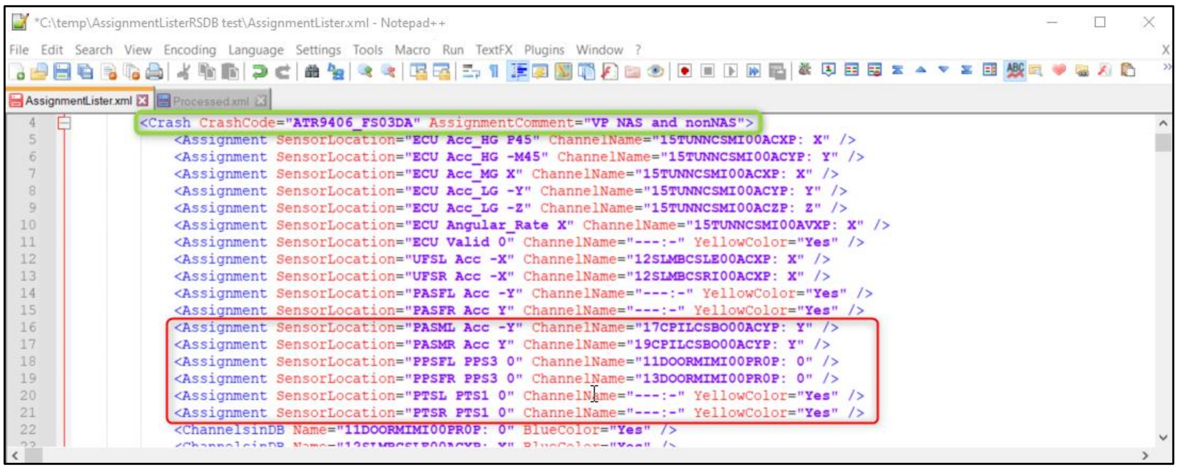
```
<Worksheet ss:Name="AssignmentListerRSDB">
    <Table ss:DefaultColumnWidth="80">
        <Row>
            <Cell ss:StyleID="Headings">
                <Data ss:Type="String">CrashCode</Data>
            </Cell>
            <Cell ss:StyleID="Headings">
                <Data ss:Type="String">Assignment Comment</Data>
            </Cell>
            <Cell ss:StyleID="Headings">
                <Data ss:Type="String">ECU Acc_HG P45</Data>
            </Cell>
            <Cell ss:StyleID="Headings">
                <Data ss:Type="String">ECU Acc_HG -M45</Data>
            </Cell>
        <Row>
            <Cell ss:StyleID="Values">
                <Data ss:Type="String">TR9406_FS03DA</Data>
            </Cell>
            <Cell ss:StyleID="Values">
                <Data ss:Type="String">VP NAS and nonNAS</Data>
            </Cell>
            <Cell ss:StyleID="Values">
                <Data ss:Type="String">15TUNNCSMI00ACXP: X</Data>
            </Cell>
            <Cell ss:StyleID="Values">
                <Data ss:Type="String">15TUNNCSMI00ACYP: Y</Data>
            </Cell>
            </Cell>
            <Cell ss:StyleID="YellowColor">
                <Data ss:Type="String">---:-</Data>
            </Cell>

            <Cell ss:StyleID="BlueColor">
                <Data ss:Type="String">11DOORMIMI00PROP: 0</Data>
            </Cell>
```

Figure 6.7: Example of Styles Added to the Data in the Processed.xml File

The Processed.xml file is then transformed to the Excel file. The AssignmentLister.xlsx is available in the same directory as the other files. The Processed.xml file can also be opened with Excel. When this XML file is opened with Excel, the Excel file with crash channel assignments is available. The AssignmentLister.xlsx file is the final output of the AssignmentLister tool. It consists of crash channel assignments in a tabular form. All the formatting and styling requirements are also added to this Excel file.

An example of the way information is transformed from the AssignmentLister.xml to the AssignmentLister.xlsx is depicted in Figure 6.8 and Figure 6.9 respectively. The Processed.xml file is also created for the transformation.

In Figure 6.6, the attributes for crash can be seen (green box). The attributes for assignments can also be seen (red box). For channel names with zero emulation (“---:-”), the attribute YellowColor = “Yes”. These XML elements and attributes are defined such that it is easy for transformation to the Excel file as per requirements.

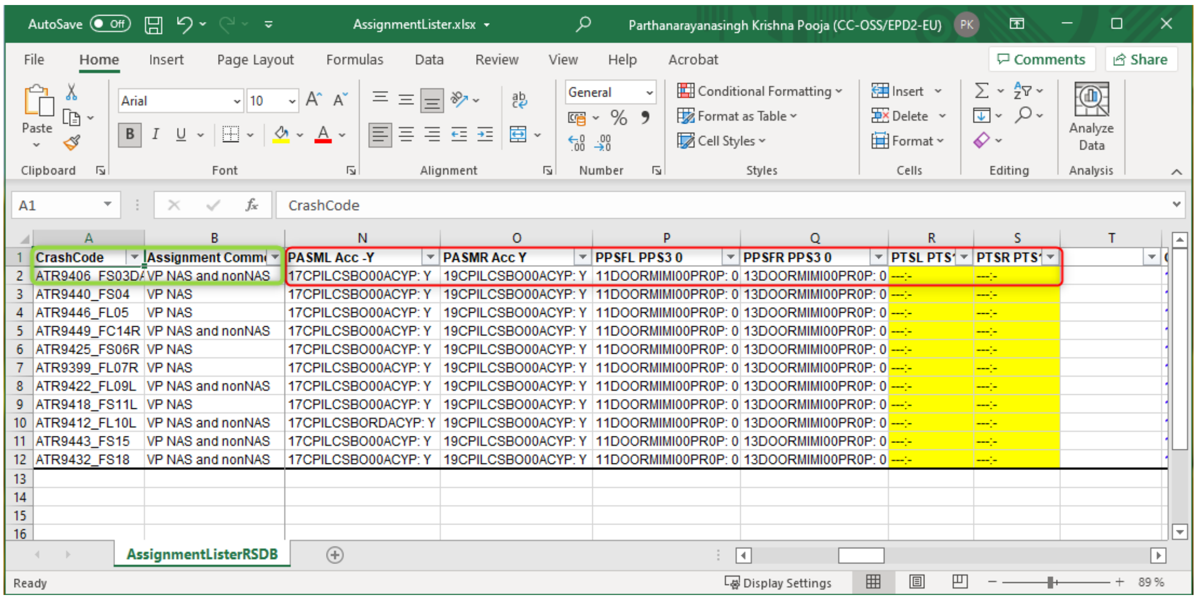

*C:\temp\AssignmentListerRSDB test\AssignmentLister.xml - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run TextFX Plugins Window ?
AssignmentLister.xml
Processed.xml

```
<Crash CrashCode="ATR9406_FS03DA" AssignmentComment="VP NAS and nonNAS">
    <Assignment SensorLocation="ECU Acc_HG P45" ChannelName="15TUNNCSMI00ACXP: X" />
    <Assignment SensorLocation="ECU Acc_HG -M45" ChannelName="15TUNNCSMI00ACYP: Y" />
    <Assignment SensorLocation="ECU Acc_MG X" ChannelName="15TUNNCSMI00ACXP: X" />
    <Assignment SensorLocation="ECU Acc_LG -Y" ChannelName="15TUNNCSMI00ACYP: Y" />
    <Assignment SensorLocation="ECU Acc_LG -Z" ChannelName="15TUNNCSMI00ACZP: Z" />
    <Assignment SensorLocation="ECU Angular_Rate X" ChannelName="15TUNNCSMI00AVXP: X" />
    <Assignment SensorLocation="ECU Valid 0" ChannelName="---:-" YellowColor="Yes" />
    <Assignment SensorLocation="UFSL Acc -X" ChannelName="12SLMBCSLE00ACXP: X" />
    <Assignment SensorLocation="UFSR Acc -X" ChannelName="12SLMBCSRI00ACXP: X" />
    <Assignment SensorLocation="PASFL Acc -Y" ChannelName="---:-" YellowColor="Yes" />
    <Assignment SensorLocation="PASFR Acc Y" ChannelName="---:-" YellowColor="Yes" />
    <Assignment SensorLocation="PASML Acc -Y" ChannelName="17CPILCSBO00ACYP: Y" />
    <Assignment SensorLocation="PASMR Acc Y" ChannelName="19CPILCSBO00ACYP: Y" />
    <Assignment SensorLocation="PPSFL PPS3 0" ChannelName="11DOORMIMI00PR0P: 0" />
    <Assignment SensorLocation="PPSFR PPS3 0" ChannelName="13DOORMIMI00PR0P: 0" />
    <Assignment SensorLocation="PTSL PTS1 0" ChannelName="---:-" YellowColor="Yes" />
    <Assignment SensorLocation="PTSR PTS1 0" ChannelName="---:-" YellowColor="Yes" />
    <ChannelsinDB Name="11DOORMIMI00PR0P: 0" BlueColor="Yes" />
```

Figure 6.8: Snippet from the AssignmentLister.xml File

In Figure 6.9, the same information which is in the XML file is depicted in the required format in Excel. The first row is the headings. “Crash Code” and “Assignment Comment” are the first two headings, and their values for the particular crash is below the headings (in the first two cells of the second row).

The rest of the headings which can be seen in the Excel file are the “SensorLocation” values. Corresponding to these headings, the “ChannelName” values are present. This mapping of crash channels to the sensors is called the mapping information, or the assignment information. The yellow background color is marked for the cell in case of zero emulation. This is also depicted in the Excel file. This is the method for converting XML data to Excel. Some more details regarding the final Excel file, and the requirements is provided further.



CrashCode	Assignment Comm	PASML Acc -Y	PASMR Acc Y	PPSFL PPS3 0	PPSFR PPS3 0	PTSL PTS	PTSR PTS
ATR9406_FS03D	VP NAS and nonNAS	17CPILCSBO00ACYP: Y	19CPILCSBO00ACYP: Y	11DOORMIMI00PR0P: 0	13DOORMIMI00PR0P: 0	---:-	---:-
ATR9440_FS04	VP NAS	17CPILCSBO00ACYP: Y	19CPILCSBO00ACYP: Y	11DOORMIMI00PR0P: 0	13DOORMIMI00PR0P: 0	---:-	---:-
ATR9446_FL05	VP NAS	17CPILCSBO00ACYP: Y	19CPILCSBO00ACYP: Y	11DOORMIMI00PR0P: 0	13DOORMIMI00PR0P: 0	---:-	---:-
ATR9449_FC14R	VP NAS and nonNAS	17CPILCSBO00ACYP: Y	19CPILCSBO00ACYP: Y	11DOORMIMI00PR0P: 0	13DOORMIMI00PR0P: 0	---:-	---:-
ATR9425_FS06R	VP NAS	17CPILCSBO00ACYP: Y	19CPILCSBO00ACYP: Y	11DOORMIMI00PR0P: 0	13DOORMIMI00PR0P: 0	---:-	---:-
ATR9399_FL07R	VP NAS	17CPILCSBO00ACYP: Y	19CPILCSBO00ACYP: Y	11DOORMIMI00PR0P: 0	13DOORMIMI00PR0P: 0	---:-	---:-
ATR9422_FL09L	VP NAS and nonNAS	17CPILCSBO00ACYP: Y	19CPILCSBO00ACYP: Y	11DOORMIMI00PR0P: 0	13DOORMIMI00PR0P: 0	---:-	---:-
ATR9418_FS11L	VP NAS	17CPILCSBO00ACYP: Y	19CPILCSBO00ACYP: Y	11DOORMIMI00PR0P: 0	13DOORMIMI00PR0P: 0	---:-	---:-
ATR9412_FL10L	VP NAS and nonNAS	17CPILCSBORDACYP: Y	19CPILCSBO00ACYP: Y	11DOORMIMI00PR0P: 0	13DOORMIMI00PR0P: 0	---:-	---:-
ATR9443_FS15	VP NAS and nonNAS	17CPILCSBO00ACYP: Y	19CPILCSBO00ACYP: Y	11DOORMIMI00PR0P: 0	13DOORMIMI00PR0P: 0	---:-	---:-
ATR9432_FS18	VP NAS and nonNAS	17CPILCSBO00ACYP: Y	19CPILCSBO00ACYP: Y	11DOORMIMI00PR0P: 0	13DOORMIMI00PR0P: 0	---:-	---:-

Figure 6.9: Part of the AssignmentLister.xlsx File

The example in the figures shows a part of the AssignmentLister.xml file, and the way in which information is transformed to the AssignmentLister.xlsx file in the required format.

Another example of the Excel file can be seen in Figure 6.10. This example shows the complete Excel output with all the headers. The first row in the Excel table consists of the headings. After that, each row corresponds to a single crash. The headings, along with the corresponding data in the Excel table are as follows:

a) Crash Code: It is the first column in the table. It consists of different crash IDs present in the airbag project.

b) Assignment Comment is the column after "Crash Code" in the Excel table.

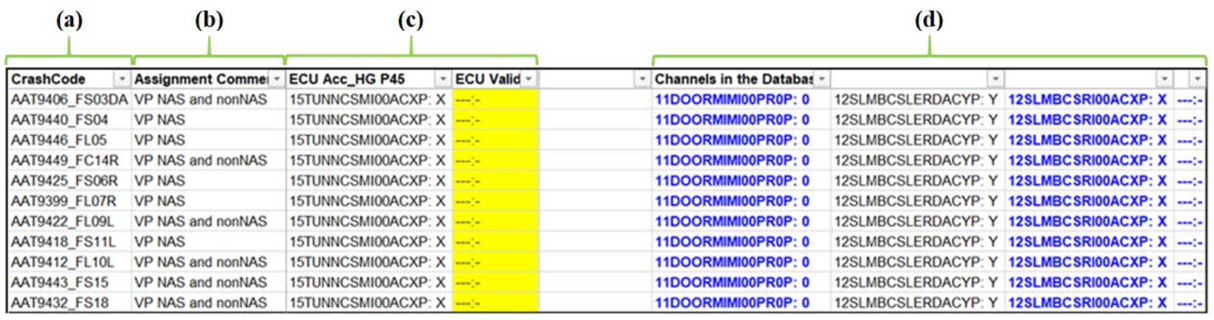
c) Sensor Information: There are many sensors present in the airbag project. The column headings after assignment comment are filled with the sensor information. The sensor name, direction, and value are separated by a space in this information. For example, "ECU Acc_HG P45". The number of sensors present depends on the project. Crash channels are assigned to the sensors. The name of the assigned crash channels is filled up in the columns corresponding to the sensor headings. For example, "15TUNNCSMI00ACXP: X". Every sensor present in the project has a crash channel. This is done for each crash present in the airbag project. This information is basically the crash channel assignment information.

d) Channels in the Database: After the sensor information, a blank column is left. This is done as it is a requirement to have a separation between the channel assignments and channels from the database. Following the blank column, the heading is channels in the database. This consists of the channels present in the local database for each crash.

The way in which the assignment information is displayed in the Excel table is according to the user requirements.

There are also some formatting and styling options which are implemented. Some of the styling features which can be seen in Figure 6.10 are bold font, filter option, auto-fit width, colors, and borders.

All the headings in the table are marked with bold font. Filter option is available for the headings. Using the filters, information in the column can be sorted if required. The columns in the Excel table are sized to fit the information. The cells with channel names as zero emulation ("---:--") have yellow background color. Blue text color is applied to used channels in the database. If a channel is assigned to a sensor for a particular crash, then it is called as a used channel. In the Excel table, for the cells filled with the assignment information, a border is applied around the entire used range of the table. These are the styles applied to the Excel table.



CrashCode	Assignment Comme	ECU Acc_HG P45	ECU Valid		Channels in the Databas			
AAT9406_FS03DA	VP NAS and nonNAS	15TUNNCSMI00ACXP: X	---:-		11DOORMIMI00PR0P: 0	12SLMBCSLERDACYP: Y	12SLMBCSRI00ACXP: X	---:-
AAT9440_FS04	VP NAS	15TUNNCSMI00ACXP: X	---:-		11DOORMIMI00PR0P: 0	12SLMBCSLERDACYP: Y	12SLMBCSRI00ACXP: X	---:-
AAT9446_FL05	VP NAS	15TUNNCSMI00ACXP: X	---:-		11DOORMIMI00PR0P: 0	12SLMBCSLERDACYP: Y	12SLMBCSRI00ACXP: X	---:-
AAT9449_FC14R	VP NAS and nonNAS	15TUNNCSMI00ACXP: X	---:-		11DOORMIMI00PR0P: 0	12SLMBCSLERDACYP: Y	12SLMBCSRI00ACXP: X	---:-
AAT9425_FS06R	VP NAS	15TUNNCSMI00ACXP: X	---:-		11DOORMIMI00PR0P: 0	12SLMBCSLERDACYP: Y	12SLMBCSRI00ACXP: X	---:-
AAT9399_FL07R	VP NAS	15TUNNCSMI00ACXP: X	---:-		11DOORMIMI00PR0P: 0	12SLMBCSLERDACYP: Y	12SLMBCSRI00ACXP: X	---:-
AAT9422_FL09L	VP NAS and nonNAS	15TUNNCSMI00ACXP: X	---:-		11DOORMIMI00PR0P: 0	12SLMBCSLERDACYP: Y	12SLMBCSRI00ACXP: X	---:-
AAT9418_FS11L	VP NAS	15TUNNCSMI00ACXP: X	---:-		11DOORMIMI00PR0P: 0	12SLMBCSLERDACYP: Y	12SLMBCSRI00ACXP: X	---:-
AAT9412_FL10L	VP NAS and nonNAS	15TUNNCSMI00ACXP: X	---:-		11DOORMIMI00PR0P: 0	12SLMBCSLERDACYP: Y	12SLMBCSRI00ACXP: X	---:-
AAT9443_FS15	VP NAS and nonNAS	15TUNNCSMI00ACXP: X	---:-		11DOORMIMI00PR0P: 0	12SLMBCSLERDACYP: Y	12SLMBCSRI00ACXP: X	---:-
AAT9432_FS18	VP NAS and nonNAS	15TUNNCSMI00ACXP: X	---:-		11DOORMIMI00PR0P: 0	12SLMBCSLERDACYP: Y	12SLMBCSRI00ACXP: X	---:-

Figure 6.10: Excel Table Displaying Crash Channel Assignment Information

Along with the styles discussed, header and column fixation is also applied to the Excel table. Header and column fixation means that the first row, and the first two columns of the table are fixed. Rest of the columns in the Excel table are dynamic. This style is indicated by the scroll bars in Figure 6.9. These are the formatting and styling features which are available. The final output from the AssignmentLister tool is available in a tabular format in Excel. This table displays the crash channel assignment information as required.

The tabular format is important for the users to see if the assignment information is correct. It can be checked if the mapping is fitting properly or not. User requirements are considered during the concept and implementation phase. Expected behavior of the AssignmentLister tool from users is taken into account. Tests are conducted to ensure the tool meets its requirements.

The type of tests conducted to evaluate the AssignmentLister tool are performance tests, unit tests, integration tests, functional tests and usability tests [33]. Performance tests is conducted to evaluate the performance of the tool with different projects. Different airbag projects are available. There are different cases present with different projects. So, this testing is done to ensure the tool works according to the requirements in all different scenarios. Unit tests are conducted to test each unit of code. These tests are performed manually. The aim of unit testing is to identify errors early in the development process of the tool. This helps to make sure the tool works as intended. Integration testing tests the working of different units of code together. Different modules of the software are tested in order to ensure they are functioning correctly. This testing is done after unit testing. Conducting these tests ensures the code is functioning correctly, and that different modules of software work together as expected. This helps to identify any issues early in the development process. Functional tests ensures that the software is performing the functions which it is supposed to, and is meeting user requirements. In functional testing, manual testing involves running the software with various inputs, and checking the resulting outputs. Usability tests are conducted to evaluate the ease of use of the tool. It helps to identify issues users may face while using the tool. This helps to ensure the tool meets user expectations.

The aim of testing software is to check if the functionality of the tool is correct, and if user requirements are met. These tests are conducted during, and after the development of the AssignmentLister tool. These tests make sure the functionality of the tool is correct, and the Excel output is as expected. From these tests, it can be seen that the Excel table is according to the user requirements. The formatting and styling options are also added correctly based on the specifications of the user. The AssignmentLister tool works correctly for different inputs, and produces the required output with the crash channel assignment information for both the use cases. From these Excel tables generated for different inputs, it can be seen that the AssignmentLister tool satisfies user requirements and expectations.

The AssignmentLister tool also works correctly with other tools in the toolchain. The existing toolchain architecture is not disturbed. The tool is integrated into the toolchain as expected, as the Excel table is created correctly without affecting the functionality of other airbag application tools present.

6.3 Comparison with the Old Solution

The old solution to get crash channel assignments was complex. Excel Macros were used to develop the tool. Using macros made the tool slow and unstable. The assignment information was not handled well. The user had to perform many manual steps in the form of mouse clicks to get the final output. The number of mouse clicks to be performed manually is 5. There were also other issues while waiting for the output to be generated. MS Office Products could not be used while waiting for the output. Sometimes, the page even stopped responding in this case. For the example shown in Section 6.2, creation of the Excel file took more than 30 seconds. If the file size is larger than the one chosen for the example, then it can take more than a minute for the Excel file to be generated using macros. There was no common solution available for the use cases.

The new solution developed for the AssignmentLister tool does not use Excel Macros to visualize crash channel assignments. C# programming with Visual Studio IDE is used to develop the tool. XML file format is used to handle the assignment information in the intermediate steps. Styles are applied to the XML file using XSLT StyleSheet. Then, the file is converted to Excel format.

The transformation from XML to Excel happens in just a few seconds. The tool handles assignment information in an easier way compared to the the old solution. The file formats used to handle the tool make the process of visualizing crash channel assignments easy. The hierarchical structure of XML helps to transform data, to create the Excel file. The desired formatting and styling options are applied to the final output with the help of StyleSheet. The final output is in a tabular form as per user requirements, and is generated in a stable manner. There are no negative effects of using MS Office Products with the new solution.

The user does not have to wait for a long time, or do multiple mouse clicks to get the output file. The user has to just enter the input file path to run the tool. The number of mouse clicks to be performed manually is just 1. The tool handles the remaining steps automatically. For the example shown in Section 6.2, the Excel file was generated in just 3 seconds. The assignment information is extracted from the input.xml file (result.xml), and even the local database (for channels in db) in this time frame itself.

A common solution is available for the available use cases. The tool is developed considering the requirements of both the use cases. The tool starts for either of the use cases based on the input provided. The StyleSheet is implemented such that it is flexible to be used for both the use cases. Depending on the settings in the StyleSheet, the Excel file is generated for the use cases. For the use case running with the result file of the simulation (.mdb file), only one worksheet is required. For the other use case, number of worksheets are required, depending on the airbag project used. For single or multiple worksheets, the tool runs successfully and displays crash channel assignments in the required format. Hence, the AssignmentLister tool works in a stable, easier and faster manner.

Some additional features are also added to the new AssignmentLister tool. In the old tool, these options were not available. The “Filter” option is added to the final Excel file. The user can filter the required information in the columns with this option. “Auto-fit width” option is also added as a new feature to the Excel file. This feature formats the column width to the text content present automatically. As the channel names are long, this feature helps the user to see the crash channel assignments in an easy way. “Yellow background color” is added to the cells having zero emulation. Zero emulation is when there are no channels assigned for a sensor. The user can check if there are no channels assigned for a sensor. This can be easily identified in the table because of the coloring option added.

The primary goal to get rid of macros is achieved with the new AssignmentLister tool. The primary goal of the tool is dependent on a few other steps, like handling the information and transferring it to a tabular format. The tool handles assignment information well. This is proven from the stability of the tool while generating the output in a tabular form. The file format selected for storing and displaying crash channel assignments is good-fitting for the requirements of the tool.

Using the AssignmentLister tool, the final output file with the crash channel assignments is available on an average of about 3-6 seconds depending on the size of the airbag project used to run the tool. With the old tool, it took a few minutes to generate the Excel file using macros. On an average, depending on the size of the airbag project, the speed of creating the output in a tabular form increased by a factor of 10 to 12. Hence, the Excel file in a tabular format is generated about 11 times faster in comparison to the old tool.

There are no negative effects of using the clipboard or any other MS Office Products while running the tool with the new solution. The tool functions correctly even with the use of MS Office Products.

Hence, considering all of the stated points, the output is available in a tabular format with the crash channel assignments in a stable, easier and faster way. Table 6.1 depicts the comparison of the old tool with the new tool based on various factors.

Features	**Old Tool** **(Run with Excel Macros)**	**New Tool** **(C# with Visual Studio IDE)**
Use of Excel Macros	Yes	No
Data Handling	Complex (using macros)	Simple (using XML format)
Speed to Create the Excel File	Slow, Seconds to a few minutes (for the example, > 30 seconds)	Faster, Just a few seconds (for the example, 3 seconds)
Stability	Unstable	Stable
Clipboard usage during Excel file generation	Not possible	Possible
Compatibility with MS Office Products	Incompatible	Compatible
Common Solution for Use Cases	No	Available
User Interface	Complex, Multiple manual steps (5)	Simple, Minimal manual steps (1)
Formatting and Styling Features	Limited options	More features

Table 6.1: Comparative Analysis of the Old and New AssignmentLister Tool

The new tool is evaluated against the old tool. Existing gaps present in the old tool is addressed by the new AssignmentLister tool. Various tests are conducted to ensure the working of the tool. The AssignmentLister tool is tested with various inputs, and the resulting outputs. Some compare tools are used to compare the results in the end. System tests ensure the results are as expected, and not corrupted. There is no specific hard requirement for the speed of the tool. The strategy without the use of macros increases the speed of the tool. The speed of creating the AssignmentLister.xlsx file is faster than the old tool. The user does not have to do multiple manual steps to generate the output file. The tool is stable to use. As a result, the new tool is easier to use than the old one.

The output generated meets the requirements, and expectations of the tool. The new features added enhance the visualization of crash channel assignments in a tabular form. The formatting and styling options can also be customized easily if needed by simple updates in the StyleSheet. Therefore, the overall performance of the AssignmentLister tool is better compared to the old solution for the stated reasons. The visualization of crash channel assignments in a tabular form is achieved with the developed AssignmentLister tool.

6.4 Summary

The results and evaluation chapter discusses the results of the AssignmentLister tool, along with its evaluation. This chapter starts with an overview of the results, along with the inputs required to produce the final output. Then, the way the intermediate XML and final Excel files are formatted as per user requirements can be seen. The details regarding the intermediate files, as well as the final Excel file are discussed with examples. The tests which are run to ensure the tool works properly are described. The evaluation of the AssignmentLister tool is discussed at the end of this section. For this, a comparative analysis of the old and new AssignmentLister tool is done. The new tool is evaluated against the old tool, and the findings are discussed. The following chapter outlines the conclusion of the thesis topic.

7 Conclusion and Future Scope

Automobile safety systems are measures meant to prevent accidents and reduce their consequences. Airbags are passive safety devices which protect passengers in the event of an accident. They lower the likelihood of serious harm or death in case of a collision. Airbag application tools are critical because they enable engineers to simulate, evaluate, and in turn improve the airbag system.

The airbag project consists of sensors. These sensors need data for simulation. For this reason, crash channels are assigned to sensors. These assignments are present for every crash in the project. They are known as crash channel assignments. The “AssignmentLister” is an airbag application tool which is used to visualize crash channel assignments in a tabular form. The old method for obtaining crash channel assignments had a few limitations. The tool was created using Excel Macros. The assignment information was poorly handled. The tool became slow and unstable because of the use of macros. The new AssignmentLister tool is developed using C# with Visual Studio. As a first step in the development of the tool, the crash channel assignment information is extracted. This information is stored to an XML file. XML is selected as the good-fitting file format for the necessary requirements of the tool. Styles are applied to this file using StyleSheet (XSLT). Then, the XML file produced is transformed to an Excel table. Depending on the specific requirements, the output format selected for the output file is in Excel format, due to its multiple benefits. This table displays crash channel assignments, with the styling options required.

As a future point, a graph could be generated from the final Excel result. The final output from the AssignmentLister tool, which is the Excel table, consists of a part with channels in the database. Amongst these channels, the channels which are assigned to sensors are called used channels. The other channels are the unused channels. This graph may contain the representation of the number of used channels in the database. This can provide an even better display of the information. Based on the requirements, this can be an option provided for the user to view the information. For this, some tools like Power BI can be used with Excel. Power BI can also be used with Excel to analyze the data present, in order to get a deeper insight through data visualization. Power BI visuals such as graphs and charts can directly be embedded into Excel to enhance data analysis of the assignment information present.

The AssignmentLister tool is developed considering future safety of the code. There will be new versions of Visual Studio, and other airbag application tools in the toolchain. As a result, it is important to update the code on a regular basis to maintain the performance of the AssignmentLister tool.

List of References

[1] Raith, Andreas, Kathrin Sattler, Rudolf Ertlmeier, and Thomas Brandmeier, "Networking and integration of active and passive safety systems." *in IEEE Proceedings of the ninth international workshop on intelligent solutions in embedded systems*, 2011, pp. 75-80.

[2] Schoeneburg, Rodolfo, and Thomas Breitling, "Enhancement of active and passive safety by future PRE-SAFE® systems," *in Proceedings of the 19th ESV Conference, Washington, DC, USA*, 2005.

[3] Ertlmeier, Rudolf, and Paul Spannaus, "Expanding design process of the Airbag Control Unit ACU-Connection of Active and Passive Safety by using vehicles dynamics for rollover and side crash detection," *in IEEE International Workshop on Intelligent Solutions in Embedded Systems*, 2008, pp. 1-9.

[4] Hussain, Aini, M. A. Hannan, Azah Mohamed, Hilmi Sanusi, and A. K. Ariffin, "Vehicle crash analysis for airbag deployment decision," *in International journal of automotive technology*, 2006, pp. 179-185.

[5] S. Shameem, G. R. K. Prasad, V. T. Ch, K. Bharath Kumar, M. Yaswanth and M. Ch, "Design Simulation and Analysis of Crash Sensor for Air Bag System," *in IEEE 3rd International Conference on Inventive Computation Technologies (ICICT)*, 2018, pp. 718-723.

[6] M. Lehmann. (2023). Student Page, Chassis Control - Occupant Safety Systems in the Bosch Docupedia Page (internal-bosch.com) [Online]. Available: https://inside-docupedia.bosch.com/confluence/aeos/core-assets/system-and-application/students-page

[7] Shirur, Naveen, Christian Birkner, Roman Henze, Thomas M. Deserno, and Darshankumar Dudhat, "Effect of airbag deployment phases on tactile occupant detection sensor," *in IEEE 2020 XII International Science-Technical Conference AUTOMOTIVE SAFETY*, 2020, pp. 1-5.

[8] D. Li, "Design and simulation of airbag flexible pressure sensor," in IEEE Conference on Telecommunications, Optics and Computer Science (TOCS), 2021, pp. 810-812.

[9] Adsul, Rahul, and S. Joshi, “Performance Modeling of Automotive Sensors and Sensor Interface Systems using Simulink,” *in International Journal of Science and Research (IJSR)*, 2013, pp. 233- 243.

[10] Hu, J., N. Orton, K. Boyle, N. Ashok, J. Klima, C. Staniak, R. Scherer, and M. Reed, "Seatbelt-Mounted Airbag Design For Occupant Protection In Tactical Vehicles During Frontal Crashes," *in Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium*, 2019, pp. 13-15.

[11] Trivedi, Mohan M., Shinko Yuanhsien Cheng, Edwin MC Childers, and Stephen J. Krotosky, "Occupant posture analysis with stereo and thermal infrared video: Algorithms and experimental evaluation," *in IEEE transactions on vehicular technology* 53, 2004, pp. 1698-1712.

[12] G. Gabriel. (2015, Jan. 27). Pedestrian Airbags, A First for Land Rover [Online]. Available: https://mobile.guideautoweb.com/en/articles/28126/pedestrian-airbags-a-first-for-land-rover/

[13] General Motors. (2012, Dec. 18). Cruze debuts industry-first flexible venting driver air bag [Online]. Available: https://phys.org/news/2012-12-cruze-debuts-industry-first-flexible-venting.html

[14] Hannan, Mahammad A., Aini Hussain, and Salina A. Samad, "System interface for an integrated intelligent safety system (ISS) for vehicle applications," *in Sensors 10, no. 2*, 2010, pp. 1141-1153.

[15] Amin, Md Syedul, Salwa Sheikh Nasir, Mamun Bin Ibne Reaz, Mohd Alauddin Mohd Ali, and Tae-Gyu Chang, "Preference and placement of vehicle crash sensors," *in Technical Gazette 21*, no. 4, 2014, pp. 889-896.

[16] J. Kocić, N. Jovičić and V. Drndarević, “Sensors and Sensor Fusion in Autonomous Vehicles,” *in 26th Telecommunications Forum (TELFOR)*, 2018, pp. 420-425.

[17] R. Lugner et al., “Evaluation of Sensor Tolerances and Inevitability for Pre-Crash Safety Systems in Real Case Scenarios,” *in IEEE 3rd Connected and Automated Vehicles Symposium (CAVS)*, 2020, pp. 1-6.

[18] N. Englisch, R. Bergelt and W. Hardt, “An Educational Platform for Automotive Software Development and Test,” *in IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*, 2020, pp. 1-4.

[19] N. Englisch, F. Hänchen, F. Ullmann, A. Masrur, and W. Hardt, “Application-Driven Evaluation of AUTOSAR Basic Software on Modern ECUs,” *in IEEE 13th International Conference on Embedded and Ubiquitous Computing*, 2015, pp. 60-67.

[20] Vodel, Matthias, Rene Bergelt, and Wolfram Hardt, “Grease framework-generic reconfigurable evaluation and aggregation of sensor data,” *in Proceedings of the 2nd International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies (ENERGY/InfoSys)*, 2012.

[21] Khan, Owes, Norbert Englisch and Wolfram Hardt, “Rapid Prototyping in AUTOSAR Based Systems,” *in GI-Jahrestagung*, 2017.

[22] Gaitzsch, Lucas, Norbert Englisch, and Wolfram Hardt, “Computer-Aided Population of Knowledge Base for Automotive Software Engineering,” *in IBS International Symposium on Computer Science, Computer Engineering and Educational Technology*, 2019.

[23] Englisch, Norbert, Owes Khan, Roland Mittag, Felix Hänchen, Ariane Heller, and Wolfram Hardt, “YellowCar,” *in INFORMATIK*, 2017.

[24] M. Fejes, Photo: N. Englisch. (2017, Mar. 14). Chemnitz University of Technology, University News, From Ecomobile to Driver Assistance Systems [Online]. Available: https://www.tu-chemnitz.de/tu/pressestelle/aktuell/7868/en

[25] Lippmann, Mirko, Batbayar Battseren, Ariane Heller, and Wolfram Hardt, “BlackPearl: Extended Automotive Multi-ECU Demonstrator Platform,” *in International Summer School on Computer Science, COMPUTER ENGINEERING AND EDUCATION TECHNOLOGY*, 2018.

[26] Kingsbury, Patrick, André Windisch, and Wolfram HARDT, “Modeling of Agile Avionics Software Development Processes through the Application of an Executable Process Framework,” *in International Conference on Design and Modeling in Science, Education, and Technology*, 2011.

[27] K. Cho, S. B. Choi, and H. Lee, “Design of an Airbag Deployment Algorithm Based on Precrash Information,” *in IEEE Transactions on Vehicular Technology*, May 2011, pp. 1438-1452.

[28] Böhmländer, Dennis, Vitor Yano, Thomas Brandmeier, Alessandro Zimmer, Lee Luan Ling, Chi-Biu Wong, and Tobias Dirndorfer, "A novel approach for intelligent pre-crash threat assessment systems," *in 17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2014, pp. 954-961.

[29] Chan, Ching-Yao, "Trends in crash detection and occupant restraint technology," *in Proceedings of the IEEE* 95, no. 2, 2007, pp. 388-396.

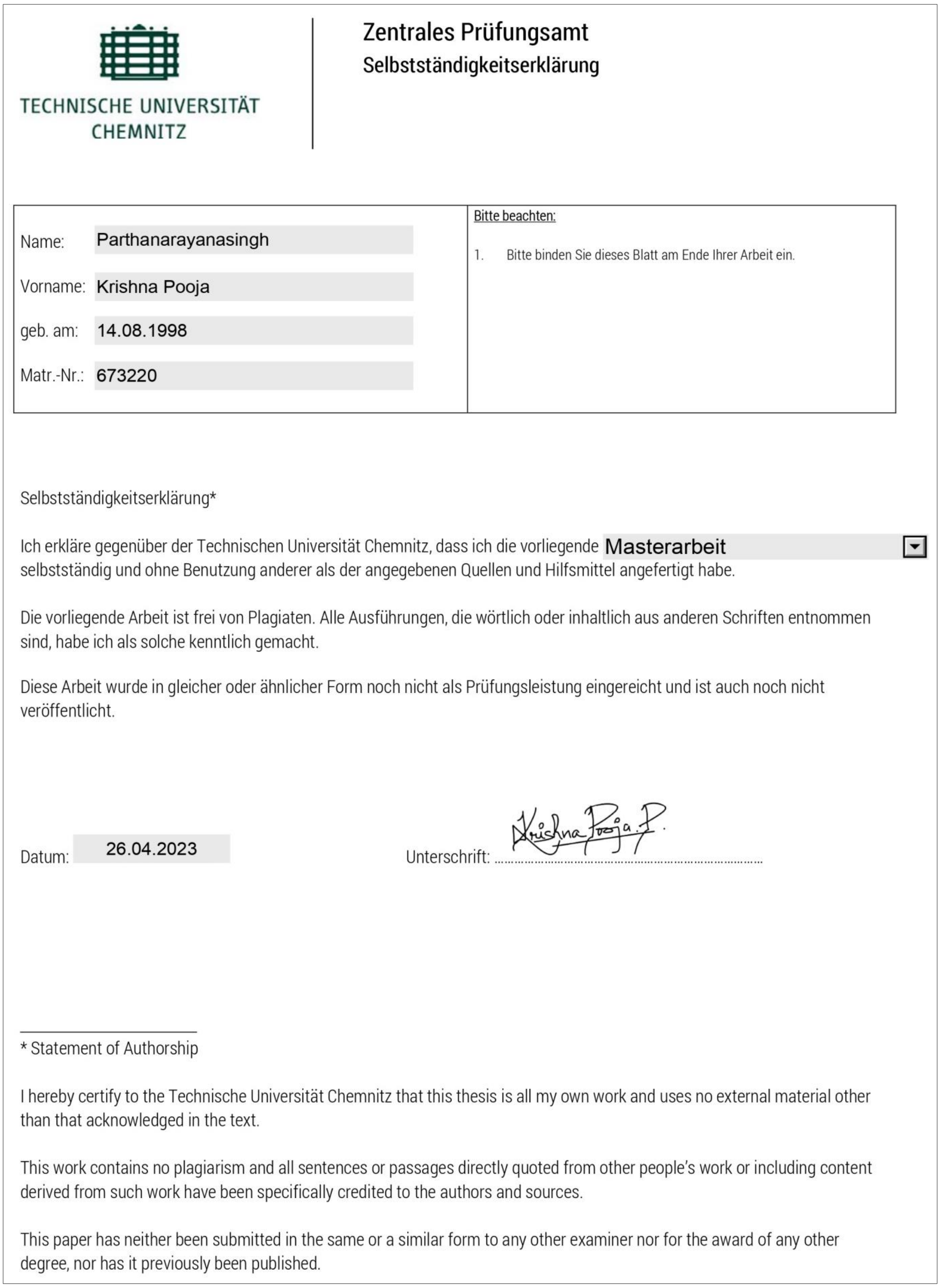
[30] K. Huber. (2006, Jun. 24). Fixed headers in large HTML tables [Online]. Available: https://www.codeproject.com/Articles/14563/Fixed-headers-in-large-HTML-tables

[31] M. Gudemann, M. Lipaczewski and F. Ortmeier, “Tool Supported Model-Based Safety Analysis and Optimization,” *in IEEE 17th Pacific Rim International Symposium on Dependable Computing*, 2011, pp. 294-295.

[32] A. Y. Bahar, S. M. Shorman, M. A. Khder, A. M. Quadir, and S. A. Almosawi, “Survey on Features and Comparisons of Programming Languages (PYTHON, JAVA, AND C#),” *in ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETSIS)*, 2022, pp. 154-163.

[33] K. Sneha and G. M. Malle, “Research on Software Testing Techniques and Software Automation Testing Tools,” *in International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, 2017, pp. 77-81.

Selbstständigkeitserklärung



Zentrales Prüfungsamt
Selbstständigkeitserklärung

TECHNISCHE UNIVERSITÄT CHEMNITZ

Name:	Parthanarayanasingh
Vorname:	Krishna Pooja
geb. am:	14.08.1998
Matr.-Nr.:	673220

Bitte beachten:

1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein.

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: 26.04.2023 Unterschrift: Krishna Pooja.P.

* Statement of Authorship

I hereby certify to the Technische Universität Chemnitz that this thesis is all my own work and uses no external material other than that acknowledged in the text.

This work contains no plagiarism and all sentences or passages directly quoted from other people's work or including content derived from such work have been specifically credited to the authors and sources.

This paper has neither been submitted in the same or a similar form to any other examiner nor for the award of any other degree, nor has it previously been published.