

8-2011

## Traffic safety: Modeling, analysis and visualization

Puneet Lakhanpal  
*University of Nevada, Las Vegas*

Follow this and additional works at: <https://digitalscholarship.unlv.edu/thesesdissertations>



Part of the [Electrical and Electronics Commons](#), [Transportation Commons](#), and the [Transport Phenomena Commons](#)

---

### Repository Citation

Lakhanpal, Puneet, "Traffic safety: Modeling, analysis and visualization" (2011). *UNLV Theses, Dissertations, Professional Papers, and Capstones*. 1235.  
<http://dx.doi.org/10.34917/2817755>

This Thesis is protected by copyright and/or related rights. It has been brought to you by Digital Scholarship@UNLV with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis has been accepted for inclusion in UNLV Theses, Dissertations, Professional Papers, and Capstones by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact [digitalscholarship@unlv.edu](mailto:digitalscholarship@unlv.edu).

# TRAFFIC SAFETY: MODELING, ANALYSIS AND VISUALIZATION

by

Puneet Lakhanpal

A thesis submitted in partial fulfillment  
of the requirements for the

**Master of Science in Electrical Engineering**

**Department of Electrical and Computer Engineering  
Howard R. Hughes College of Engineering  
Graduate College**

**University of Nevada, Las Vegas  
August 2011**



THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

**Puneet Lakhanpal**

entitled

**Traffic Safety: Modeling, Analysis and Visualization**

be accepted in partial fulfillment of the requirements for the degree of

**Master of Science in Electrical Engineering**

Department of Electrical and Computer Engineering

Pushkin Kachroo, Committee Chair

Masha Wilson, Committee Member

Rama Venkat, Committee Member

Monika Neda, Graduate College Representative

Ronald Smith, Ph. D., Vice President for Research and Graduate Studies  
and Dean of the Graduate College

**August 2011**

## ABSTRACT

### **Traffic Safety: Modeling, Analysis and Visualization**

by

Puneet Lakhanpal

Dr. Pushkin Kachroo, Examination Committee Chair  
Professor of Electrical and Computer Engineering  
University of Nevada, Las Vegas

Traffic Safety has always been one of the major issues of concern in United States. Every year, stringent efforts are made by the national agencies and safety offices to uplift the traffic safety standards and build systems which can guide them in policy making, reducing crashes and routing the financial resources in an optimal direction. This thesis studies the traffic safety from three different angles: modeling, analysis and visualization. In the beginning, these three components are explored in the domain of Injury Severity. Later on, the focus is shifted towards the Traffic Safety related to Safety Belts. Factors and models of the Injury Severity are explored and their visualizations are interpreted. The details of the Daytime Safety Belt Usage Surveys before and after the Click It or Ticket Mobilization (CIOT) in Nevada are provided. New iPhone and Windows Mobile based softwares are developed to facilitate the data collection process in Daytime Safety Belt Usage Surveys. Further, a comparative speed and accuracy analysis is performed between five data collection templates to corroborate the efficiency of the designed iPhone and Windows Mobile softwares. These efficient softwares form an important ingredient of a proposed feedback motivated framework for designing the Safety Belt Campaigns. The proposed feedback framework includes a Susceptible-Exposed-Infected-Carrier-

Recovered-Totally Immune (SEICRM) model for studying the diffusion of campaign information among the humans. The foundation of the SEICRM model comes from deterministic epidemiological models of infectious diseases. Additionally, the details of the iPhone application designed for Daytime Safety Belt Usage Surveys is provided, with fundamental concepts from the basics that can be quickly picked up even by novice developers. Finally, an online interactive data visualization software is developed using Adobe Flex, PHP, MySql and Google Maps API for Flash. A glimpse is also shown of the Adobe Flex programming, the backbone behind interactive data visualization software.

## ACKNOWLEDGMENTS

I would like to express my sincere thanks and respect to my advisor Prof. Pushkin Kachroo, who has been an elder brother to me and has always supported me in all my endeavours.

I am thankful to Masha Wilson for her faith in me as my Project Manager for approximately two years. Also, I am very grateful to Prof. Rama Venkat and Prof. Monika Neda for their helpful remarks.

I am thankful to my batchmates Sourabh Sriom, Shaurya Agarwal, Romesh Khaddar, Anuj Nayyar, Pratik Verma and Neveen Shlayan for the whole hearted support they have provided to me during my masters. I would like to give special thanks to Atul Sancheti for helping me out in my thesis by conducting the statistical analysis during the comparative study of softwares. Also, I am highly appreciative of Sergio Contreras for showing me how to develop iPhone applications. I am thankful to him for instigating the desire in me to go ahead and learn the iOS SDK development.

I am honored to have the love and support of my parents, Mr. Ashok Lakhanpal and Mrs. Veena Kumari, and my brother, Manuj. Without them, I would be nowhere in this world.

Last but not the least, I am greatly indebted to a few friends, Manita Choudhary, Tania Choudhary, Ramnik Kaur and Manish Bahri for their love and support during the entire period of my study.

This research has been done under the project funding from Department of Public Safety-Office of Traffic Safety, Nevada. Their support is gratefully acknowledged.

I dedicate this thesis to the tough journey of Real Analysis and the flavors of

iPhone Development and Adobe Flex.

## TABLE OF CONTENTS

ABSTRACT . . . . .	iii
ACKNOWLEDGMENTS . . . . .	v
TABLE OF CONTENTS . . . . .	vii
LIST OF FIGURES . . . . .	x
CHAPTER 1 INTRODUCTION . . . . .	1
Motivation and Research Goal . . . . .	1
Contributions . . . . .	2
Outline of the Thesis . . . . .	3
CHAPTER 2 INJURY SEVERITY AND CRASH ANALYSIS . . . . .	6
Injury categories and Associated factors . . . . .	6
Summary of findings from Research Literature . . . . .	9
Driver Related Injury Severity Studies . . . . .	10
Motorcyclists related injury severity studies . . . . .	12
Pedestrian related injury severity studies . . . . .	14
Crash related injury severity studies . . . . .	15
Models for injury severity analysis . . . . .	17
Ordered Probit Model (Duncan et al., 1998) . . . . .	17
Non-Parametric Classification Tree Techniques (Chang & Wang, 2006) . . . . .	19
Bayesian Ordered Probit Model (Y. Xie & Liang, 2009b) . . . . .	20
Unordered Probability Modeling (Savolainen & Mannering, 2007) . . . . .	21
Injury/Crash Analysis Systems . . . . .	23
WISQARS: Web-based Injury Statistics Query and Reporting System . . . . .	24
Trauma Data Analysis systems . . . . .	32
UMC and NDOT: Integrated Trauma and Crash Data . . . . .	34
CHAPTER 3 DAYTIME SAFETY BELT USAGE STUDIES . . . . .	41
Importance of Safety Belts . . . . .	41
Daytime Safety Belt Usage Surveys . . . . .	42
Daytime Safety Belt Usage Surveys in Nevada . . . . .	43
Site Selection . . . . .	43
Survey Design: Stage One . . . . .	44
Survey Design: Stage Two . . . . .	47
Basic data analysis . . . . .	61
Calculating the Weighted Data . . . . .	70
Statistical Analysis . . . . .	77
CHAPTER 4 DATA COLLECTION SOFTWARES FOR SAFETY BELT STUDIES . . . . .	89
Data Collection Templates . . . . .	90
Template 1: Paper & Pen . . . . .	90



Template 2: 2006 PDA Design (Vivoda & Eby, 2006) . . . . .	91
Template 3: 2007-09 PDA Design . . . . .	94
Template 4: 2010 PDA Design . . . . .	95
Template 5: iPhone . . . . .	101
Comparison Study of five templates . . . . .	102
Study Procedure . . . . .	102
Comparison 1: Speed . . . . .	104
Comparison 2: Accuracy . . . . .	116
CHAPTER 5 DESIGNING FEEDBACK MOTIVATED TRAFFIC SAFETY	
CAMPAIGNS . . . . .	137
History and success of CIOT Mobilization . . . . .	138
Media Advertising . . . . .	140
Feedback Framework for Campaign Design . . . . .	142
Conceptual framework for SEICRM model . . . . .	144
SEICRM categories . . . . .	145
Proposed SEICRM Model . . . . .	148
Explanation of the SEICRM model . . . . .	148
CHAPTER 6 iPHONE APPLICATION FOR DAYTIME SAFETY BELT US-	
AGE STUDIES . . . . .	158
Overview and Requirements . . . . .	158
Model-View-Controller Paradigm . . . . .	160
Differences between iOS and Mac OS X development . . . . .	160
Cocoa and Cocoa Touch Frameworks . . . . .	162
Building the Iphone Application . . . . .	164
Step1: Creating the Project SafetyIphone . . . . .	164
Step2: Adding resources to the SafetyIphone project . . . . .	166
Step 3: Adding Custom View Controllers . . . . .	173
Modifying the App delegate . . . . .	180
Modifying FirstPageController . . . . .	193
Modifying SecondPageController . . . . .	232
CHAPTER 7 GETTING FAMILIAR WITH ADOBE FLEX, PHP AND MYSQL 266	
Adobe Flex: What and Why ? . . . . .	266
Installation Details . . . . .	268
Installation of Adobe Flash Builder . . . . .	268
What is PHP and MySQL . . . . .	271
Installation of WAMP on Windows and LAMP on Linux . . . . .	272
Short tutorial on PhpMyAdmin and MySQL . . . . .	274
Fundamentals of Flex 3 . . . . .	277
A review of the components used . . . . .	277
Metadata tags . . . . .	280
Working with Events . . . . .	282
Differences between Flex 3 and Flex 4 . . . . .	287

CHAPTER 8	INTERACTIVE SAFETY BELT DATA VISUALIZATION SOFTWARE	294
Container for all components		294
Styling the application		296
Google Maps API for Flash		303
Features in Google Maps		303
Google Maps in Interactive Software		310
Custom Item Renderers		333
Creating Custom Item Renderers/Item Editors		333
Custom Item Renderers in Interactive Safety Belt Software		336
Creating Custom Events		340
Creating Actionscript Event class		341
Using [Event] metadata tag in MXML		343
CHAPTER 9	CONCLUSIONS, SUMMARY AND FUTURE WORK	347
Summary		347
Contributions		348
Future Work		349
VITA		361

## LIST OF FIGURES

2.1	Wisqars: Top 10 causes of Fatal injuries causing death in 2007 . . . .	26
2.2	Wisqars: Interactive Fatal Injury Mapping Module . . . . .	28
2.3	Wisqars: Non fatal injury database . . . . .	29
2.4	Wisqars: Violent Death database . . . . .	31
2.5	Wisqars: Cost of Injury Database . . . . .	33
2.6	American Trauma Center: Variation in areas covered with different response times . . . . .	35
2.7	American Trauma Center: Hospitals and Trauma Centers in Nevada .	36
2.8	Variables in Integrated NDOT Crash and Trauma Database . . . . .	37
2.9	R-PHP backend: Selecting analysis and variables . . . . .	39
2.10	R-PHP backend: Results in graphical and textual format . . . . .	40
3.1	Roadway Functional Classification - Clark County . . . . .	49
3.2	Roadway Functional Classification - Washoe County . . . . .	49
3.3	Selecting an Urban High Volume site in Clark county . . . . .	53
3.4	Sample observation locations for Clark and Washoe County . . . . .	54
3.5	Data collection software developed at TRC in 2010 . . . . .	57
3.6	Breakdown by Ethnicity of Drivers . . . . .	70
3.7	Seat Belt Use for Drivers and Passengers for Different Functional Classes of Streets . . . . .	71
3.8	GOF Test of Normality for <i>Diff</i> . . . . .	82
3.9	Histogram for <i>Diff</i> . . . . .	82
3.10	GOF Test of Normality for <i>PC</i> . . . . .	83
3.11	Histogram for <i>PC</i> . . . . .	83
3.12	Histogram for <i>Diff_bar</i> . . . . .	85
3.13	Histogram for <i>PC_bar</i> . . . . .	85
4.1	Template 1: Paper & Pen . . . . .	91
4.2	The design used in 2006 study (Vivoda & Eby, 2006) . . . . .	92
4.3	Template 2: 2006 PDA design . . . . .	93
4.4	Template 3: 2007-09 PDA Design . . . . .	94
4.5	Template 4: 2010 PDA Design . . . . .	95
4.6	Matrix design for two variables . . . . .	96
4.7	Hierarchical structure for theoretically $\infty$ variables (3 shown) . . . . .	97
4.8	Possible structures for 3 variables in data collection . . . . .	97
4.9	More possible structures for 3 variables in data collection . . . . .	98
4.10	Combining the variables . . . . .	99
4.11	Template 5: iPhone . . . . .	102
4.12	Traffic Counter . . . . .	105
4.13	Speed comparison: I-15 between Henderson and Arden . . . . .	106
4.14	Speed comparison: US-95 between SR 157 and Indian Springs . . . . .	108
4.15	Speed comparison: SR-161 at I-15 . . . . .	109
4.16	Speed comparison: I-15 at Sahara . . . . .	111
4.17	Speed comparison: Maryland at Charleston . . . . .	112

4.18	Speed comparison: Torrey Pines at Spring Mountain . . . . .	114
4.19	Speed comparison: Overall Results . . . . .	115
5.1	Where Nevadans read, saw or heard about Seat Belt Law Enforcement (n = 341) . . . . .	141
5.2	Coverage Areas of Televisions . . . . .	143
5.3	The feedback framework . . . . .	144
5.4	Proposed Model for Diffusion of Campaign Information . . . . .	153
6.1	Screenshot of the iPhone application . . . . .	159
6.2	Welcome screen in Xcode . . . . .	164
6.3	Selecting Window-based application template . . . . .	165
6.4	Files automatically created in Xcode . . . . .	166
6.5	Classes in UIKit framework (Adapted from iOS developer library, 2010d)	168
6.6	Classes in Foundation framework (Adapted from iOS developer library, 2010a) . . . . .	169
6.7	Adding file to project . . . . .	171
6.8	Property list . . . . .	172
6.9	UIViewController class as a part of UIKit framework(Adapted from iOS Developer Library, 2011c) . . . . .	174
6.10	Structure of FirstViewController . . . . .	175
6.11	Adding <i>FirstPageController</i> to <i>SafetyIphone</i> project . . . . .	178
6.12	Creating <i>FirstViewController</i> source and header files . . . . .	179
6.13	Window object in <i>MainWindow.xib</i> . . . . .	183
6.14	<i>FirstPageController.xib</i> opened in Interface Builder . . . . .	194
6.15	Changing the owner of <i>FirstPageController.xib</i> . . . . .	195
6.16	<i>view</i> outlet connected to <i>View</i> object in <i>FirstPageController.xib</i> . . .	195
6.17	Attaching a Navigation Bar to the <i>View</i> object in <i>FirstPageController.xib</i>	196
6.18	Adding a Label to the <i>View</i> object in <i>FirstPageController.xib</i> . . . .	197
6.19	Adding a button to <i>FirstPageController.xib</i> . . . . .	197
6.20	Final <i>View</i> of <i>FirstPageController.xib</i> . . . . .	198
6.21	Changing the owner of <i>SecondPageController.xib</i> . . . . .	233
6.22	<i>view</i> outlet connected to <i>View</i> object in <i>SecondPageController.xib</i> . .	233
6.23	Adding a <i>View</i> from the <i>Library</i> . . . . .	234
6.24	Final <i>View</i> of <i>SecondPageController.xib</i> . . . . .	235
6.25	Hierarchial structure of objects in View Mode . . . . .	236
7.1	Screenshot of the final application . . . . .	267
7.2	Installing Adobe Flash Builder 4: Entering the serial number . . . . .	270
7.3	Installing Adobe Flash Builder 4: Install Options Screen . . . . .	271
7.4	Installing Adobe Flash Builder 4: Eclipse plug-in version . . . . .	272
7.5	WAMP server configuration HomePage . . . . .	274
7.6	PhpMyAdmin Homepage . . . . .	275
7.7	Creating a table in MySQL . . . . .	276
7.8	Inserting values in the table . . . . .	276

7.9	Final table in MYSQL . . . . .	277
7.10	Custom SQL Query, introducing the % wildcard . . . . .	278
7.11	States demo in Flex 3 and Flex 4 . . . . .	293
8.1	Interactive Safety Belt Data Visualization Software (Lakhanpal, 2010)	295
8.2	Google Map API for Flash: Styled Maps . . . . .	304
8.3	Google Maps API for Flash: Driving Directions Service . . . . .	305
8.4	Google Maps API for Flash: Database for generating Encoded Polylines	310
8.5	Google Maps API for Flash: Generated Traffic Animation with En- coded Polylines . . . . .	311
8.6	Interactive Software: Displaying sites on Google MAPS . . . . .	311
8.7	Interactive Software: Panel obtained by a single click on the marker .	320
8.8	Interactive Software: Columnchart x axis showing year, age, gender and ethnicity distribution . . . . .	321
8.9	Custom Item Renderers/Item Editors . . . . .	336
8.10	TileList Custom Item Renderers . . . . .	337

# CHAPTER 1

## INTRODUCTION

### 1.1 Motivation and Research Goal

Traffic Safety is a primary research area in United States (U.S.). In U.S., history documents several efforts that have been made to ensure that the gift of life is preserved for all its residents. Several national level agencies like National Highway Traffic Safety Administration, Office of Traffic Safety, Department of Transportation, National Transportation Safety Board etc. strive hard throughout the year, engendering public awareness that people should be belted during the day or night, should not drive while they are drunk and should follow the pedestrian crossing rules and regulations. Still, fatalities happen and people are injured to different levels, certainly in a few cases when the safety belts are not used. Several statistical figures exist which point towards the need for a responsible behavior from every driver and compliance with the state's rules and regulations. It is important that people understand that negligence towards traffic safety not only costs them or their family members, but also heavily impacts the surrounding innocent victims who have no idea whatsoever of the imminent deadly moments.

A definitive framework is required to steer the safety efforts in the right direction. Till now, the effectiveness of these efforts have been explored through different criteria, for example: how many lives were saved, how many citations were given for not wearing the safety belts. Although these measures has been successful for deciding the route of the efforts, they are still not sufficient to reach a stage where the behavior of people can be expected and more importantly, controlled. Thus, a conceptual

framework is required which models how people respond to traffic safety efforts. An accurate model of such behavior can be used to guide the efforts so that the behavior can be controlled in a feedback manner.

This research proposes such a framework to model the behavior towards safety belt usage. Furthermore, it adds a new dimension to the traffic safety studies, primarily Injury Severity and Daytime Safety Belt Studies. Besides modeling and analysis in both these domains, stress is given up the necessity of improving the current data collection standards and then visualizing it to perform an in-depth online analysis accessible to the general public.

## 1.2 Contributions

The specific contributions of this thesis work are listed below.

1. An Optimal Click theory on the constrained and unconstrained design surfaces has been proposed and a data collection template has been designed on Personal Digital Assistant (PDA) running Windows Mobile 6 platform and iPhone 4. The development of the PDA and iPhone 4 softwares revolutionizes the current trends in data collection. This work is presented in Chapter 4.
2. A comparative study, documenting the differences in speed and accuracy of five data collection templates including paper and newly developed softwares has been provided. Previous data collection designs were re-programmed to compare them with the new ones. Further, statistical analysis is performed to strengthen the claims that there is no significant difference in the accuracy of

the five templates. This work is presented in Chapter 4.

3. A feedback framework has been proposed to measure the behavior of the people and control the safety belt campaigns year after year. A model, based on Epidemiology, is proposed to study the flow of campaign information through people and observe how different people respond to different channels of the safety belt campaigns, specifically 'Click It or Ticket'. This work is presented in Chapter 5.
4. An Interactive Data Visualization Software has been designed which performs the data analysis over the past five years not just on a statewide level but also delves deep into the analysis over every observation site. This software, developed in Adobe Flex 3 with Php and MySQL as helpers at the backend, reaches new heights in the area of Daytime Safety Belt Analysis and has been fully discussed in this thesis. Code listings along with figures have been provided to illustrate the concepts in a better manner. This work has been presented in Chapter 8.

### 1.3 Outline of the Thesis

This thesis is divided into the following chapters.

1. Chapter 1 presents the motivation and contributions of the thesis.
2. Chapter 2 analyzes the factors responsible for increasing and decreasing injury severity. A literature survey of the papers in this domain is provided and the synopsis of a few models extracted from them are presented. At the end of



the chapter, three online crash/injury severity analysis systems are provided for facilitating injury related queries and exploring the inter-relationships between a plethora of variables in the crash data.

3. Chapter 3 presents a detailed analysis of the Daytime Safety Belt Usage Surveys in Nevada. In the beginning, a survey design is presented which provides the specifics of selecting the observation sites for data collection. Later, the collected data is analyzed and a few basic results are presented.
4. Chapter 4 presents two newly developed data collection softwares on Windows Mobile and iPhone. A comparative study for examining the performance of five data collection templates on the grounds of speed and accuracy is provided. Data is collected through all these templates and percentage observed data is considered to be the basis of comparing speeds of different templates.  $\chi^2$ , T-test and F-test are done on the collected safety belt usage data to test the hypothesis that there is no significant difference between the data obtained from different templates.
5. Chapter 5 presents a feedback motivated framework for designing safety belt campaigns. Initially, the history and the success stories of 'Click It or Ticket' Mobilization are provided. Simultaneously, huge resources spent on elevating the law enforcement and buying advertising media are observed, motivating us to develop a feedback framework which involves studying the behavior of people towards campaigns through the sensors designed in Chapter 4 and using the measured behavior to steer the campaigns towards a target behavior. A

model derived from Epidemiology is then proposed to study how the campaign information diffuses through the population.

6. Chapter 6 presents the fundamental framework required to build an iPhone application, The steps in building the iPhone application for Daytime Safety Belt Usage Surveys are discussed with an initial focus on providing a line by line description of the included code.
7. Chapter 7 presents the essential elements of installing and programming Adobe Flex, PHP and MySQL languages. This chapter is a pre-requisite to comprehend the detailed concepts involved in creating an Interactive Data Visualization Software developed in the next chapter.
8. Chapter 8 presents an Interactive Data Visualization Software developed for the Daytime Safety Belt Usage Surveys in Nevada. The data collected during these surveys is hosted on a MySQL server and data queries are performed through PHP to show nice animated results through on-the-fly charts and data-grid components. A few code snippets to perform advanced tasks like working with Google Map API for Flash, customizing the item renderers etc. in Adobe Flex are presented.

## CHAPTER 2

### INJURY SEVERITY AND CRASH ANALYSIS

#### Abstract

Injury severity analysis is of great importance for the researchers and the practitioners who look into details of accidents, external costs and transportation planning. Real time data obtained from the injuries can be used to build statistical models which analyze the effect of likely factors which might have contributed towards different levels of injuries. Thus, the knowledge of crashes and injuries can be used to prevent further accidents. Such an analysis is very important for traffic safety improvements. Therefore, this chapter explores several factors and models from the injury severity literature and highlights the online systems which allow the researchers to visually interpret the highly complex crash and injury severity data.

#### 2.1 Injury categories and Associated factors

Injury severity has been divided into a few categories based upon the damage involved. Various researchers have divided the injury levels in different categories. However, the essence remains the same. A few of them noted down from research literature are given below.

##### **Injury categories:**

1. Property damage only, incapacitating and fatal
2. Property damage only, Possible injury, Injury (evident, disabling or fatality)
3. Slight injury, Serious injury, Fatal injury

4. No injury, Possible injury, Non incapacitated injury, Capacitated injury, Fatal injury
5. No injury, Slight injury, Severe injury, Fatal injury
6. Survival Risk Ratio (the number of patients with a certain injury code who have not died in the hospital and the total number of patients diagnosed with that code)
7. Injury Severity Score (0-75)
8. Low individual severity, High individual severity
9. Non fatal, Fatal
10. Non-treated injury, Treated injury, Admitted injury, Death

A lot of research has been performed to identify the factors which might significantly affect the injury severity. A few of them have been mentioned below.

**Associated factors:**

1. Driver Characteristics:

Driver Age, Gender, Alcohol Consumption, Safety equipment usage, Driver experience

2. Vehicle characteristics:

Vehicle type and the number of vehicles involved in the crash

3. Traffic Characteristics:

Speed of the vehicle, Speed limit, Volume

4. Road factors:

Road curvature, number of lanes, type of road (e.g. rural, urban), surface conditions, junctions.

5. Crash characteristics:

Type of crash, Point of impact(left/right/top/bottom)

6. Other characteristics:

Day of the week, Time of the day, Weather conditions, Lighting conditions, Travel Purpose

In the research literature connected to injury severity, many authors agree upon a few factors which increase or decrease injury severity. However, a few studies exist in which author argue about the influence of a certain factor on injury severity. The factors over which concensus exists and over which argument can be found are shown below:

**Factors associated with common consensus:**

1. Factors increasing injury severity:

- Driver age
- Alcohol consumption
- Head on collision
- Driving heavy vehicles
- Presence of poor lighting conditions

- Existence of vertical and horizontal curvature
- Rural vs. urban areas
- Speeding

2. Factors decreasing injury severity:

- Wearing a safety belt
- Having airbags

**Factors associated with conflict:**

1. Gender(especially, Female drivers)
2. Intersections type(T and Y interchanges and junctions)
3. Road surface conditions (presence of icy/wet/normal pavements and surfaces)
4. Seating position(drivers seat position/left rear position)
5. Weather conditions(Fine weather/rain/snow)

## 2.2 Summary of findings from Research Literature

Extensive research has been performed in which various methods of analysis have been exploited to provide factors which might increase injury severity. Since injury severity covers a vast ground, this research has been divided into branches such as driver related studies, motorcyclists related studies and crash related studies. This section provides a brief overview of the injury severity studies performed in all the aforementioned branches and highlights the key findings in an enumerated manner.

Firstly, the conclusions from each of the papers studied have been mentioned and later, a few algorithms used in these papers have been discussed.

### 2.2.1 Driver Related Injury Severity Studies

#### **Chimba & Sando (2009):**

1. Method of analysis:

ANN backpropagation, Ordered probit model

2. Factors considered:

Driver characteristics, Speed limit, General info, Weather, Road geometry.

3. Factors increasing injury severity:

No daylight, Cloudy sky, Curved sections, Higher speed limit, Alcohol use,  
Turning movement

#### **Y. Xie & Liang (2009a):**

1. Method of analysis:

Ordered probit model, Bayesian probit model

2. Factors considered:

Driver characteristics, Vehicle characteristics, General info, Road geometry,  
Crash characteristics, Weather

3. Factors increasing injury severity:

Driver age, Vehicle age, Drunk driving, curvy road alignments, inadequate lighting, initial impact points on left area, rollover and fire.

**Boufous et al. (2008):**

1. Method of analysis:

Linear regression

2. Factors considered:

Driver characteristics, vehicle characteristics, crash characteristics, general info, road geometry, area type.

3. Factors increasing injury severity:

Complex intersections(Y/T/roundabouts), High speed limit, no safety belt, drivers errors, speeding, rural areas.

**Conroy et al. (2008):**

1. Method of analysis:

Logistic regression

2. Factors considered:

Driver characteristics, crash characteristics, vehicle damage.

3. Factors increasing injury severity:

Drivers with intrusion into their position, driving a passenger vehicle.

**Abdel-Aty (2003):**

1. Method of analysis:

Ordered probit model, Multinomial logit model, nested logit model.



2. Factors considered:

Driver characteristics, crash characteristics, General info, Road geometry, Area type, Toll characteristics, Weather.

3. Factors increasing injury severity:

Being over 65, Being female, Impact on side, No safety belt use, e-pass use, number of impacts, adverse weather, not driving a truck.

### 2.2.2 Motorcyclists related injury severity studies

**Pai (2009):**

1. Method of analysis:

Binary logistic model

2. Factors considered:

Driver characteristics, motorcyclist characteristics, crash characteristics, General info, Weather

3. Factors increasing injury severity:

Riding over 60, Engine size over 125cc, heavy goods vehicle involvement,  $\geq 2$  vehicles involved, fine weather, non built roads, right-of-way violation.

**Pai & Saleh (2008):**

1. Method of analysis:

Ordered probit model

2. Factors considered:

Motorcycle characteristics ,Motorcyclist characteristics, crash characteristics,  
General info, Weather, Road Geometry

3. Factors increasing injury severity:

Rider being male or over 60, Increasing Engine size , Crash partner other than  
motorcycle, Darkness, Fine weather, Spring/summer, Midnight/early morning,  
Weekend riding, Speeding.

**Chang & Wang (2006):**

1. Method of analysis:

Nested logit model, Standard multinomial logit model.

2. Factors considered:

Driver and rider characteristics, vehicle characteristics , crash characteristics,  
General info, Speed, Road Geometry.

3. Factors increasing injury severity:

Increasing age, speeding, April and July, darkness, collisions with roadside ob-  
ject, being female, alchohol involvement.

**O'Donnell & Connor (1996):**

1. Method of analysis:

Ordered probit model, Ordered logit model

2. Factors considered:

Vehicle characteristics, crash characteristics, occupants characteristics.

3. Factors increasing injury severity:

Seating on left-rear position, Female, Headon collision.

2.2.3 Pedestrian related injury severity studies

**Kim et al. (2008):**

1. Method of analysis:

Heteroskedastic logit model

2. Factors considered:

Driver characteristics, crash characteristics, General info, Road geometry, Pedestrian characteristics, Area type, Weather.

3. Factors increasing injury severity:

Intoxicated driver, darkness with or without streetlights, greater pedestrian age, sport-utility vehicle, truck, freeway, state-route, speeding.

**Sze & S.C (2007):**

1. Method of analysis:

Logistic regression

2. Factors considered:

General info, Road geometry, non-motorist characteristics, speed limit, congestion.

3. Factors increasing injury severity:

Age above 65, Head injury, a crash at a crossing or on a road section with a speed limit above 50 km/h or at a signalized intersection

**Lee & Abdel-Aty (2005):**

1. Method of analysis:

Ordered probit model

2. Factors considered:

Driver characteristics, vehicle characteristics, general info, road geometry, pedestrian characteristics.

3. Factors increasing injury severity:

Pedestrian being old, female or intoxicated, high vehicle speed, adverse weather and dark lighting, vans, buses and trucks versus passenger cars, rural versus urban areas, intersections without traffic control

**Ballesteros et al. (2004):**

1. Method of analysis:

Logistic regression

2. Factors considered:

Vehicle characteristics, speed limit

3. Factors increasing injury severity:

SUVs and PUs compared to conventional cars.

2.2.4 Crash related injury severity studies

**Quddus et al. (2009):**

1. Method of analysis:

Ordered Logit, Heterogenous choice model(HCM), Generalized ordered logit,  
Partial proportional odds.

2. Factors considered:

General info, Traffic characteristics, Congestion, Weather, Road Geometry.

3. Factors increasing injury severity:

Fine weather versus rain, weekdays and darkness.

4. Factors not influencing injury severity:

Congestion, traffic flow, snow.

**Eluru et al. (2008):**

1. Method of analysis:

Ordered response logit model, Mixed Generalized ordered-response logit.

2. Factors considered:

Driver characteristics, Vehicle characteristics, Non-motorist characteristics, Weather,  
Road Geometry.

3. Factors increasing injury severity:

Being pedestrian versus being cyclist, Higher speed limits, Being male and older,  
Crashes with vehicles other than passenger cars, Frontal impact crash, Evening  
and late night periods

4. Factors decreasing injury severity:

Snow, Signalized intersections.

**Gray et al. (2008):**

1. Method of analysis:

Ordered Probit model.

2. Factors considered:

Young male driver characteristics, Crash characteristics, General info, Road geometry, Weather.

3. Factors increasing injury severity:

Driving in darkness, Trips in early morning and towards the end of week, Driving on main roads, Overtaking maneuvers, Weather other than fine no high winds, Speed limit of 60 mph, Passing the site of a previous accident versus other carriageaway hazards.

## 2.3 Models for injury severity analysis

### 2.3.1 Ordered Probit Model (Duncan et al., 1998)

Ordered probit models are used to model marginal probability effects of contributory variables on severity and take into account indexed outputs, explanatory variables (inputs) and an error error term with normal distribution. It is assumed that there exists some continuous latent variable. Ordered probit models are used in injury severity because, it identifies statistically significant relationships between explanatory variables and a dependent variable( output i.e. injury severity). Given a unit change in explanatory variable, it does not assume that a minor injury and no injury is the same. Hence, ordered probit model captures qualitative differences

between different injury severities.

The ordered probit model uses the following form:

$$y^* = \beta x + \epsilon \quad (2.1)$$

where,  $y$  is the dependent variable (injury severity) coded as  $0, 1, 2 \dots i - 1$  (where  $i$  = Number of dependent variables);  $\beta$  is the vector of estimated parameters and  $x$  is the vector of explanatory variables;  $\epsilon$  is error term which is assumed to be normally distributed (zero mean and unit variance) with cumulative distribution function  $\varphi$  and density function denoted by  $\phi$ . Given a crash, an individual falls in category  $i$  if  $\mu_{i-1} < y^* < \mu_i$ . The injury data,  $y$ , are related to the underlying latent variable  $y^*$ , through thresholds  $\mu_i$ . The probabilities can be written as:

$$P(y = i) = \varphi(\mu_i - \beta x) - \varphi(\mu_{i-1} - \beta x) \quad (2.2)$$

where,  $i = 1, 2, 3 \dots$  number of dependent variables,  $\mu_0 = +\infty$  and  $\mu_1 < \mu_1 < \mu_3$  are defined as three thresholds between which categorical responses are estimated. The likelihood function is used for parameter  $\mu$  and  $\beta$  estimation as given in McKelvey & Zavoina (Summer 1975). The thresholds  $\mu$  show the range of normal distribution associated with the specific values of the response variable. The parameters  $\beta$  represent effect of changes in explanatory variables.

### 2.3.2 Non-Parametric Classification Tree Techniques (Chang & Wang, 2006)

Classification and Regression Tree(CART) analysis is performed in engineering for prediction problems. A classification tree is developed if the value of the target variable is discrete and a regression tree is developed when the value of target variable is continuous. Injury severity is a discrete variable, hence a classification tree can be developed as a statistical model.

#### **Steps in development of a CART model:**

##### 1. Tree Growing:

The principle behind tree growing is to recursively partition the target variable to maximize purity in the two child nodes. For example: In injury severity(binary variable:injury or no injury), the program would check all the input variables(splitters) to find a threshold which would divide the individuals into two separate groups who were injured and who were not injured. This would be based on a binary decision which first finds a threshold for, lets say age, and divides the dataset into two groups based on that threshold. Similarly, the thresholds for other factors are found to achieve the best purity possible in the terminal nodes. For N categories, the algorithm recursively partitions the target variable to achieve a saturation tree. To develop a cart model, a dataset is divided into training data set and testing dataset and the saturated tree is constructed from the training data.

##### 2. Pruning:

This process aims to reduce the complex saturated trees into a sequence of sim-



pler trees by pruning the terminal nodes which increase the complexity of the process. All those branches are removed which add little to the predictive values of the tree. As mentioned before, it depends upon a compexity parameter which is a cost function of misclassification of data. At the end of pruning, relationship between misclassification cost and tree complexity in terms of number of terminal nodes is obtained.

### 3. Selecting optimal tree:

In this process, an optimal tree is chosen from the pruned trees. The misclassification cost decreases monotonically for the training data with the growth of trees, indicating a best fit obtained for the saturated tree. On the other hand, the misclassification cost for the testing data decreases first and then increases after reaching a minimum, indicating that the saturated tree is greatly overfit when applied to analyze the testing data. Thus, the optimal tree is chosen when the misclassification costs reach minimum for both learning and testing data.

A more detailed analysis of classification and regression trees is provided in L. Breiman & Stone (1998).

#### 2.3.3 Bayesian Ordered Probit Model (Y. Xie & Liang, 2009b)

In Ordered probit model, parameter estimation is totally based on the data quality. However, in Bayesian Ordered Probit model, MLE algorithm is replaced by Bayesian Inference in which prior distributions are used to accomodate for less data.

#### 1. Bayesian Inference:

If  $\theta$  represents the parameters to be estimated, the posterior distribution of  $\theta$

is calculated as:

$$\pi(\theta|y) = \frac{f(y|\theta)}{m(y)} \propto f(y|\theta)\pi(\theta) \quad (2.3)$$

where  $y = y_1, y_2, \dots, y_n$  = observed outcomes;  $\pi(\theta)$  = prior distribution of  $\theta$ ;  $f(y|\theta)$  = sampling distribution;  $m(y) = \int_{\theta} f(y|\theta)\pi(\theta)d\theta$  = marginal distribution of  $y$ ; and  $\pi(\theta|y)$  = posterior distribution of  $\theta$ .

## 2. Modeling:

The latent variables in Ordered Probit model are augmented to the vector of unknown parameters  $\beta$  and  $\theta$  such that their joint posterior distribution is calculated as:

$$\pi(\beta, \gamma^*, z|y) \propto \pi(\beta, \gamma^*) \prod_{i=1}^n \left\{ \phi(z_i - x_i^T \beta) \times \prod_{k=1}^C [I(\gamma_{k-1} < z_i < \gamma_k)]^{I(y_i=k)} \right\} \quad (2.4)$$

where  $\pi(\beta, \gamma)$  = prior distribution of  $\beta$ ,  $\gamma$ ;  $\phi(\cdot)$  = probability density function of the standard normal distribution; and  $z = z_1, z_2, \dots, z_n$ ; and  $z_i \sim N(x_i^T \beta, 1)$ .

Markov chain Monte Carlo algorithm is used to simulate approximately independent samples from joint posterior distribution.

### 2.3.4 Unordered Probability Modeling (Savolainen & Mannering, 2007)

The statistical modelling of crash injury severity data can be performed by ordered probability models(ordered logit and probit) and unordered probability models(multinomial and nested logit). Ordered probability models are sometimes not effective for injury severity analysis since they might not be able to cope with the underreporting which happens in case of lower injury levels, thus facing a problem

of bias and inconsistent estimation of model coefficients. In contrast, the unordered multinomial logit probability model consistently estimates the coefficients as given in S.P. Washington (2003).

First, a linear function is defined that determines motorcyclist  $n$ 's injury-severity outcome  $i$  as:

$$S_{in} = \beta_i X_{in} + \epsilon_{in} \quad (2.5)$$

where,  $X_{in}$  is a vector of measurable characteristics that determine the injury severity for motorcyclist  $n$ ,  $\beta_i$  is a vector of estimable coefficients and  $\epsilon_{in}$  is an error term accounting for unobserved effects influencing the injury severity of crash  $n$ . According to Manski & McFadden (1981), if  $\epsilon_{in}$  are assumed to be generalized extreme value distributed the standard multinomial logit model results,

$$P_n(i) = \frac{\exp[\beta_i X_{in}]}{\sum_{\forall I} \exp(\beta_I X_{In})} \quad (2.6)$$

where,  $P_n(i)$  is the probability that crash  $n$  will result in motorcyclist injury outcome  $i$  and  $I$  is the set of possible crash-injury-severity outcomes. The above equation is estimable using maximum likelihood techniques. In multinomial logit model, it is assumed that the error term  $\epsilon_{in}$  is independently distributed across the outcomes which is bad if outcomes share unobserved effects. However, the nested logit model generated using the generalized extreme value distribution of  $\epsilon_{in}$  overcomes this restriction, thus grouping alternatives that share unobserved effects into conditional nests. The nested logit model has calculated the probability for injury outcome  $i$  resulted by

motorcyclist  $n$ 's crash as:

$$P_n(i) = \frac{\exp[\beta_i X_{in} + \phi_i LS_{in}]}{\sum_{\forall I} \exp(\beta_I X_{In} + \phi_I LS_{In})} \quad (2.7)$$

$$P_n(j|i) = \frac{\exp[\beta_{j|i} X_n]}{\sum_{\forall J} \exp(\beta_{J|i} X_{Jn})} \quad (2.8)$$

$$LS_{in} = LN[\sum_{\forall J} \exp(\beta_{J|i} X_{Jn})] \quad (2.9)$$

where,  $P_n(i)$  is the unconditional probability of motorcyclist  $n$  having injury outcome  $i$ ,  $X$  is vectors of measurable characteristics defined above,  $\beta$  is a vector of estimable coefficients and  $P_n(j|i)$  is the probability of motorcyclist  $n$ 's crash having injury severity  $j$  conditioned on the injury severity being in injury-severity category  $i$ ,  $J$  is the conditional set of outcomes (conditioned on  $i$ ),  $I$  is the unconditional set of outcome categories,  $LS_{in}$  is the inclusive value (logsum), and  $\phi_i$  is an estimable parameter. The details about estimation of the nested model can be found in Savolainen & Mannering (2007).

## 2.4 Injury/Crash Analysis Systems

Several researchers throughout US have been striving hard to provide different tools that can be used nationwide to explore the injury and crash statistics over the past few years. These tools not only enable the media, health professionals and the general public to learn about public health and economic burden associated with injuries throughout US but also provide the capabilities to find the correlation between the factors responsible for different levels of injury. This section provides a general

overview of these systems that are currently available or are under development.

#### 2.4.1 WISQARS: Web-based Injury Statistics Query and Reporting System

The Centers for Disease Control and Prevention (Atlanta,GA) maintain a web based interactive database which can be queried to provide four types of data: fatal and non fatal injury, violent death and cost of injury from a wide pool of trusted sources. The users can query this database and sort the results to produce reports, charts and maps based on the following options available:

1. Intent of injury:

For example: unintentional, violence related, homicide/assault, legal intervention, suicide etc.

2. Cause of injury:

For example: Fall, motor vehicle crash, poisoning etc.

3. Body region:

For example: Traumatic brain injury, spinal cord, torso etc.

4. Type of injury:

For example: Fracture, dislocation, burn, internal injury etc.

5. Geographic location of injury:

For example: National, regional, state

6. Person Characteristics:

For example: Gender, Ethnicity, Age

The WISQARS interactive database can be found online (Centers for Disease Control and Prevention, 2011).

#### 2.4.1.1 WISQARS fatal injury reports

WISQARS fatal injury reports include the following assets:

1. Reports showing number of injury deaths and death rates categorized by intent and cause of injury, geographical stratification, ethnicity, gender and age.
2. Reports showing the leading cause of deaths highlighting the impact of injury related deaths compared to other causes.
3. Reports showing the Years of potential life lost by demonstrating the effect of premature injury deaths compared to other causes.

Figure 2.1 shows the query that can be made to the fatal injury database for finding out the topmost 10 leading causes of death for 2007 (the most recent data). It also shows the result of the query, hence describing the leading causes in 2007.

Furthermore, the National Center for Injury Prevention and Control (NCIPC) maintains a WISQARSTM (Web-based Injury Statistics Query and Reporting System) Fatal Injury Mapping Module which allows users to produce color coded fatal injury maps, thereby showing the patterns of injury death rates over customized geographical levels. This information can be vital for identifying the populations at high risk of injury by its cause and intent. This module utilizes a database spanning seven years of data, which is sufficient for providing county-level reliable estimates of the injury-related death rates. The public health officials can use this module to

Figure 2.1: Wisqars: Top 10 causes of Fatal injuries causing death in 2007

# WISQARS Leading Causes of Death Reports, 1999 - 2007

Choose your **Report Options**, then click the **Submit Request** button.

For more information about an option or a category of options, click on the underlined name or phrase. To return to this page, click on the "back" button in your browser toolbar.

**Report Options**

Census Region/State  
United States

Race  
All Races

Sex  
Both Sexes

Year(s) of Report  
2007 to 2007

Hispanic Origin  
All

Output Options  
Standard Output

Submit Request Reset

**Advanced Options** (Not Required)

Number of Causes  
Top 10

Categories of Causes  
All Deaths-with drill-down to ICD codes

Age Group Formatting

☒ 1-14 in 5-year groups; 15-65+ in 10-year groups  
☐ 1-24 in 5-year groups; 25-65+ in 10-year groups  
☐ 1-14 in 5-year groups; 15-85+ in 10-year groups  
☐ Custom Age Range <1 to <1

Submit Request Reset

(a) Query for top 10 causes of death

## 10 Leading Causes of Death, United States 2007, All Races, Both Sexes

[Click on any colored box for detailed causes and ICD codes.](#)

[Click on any age group for percentages.](#)

Rank	Age Groups										All Ages
	<1	1-4	5-9	10-14	15-24	25-34	35-44	45-54	55-64	65+	
1	Congenital Anomalies 5,785	Unintentional Injury 1,588	Unintentional Injury 965	Unintentional Injury 1,229	Unintentional Injury 15,897	Unintentional Injury 14,977	Unintentional Injury 16,931	Malignant Neoplasms 50,167	Malignant Neoplasms 103,171	Heart Disease 496,095	Heart Disease 616,067
2	Short Gestation 4,857	Congenital Anomalies 546	Malignant Neoplasms 480	Malignant Neoplasms 479	Homicide 5,551	Suicide 5,278	Malignant Neoplasms 13,288	Heart Disease 37,434	Heart Disease 65,527	Malignant Neoplasms 389,730	Malignant Neoplasms 562,875
3	SIDS 2,453	Homicide 398	Congenital Anomalies 196	Homicide 213	Suicide 4,140	Homicide 4,758	Heart Disease 11,839	Unintentional Injury 20,315	Chronic Low Respiratory Disease 12,777	Cerebro-vascular 115,961	Cerebro-vascular 135,952
4	Maternal Pregnancy Comp. 1,769	Malignant Neoplasms 364	Homicide 133	Suicide 180	Malignant Neoplasms 1,653	Malignant Neoplasms 3,463	Suicide 5,722	Liver Disease 8,212	Unintentional Injury 12,193	Chronic Low Respiratory Disease 109,562	Chronic Low Respiratory Disease 127,924
5	Unintentional Injury 1,285	Heart Disease 173	Heart Disease 110	Congenital Anomalies 178	Heart Disease 1,084	Heart Disease 3,223	HIV 3,572	Suicide 7,778	Diabetes Mellitus 11,304	Alzheimer's Disease 73,797	Unintentional Injury 123,706
6	Placenta Cord Membranes 1,135	Influenza & Pneumonia 109	Chronic Low Respiratory Disease 54	Heart Disease 131	Congenital Anomalies 402	HIV 1,091	Homicide 3,052	Cerebro-vascular 6,385	Cerebro-vascular 10,500	Diabetes Mellitus 51,528	Alzheimer's Disease 74,632
7	Bacterial Sepsis 820	Septicemia 78	Influenza & Pneumonia 48	Chronic Low Respiratory Disease 64	Cerebro-vascular 195	Diabetes Mellitus 610	Liver Disease 2,570	Diabetes Mellitus 5,753	Liver Disease 8,004	Influenza & Pneumonia 45,941	Diabetes Mellitus 71,382
8	Respiratory Distress 789	Perinatal Period 70	Benign Neoplasms 41	Influenza & Pneumonia 55	Diabetes Mellitus 168	Cerebro-vascular 505	Cerebro-vascular 2,133	HIV 4,156	Suicide 5,069	Nephritis 38,484	Influenza & Pneumonia 52,717
9	Circulatory System Disease 624	Benign Neoplasms 59	Cerebro-vascular 38	Cerebro-vascular 45	Influenza & Pneumonia 163	Congenital Anomalies 417	Diabetes Mellitus 1,984	Chronic Low Respiratory Disease 4,153	Nephritis 4,440	Unintentional Injury 38,292	Nephritis 46,448
10	Neonatal Hemorrhage 597	Chronic Low Respiratory Disease 57	Septicemia 36	Benign Neoplasms 43	Three Tied 160	Liver Disease 384	Septicemia 910	Viral Hepatitis 2,815	Septicemia 4,231	Septicemia 26,362	Septicemia 34,828

(b) Results produced from the query

help themselves in program planning and evaluation activities. Similarly, the policy makers can seek help from this module while developing injury-prevention programs while the legislation can reference the maps from this module to illustrate injury-related death rate programs in their respective areas. Last but not the least, this module proves helpful in identifying county level geographical injury patterns which steer the motor vehicle safety interventions in the regions with the greatest need. Figure 2.2 displays a custom query created to map the transportation related fatal injuries with all intents for 'White' ethnicity males within 20-60 age group in Nevada from 2000-2006.

#### 2.4.1.2 Non Fatal Injury Data

WISQARS also maintains a database for non fatal injuries. This database can be used to create reports which provide the following:

1. National estimates of injuries treated in U.S. hospital emergency departments by intent and cause of injury, ethnicity, gender and disposition when released from the emergency department.
2. Leading causes of non fatal injuries treated in emergency departments by age, gender, intent of injury and disposition when released from the emergency department.

Figure 2.3 shows a custom query created to query this database for finding the leading causes of the non fatal injuries irrespective of the gender over 2001-2009. A subset of the results has been shown, thus focussing upon the people within 15-24 years of age.



Figure 2.2: Wisqars: Interactive Fatal Injury Mapping Module

CDC Home  
Centers for Disease Control and Prevention  
Your Online Source for Credible Health Information

A-Z Index A B C D E F G H I J K L M N O P Q R S T U V W X Y Z #

WISQARS™

WISQARS Home > Fatal Injury Queries

### Fatal Injury Mapping

Help

**Injury Type Options**

All Injury/TBI: All Injury

Intent of Injury: All Intent

Mechanism of Injury: Transportation-Related, Overall

**Scope of Map**

Census Region/State: Nevada

Years of Report: 2000-2006

**Demographic Subsetting Options**

Race: White

Hispanic Origin: All Ethnicities

Sex: Males only

☐ All Ages (includes unknown age)

☐ Age Groups: 0-4 To 0-4

☒ Custom Age Range: 20 To 60

**Death Rate Calculation Options**

Type of Rate: ☒ Crude Rates ☐ Age-adjusted Rates

Map Level: ☐ State ☒ County

Geospatial smoothing: ☐ No smoothing ☒ Smoothing

**Cost-of-Injury Option**

Medical/Work Loss Costs: ☒ No Cost Estimates ☐ Cost Estimates

**Map Detail Options**

Interval Type: Quantiles (rankings)

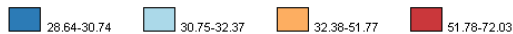
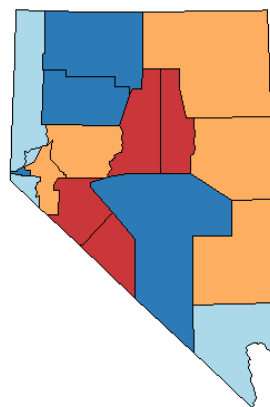
Number of Intervals: 4

Color Scheme: Blue-Yellow-Red (Div)

Submit / Generate a New Map

(a) Custom query for fatal injury mapping

**2000-2006, Nevada**  
Smoothed Death Rates per 100,000 Population  
Transportation-Related, Overall, All Intent, White, All Ethnicities, Males, Ages 20-60 Years  
Annualized Crude Rate for Nevada: 30.94



Reports for All Ages include those of unknown age.  
\* Rates based on 20 or fewer deaths may be unstable. These rates are suppressed for counties (see legend above); such rates in the title have an asterisk.  
Rates appearing in this map have been geospatially smoothed.

Produced by: Office of Statistics & Programming, National Center for Injury Prevention & Control, CDC  
Data Sources: NCHS National Vital Statistics System for numbers of deaths; US Census Bureau for population estimates.

(b) Results produced from custom query

Figure 2.3: Wisqars: Non fatal injury database

#### WISQARS Leading Causes of Nonfatal Injury Reports

[Pick a quick report.](#)

This option offers 9 choices of common reports on 10 leading causes of nonfatal injury, already compiled and ready for viewing.

OR

[Request a customized report.](#)

This option allows you to make a variety of choices and request a report tailored to your needs.

#### Pick a Quick Report

Select any of the common 10 Leading Causes of Nonfatal Injury Reports.

[All Injuries](#)

[All Injuries: Males](#)

[All Injuries: Females](#)

[Unintentional Injuries](#)

[Unintentional Injuries: Males](#)

[Unintentional Injuries: Females](#)

[Violence-Related](#)

[Violence: Males](#)

[Violence: Females](#)

OR

#### Request a Customized Report

Choose your Report Options, then click the Submit Request button.

For more information about a category of report options, click the underlined category name.

[Sex](#)

Both Sexes ▼

[Year\(s\) of Report](#)

2001 ▼ to 2009 ▼

[Disposition](#)

Where the injured person went when released from emergency department)

All Cases ▼

[Intent of Injury](#)

All Injuries ▼

[Number of Causes](#)

Top 10 ▼

[Display Options](#)

Standard Output ▼

Submit Request

Reset

(a) Custom query for non fatal injuries

#### 10 Leading Causes of Nonfatal Injuries, United States

2001 - 2009, All Races , Both Sexes , Disposition: All Cases

Ages: 15-24

Cause of Injury	Number of Injuries	Percentage of All Injuries in Age Group
Overall Injuries	53,703,560	100.0%
Unintentional Struck By/Against	9,058,099	16.9%
Unintentional Fall	7,898,807	14.7%
Unintentional MV-Occupant	7,637,094	14.2%
Unintentional Overexertion	6,845,168	12.7%
Unintentional Cut/Pierce	4,381,743	8.2%
Other Assault <sup>A</sup> Struck By/Against	4,118,362	7.7%
Unintentional Unknown/Unspecified	1,552,420	2.9%
Unintentional Other Bite/Sting	1,536,018	2.9%
Unintentional Other Specified	1,513,459	2.8%
Unintentional Other Transport	1,206,117	2.2%
All Others*	7,956,274	14.8%

[Download Results in a Spreadsheet \(CSV\) File](#)

[Help with Download](#)

(b) Results produced from custom query

#### 2.4.1.3 Violent Death Data

The WISQARS database provides its users the functionality to generate the reports which contain the following:

1. Number of violent-related deaths and death rates.
2. Intent and cause of injury
3. Details about victim and offenders, including demographics, victim-offender relationship etc.
4. Details about suicide victims suspected of a recent homicide.

In 2002, the Centers for Disease Control and Prevention started the implementation of National Violent Death Reporting System (NVDRS). NVDRS is a state-based surveillance system that connects the data from law enforcement, medical examiners and crime laboratories and assist the participating states to route their prevention efforts. According to Centers for Disease Control & Prevention 2008, a death due to violence is defined as "a death resulting from the intentional use of physical force or power against oneself, another person, or against a group or community". NVDRS collects information about homicides, suicides, deaths by legal intervention-excluding executions-and deaths of undetermined intent. It currently covers 18 states namely Alaska, Colorado, Georgia, Kentucky, Maryland, Massachusetts, Michigan, New Jersey, New Mexico, North Carolina, Ohio, Oklahoma, Oregon, Rhode Island, South Carolina, Virginia, Utah and Wisconsin.

Figure 2.4 shows a custom query designed to operate on the Violent death database for finding the number of violent deaths in 16 participating states (not containing Michigan and Ohio yet) due to motor vehicle crashes.

Figure 2.4: Wisqars: Violent Death database

#### National Violent Death Reporting System (NVDRS)

Choose your **Report Options**, then click the **Submit** button.

For more information about an option or a category of options, click on the underlined name or phrase. To return to this page, click on the "back" button in your browser toolbar. Refreshing this page between queries may expedite your request.

##### Report Options

##### 1. Select a report type. (Select only one button from the eight options below.)

- Victims of Violence**
  - ☒ Violent Death Counts and Rates
  - ☐ Violent Death Counts and Percentages by Place of Injury, Pregnancy Status, Homeless Status, Military Status, and Known Circumstances of Death
- Suicide Victims Suspected of a Recent Homicide**
  - ☐ Suicide Counts and Rates
  - ☐ Suicide Counts and Percentages by Place of Injury, Pregnancy Status, Homeless Status, Military Status, and Known Circumstances of Death
- Suspects**
  - ☐ Suspects' Counts and Percentages only
- Incidents of Violence**
  - ☐ All Violent Incident Counts and Percentages
  - ☐ Single-Victim Violent Incident Counts and Percentages
  - ☐ Multiple-Victim Violent Incident Counts and Percentages

##### Deaths and Rates

- ☒ Age-adjusted Rates, Crude Rates and Death Counts
- Use  as the Standard Year
- ☐ Crude Rates and Death Counts

##### 2. Select a mode of determining a manner and cause of death. (Select only one radio button.)

- ☒ Abstracter Assigned (*recommended*)
- ☐ ICD-10 Underlying Cause of Death Codes

##### 3. What was the intent or manner of the injury? (Select one or more boxes.)

- ☒ All Intents (Uncheck box to select sub-groups)
  - ☐ Unintentional firearm
  - ☐ Homicide
  - ☐ Legal Intervention
  - ☐ Suicide
  - ☐ Undetermined intent
  - ☐ Homicide followed by Suicide

(a) Querying Violent Death (NVDRS) database

2008, 16 NVDRS States: AK, CO, GA, KY, MD, MA, NJ, NM, NC, OK, OR, RI, SC, UT, VA, WI  
All Victims Death Counts and Rates per 100,000, Abstracter Assigned Mode  
All Intents, Motor vehicle  
All Races, Both Sexes, All Ages

Number of Deaths	Population	Crude Rate	Age-Adjusted Rate**
176	80,842,827	0.22	0.21

Reports for All Ages include those of unknown age.  
 \* No population estimate/rate not applicable.  
 \* Numbers of deaths 20 or fewer are shown in white. Rates based on these numbers are also shown in white and may be unstable. Use with caution.  
 \*\* Standard Population is 2000, all races, both sexes.  
 Data Sources: National Violent Death Reporting System (NVDRS) for Number of Deaths, Bureau of Census for Population Estimates.  
 Produced by: Office of Statistics and Programming, National Center for Injury Prevention and Control, CDC

(b) Results produced from the query

#### 2.4.1.4 Cost of Injury Data

The WISQARS database provides cost estimates for injury deaths(including violent deaths) and nonfatal injuries where the patient was treated and released from the hospital. The reports generated contain the following information:

1. Medical costs (e.g., treatment and rehabilitation)

2. Work loss costs (e.g., lost wages, benefits, and self-reported household services)
3. Combined costs (medical plus work loss)

The users have the freedom to choose from the following options: geographic coverage for fatal and non fatal injuries, gender and ethnicity of injured persons and the type of lifetime costs. Figure 2.5 shows one of the three screens for querying the database to estimate the costs incurred throughout US during the year 2005 (the only one available) in the fatal injuries for all age groups irrespective of gender where the causes of injury were limited to: Motor Vehicle, Traffic (containing Motorcyclists, Occupants, Pedal Cyclists and Pedestrians). Due to the limited size of the screen, only a part of the costs taken over all the age groups could be displayed in Figure 2.5.

#### 2.4.2 Trauma Data Analysis systems

The people who undergo severe motor vehicle accidents or get wounded through gunshots, knives and burns can easily get intensely traumatized. In such cases, the survival of the trauma victims solely depends upon how strong the trauma system is in the vicinity of the accident location. The trauma system must be able to provide immediate medical assistance to the victims. Most importantly, the trauma system should have a low response time for immediately sending the medical facilities to the scene of injury, thereby bringing the victim back as soon as possible so that he/she can be attended by well qualified doctors.

The American Trauma Society has been doing a great job, taking care of the trauma victims and being a leader in trauma prevention since the past 30 years. This

Figure 2.5: Wisqars: Cost of Injury Database

## Data & Statistics (WISQARS™): Cost of Injury Reports

[WISQARS Home](#) > [Fatal Injury Queries](#)

Help

Screen 2 of 3

[< Back](#) [Go to Next Screen >](#)

[Reset Screen](#)

### Mechanism Level

Indicate level of detail for Mechanism (select only one radio button):

- ☐ Mechanism Level 1 (All Mechanisms combined)  
☒ Mechanism Level 2 (e.g., 'Fall', 'Motor Vehicle Traffic')  
☐ Mechanism Level 3 (e.g., 'Fire/Flame', 'Motor Vehicle Traffic Occupant')  
☐ Mechanism Level 4 (Residential Fire/Flame)

### Intent

- ☒ All Intents  
☐ Unintentional  
☐ Violence-related  
☐ Homicide  
☐ Legal Intervention  
☐ Suicide  
☐ Undetermined

### Mechanism

- ☐ All Mechanisms of Injury  
☐ Cut/Pierce  
☐ Drowning/Submersion  
☐ Fall  
☐ Fire/Burn  
☒ Fire/Flame  
☐ Residential Fire/Flame  
☐ Hot Object/Substance  
☐ Firearm  
☐ Machinery  
☐ Natural/Environmental  
☐ Overexertion  
☐ Poisoning  
☐ Struck By/Against  
☐ Suffocation  
☐ Transportation  
☒ Motor Vehicle, Traffic  
☐ Motorcyclist  
☐ Occupant  
☐ Pedal Cyclist  
☐ Pedestrian

(a) Querying Cost of Injury database

## Data & Statistics (WISQARS™): Cost of Injury Reports

[Printable View](#) [Export Data](#)

Fatal Injuries, Both Sexes, All Ages, United States, 2005

Intent: All

Mechanism: Motor vehicle, Traffic, PedalCycl-Other/Pedestrian-Other

Number of Deaths and Estimated Lifetime Costs Classified by Mechanism and Age group

Costs Expressed in Year 2005 United States Prices

Deaths and Type of Cost		Age									
		00-04	05-09	10-14	15-19	20-24	25-29	30-34	35-39	40-44	45-49
Motor Vehicle - Traffic	Deaths	--	629	560	763	4,829	5,828	3,825	3,222	3,050	3,441
	Medical Cost	Average	\$9,338	\$13,283	\$11,772	\$6,985	\$6,191	\$7,867	\$7,798	\$7,967	\$8,028
	Total		\$5,873,000	\$7,439,000	\$8,982,000	\$33,731,000	\$36,079,000	\$30,093,000	\$25,125,000	\$24,300,000	\$27,626,000
	Work Loss Cost	Average	\$928,717	\$1,034,799	\$1,159,441	\$1,275,877	\$1,351,160	\$1,341,666	\$1,248,680	\$1,115,493	\$959,242
	Total		\$584,163,000	\$759,488,000	\$884,653,000	\$6,161,209,000	\$7,874,562,000	\$5,131,874,000	\$4,023,247,000	\$3,402,253,000	\$3,300,752,000
	Combined Cost	Average	\$938,054	\$1,048,082	\$1,171,212	\$1,282,862	\$1,357,351	\$1,349,534	\$1,256,478	\$1,123,460	\$967,270
Pedal Cyclist - Other/Pedestrian - Other	Deaths	--	133	29	35	75	70	75	80	85	110
	Medical Cost	Average	\$5,564	\$11,505	\$21,181	\$6,458	\$5,976	\$4,928	\$7,673	\$5,955	\$9,365
	Total		\$740,000	\$334,000	\$741,000	\$484,000	\$418,000	\$370,000	\$614,000	\$506,000	\$1,030,000
	Work Loss Cost	Average	\$931,954	\$1,955,588	\$1,175,026	\$1,297,841	\$1,376,737	\$1,361,939	\$1,246,283	\$1,121,046	\$965,769
	Total		\$123,960,000	\$30,612,000	\$41,126,000	\$97,336,000	\$96,372,000	\$102,145,000	\$99,703,000	\$95,289,000	\$106,235,000
	Combined Cost	Average	\$937,518	\$1,067,093	\$1,196,206	\$1,304,299	\$1,382,714	\$1,366,867	\$1,253,956	\$1,127,001	\$975,134
Total	Deaths	--	762	589	798	4,904	5,898	3,900	3,302	3,135	3,551
	Medical Cost	Average	\$8,679	\$13,196	\$12,184	\$6,977	\$6,188	\$7,811	\$7,795	\$7,913	\$8,070
	Total		\$6,613,000	\$7,772,000	\$9,723,000	\$34,215,000	\$36,497,000	\$30,463,000	\$25,738,000	\$24,806,000	\$28,656,000
	Work Loss Cost	Average	\$929,282	\$1,035,823	\$1,160,124	\$1,276,213	\$1,351,464	\$1,342,056	\$1,248,622	\$1,115,644	\$959,444
	Total		\$708,113,000	\$610,100,000	\$925,779,000	\$6,258,547,000	\$7,970,934,000	\$5,234,019,000	\$4,122,950,000	\$3,497,542,000	\$3,406,986,000
	Combined Cost	Average	\$937,961	\$1,049,018	\$1,172,308	\$1,283,190	\$1,357,652	\$1,349,867	\$1,256,417	\$1,123,556	\$967,514

(b) Results produced from the query

organization maintains an online database which geographically keeps track of the hospitals and trauma centers throughout US, so that the specified accident location can be matched with the database and the medical team which falls within a given response time (45 or 60 minutes) can be dispatched. If a medical facility is not close to the accident location such that an ambulance won't be able to reach the designated location within the required response time, the system also provides the ability to estimate the area that can be covered by an ambulance or more importantly, a helicopter within that response time.

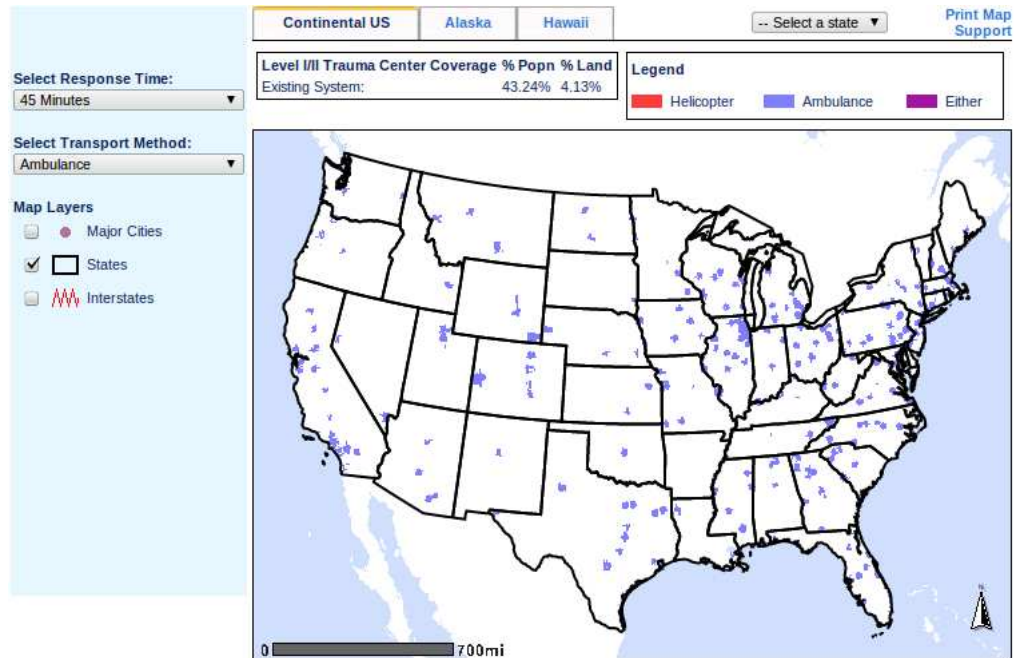
Figure 2.6 shows the difference between the areas covered throughout US within 45 and 60 minutes response time. By increasing the response time from 45 to 60 minutes, notice how the area covered by the ambulances throughout US increases.

This web based system can be used to provide the details of the locations of hospitals and trauma centers at the state level too, with the most recent update in 2009. Figure 2.7 provides a state level and a zoomed display of the medical facilities available in Nevada, with the legend provided on the left panel. The blue region represents the area that can be covered by the ambulances within 45 minutes response time. This figure also shows the location of two trauma centers around the vegas area.

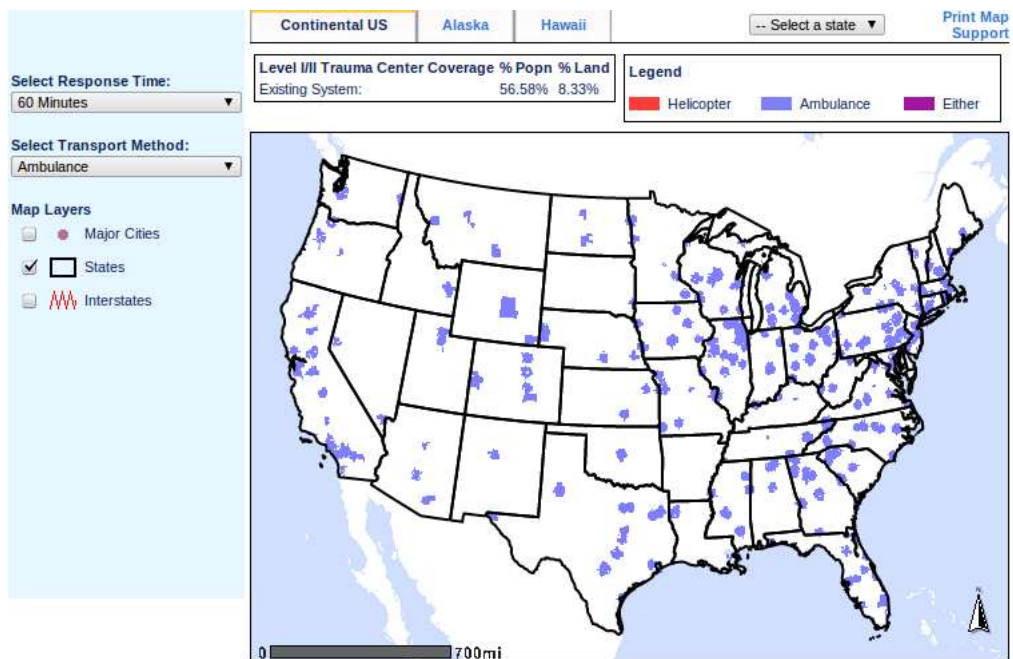
#### 2.4.3 UMC and NDOT: Integrated Trauma and Crash Data

In Nevada, the varying distributions of population and fatality crashes make it crucial to study the fatality trends over the years. For this purpose, the University Medical Center(UMC) has prepared an integrated repository of the NDOT crash data

Figure 2.6: American Trauma Center: Variation in areas covered with different response times



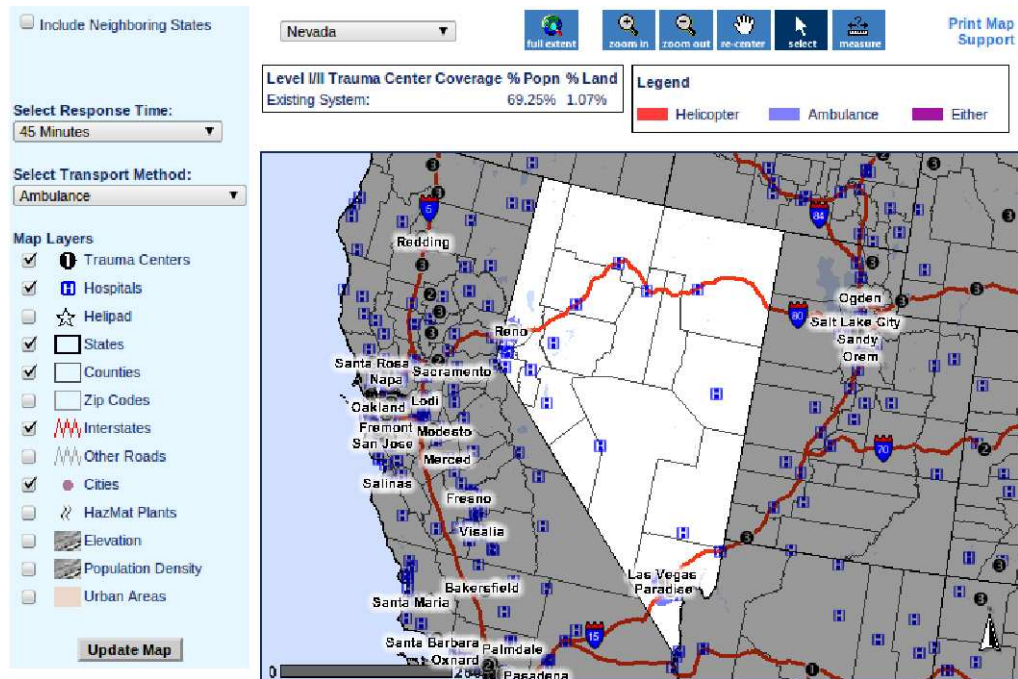
(a) Area covered with 45 minutes response time



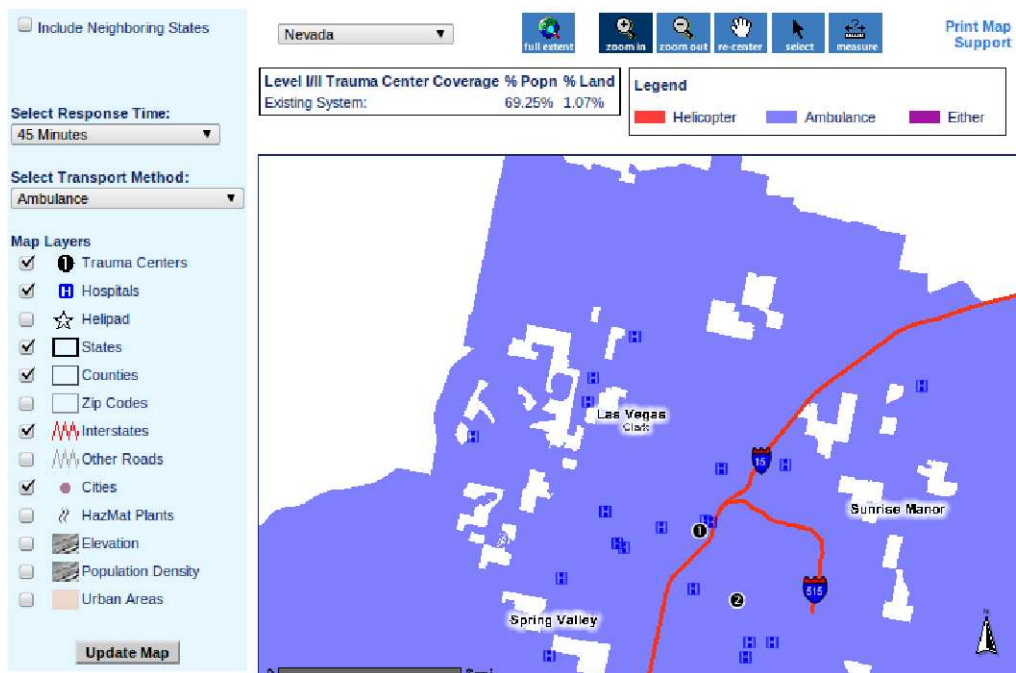
(b) Area covered with 60 minutes response time



Figure 2.7: American Trauma Center: Hospitals and Trauma Centers in Nevada

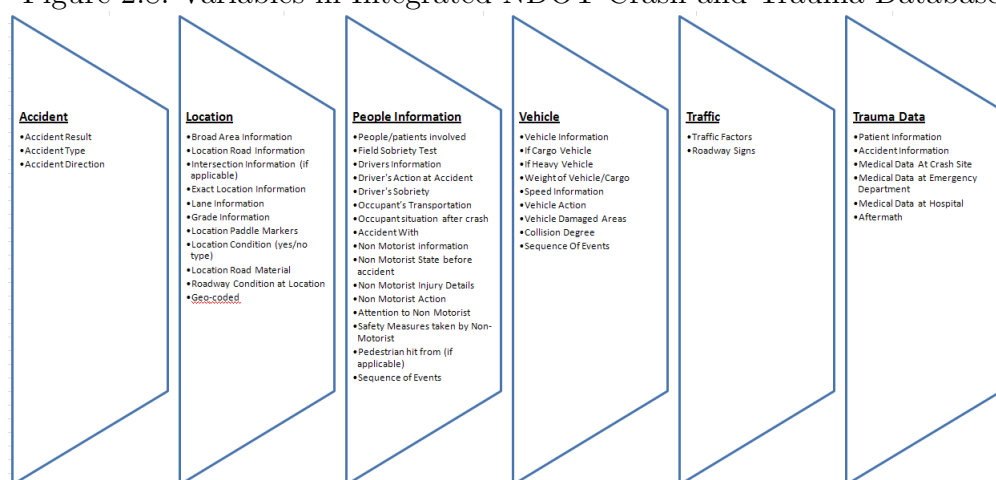


(a) Medical facilities in Nevada



(b) Medical facilities around Las Vegas area

Figure 2.8: Variables in Integrated NDOT Crash and Trauma Database



and the UMC trauma data from 2003-2008. The trauma data from UMC provides a lot of information about a wide variety of parameters: patient's city, race, sex, injury city, injury date, patient's condition at the scene, patient's condition upon arrival at the hospital, safety equipment usage, injury severity score, probability of survival, duration in hospital/icu, insurance, hospital charges etc. Similarly, the NDOT crash data provides vital information about the accident, the location, the injured person, vehicle and traffic conditions. Currently, the Transportation Research Center at UNLV is currently working on refining the linkage between these variables. In a nutshell, the work is being done to connect the following variables as shown in Figure 2.8.

The Transportation Research Center at UNLV is building an online database which will enable the users and analysts to create custom queries and generate tables and graphs based upon the options provided. Although the back-end of this online database has been developed, work is currently being done to create an attractive front-end, preferably using Adobe Flex 3/4 technologies that will be discussed later

in the thesis. The backend of this database has the following features:

1. Access to users with respective passwords, though guest access is also provided.
2. Choice of running a predefined analysis or loading the data for a customized analysis.
3. In case of running predefined analysis, options are available to show the results in textual or a graphical format, along with enabling the users to save the generated images.
4. For the customized analysis, the users are able to view and select options from a pool of variables from multiple databases at once.
5. The basic statistical analysis is being performed using R-PHP at the backend.

The next couple of figures show the designed R-PHP backend for the combined crash data and trauma data analysis. After logging onto the main screen (or continuing as a guest), the user is asked to select the analysis method. Next, the user is required to select the variables in which he/she is interested in. This process is shown in Figure 2.9. The generated textttual output and the graphs from the basic statistical analysis are shown in Figure 2.10.

With this, we finish the discussion on the analysis models and the visual systems being used to explore Injury Severity. In the next chapter, we discuss the saftey belts and the associated studies, which aim to reduce the levels of injury in motor vehicle crashes.

Figure 2.9: R-PHP backend: Selecting analysis and variables

http://localhost/Crash\_Data\_PHP/analyze.php

Welcome User : Guest [Logout](#)

**Predefined Analysis**

- ☐ Persons aged 16 through 20 involved in crashes
- ☐ Hospitalized Driver's Injuries by Age Group
- ☐ Injury Rates and Outcome for Males by Age in Crashes
- ☐ Injury Rates and Outcome for Females by Age in Crashes
- ☐ Injury Outcome by Month of Crash for Pedestrians in Crashes
- ☐ Injury Outcome for Pedestrians Involved in Alcohol-Related Crashes

[Submit](#)

**Load Data for New Analysis**

- ☐ Linked NDOT and Trauma Data
- ☐ NDOT Collision Data
- ☐ NDOT Scene Conditions Data
- ☐ NDOT Locations Data
- ☐ NDOT Occupant Data
- ☐ NDOT Party Data
- ☐ NDOT Vehicle Data
- ☐ Complete Linked Data

[Submit](#)

(a) Selecting the method of analysis

http://localhost/Crash\_Data\_PHP/load.php

<input type="checkbox"/> Driverless Vehicle	<input type="checkbox"/> 100 Fast for Conditions	<input type="checkbox"/> Exceeding Speed Limit	<input type="checkbox"/> Object Avoidance	<input type="checkbox"/> Wrong Way or
<input type="checkbox"/> Disregard Control Device	<input type="checkbox"/> Driver inattention or Distracted	<input type="checkbox"/> Driver inattention or Distracted Text	<input type="checkbox"/> Fail to yield Right of Way	<input type="checkbox"/> Other Improper
<input type="checkbox"/> Speed Zone	<input type="checkbox"/> Signal Light	<input type="checkbox"/> Flashing light	<input type="checkbox"/> Other Description	<input type="checkbox"/> School Zone
<input type="checkbox"/> Signal Light MultiPick	<input type="checkbox"/> Flashing light MultiPick	<input type="checkbox"/> School Zone MultiPick	<input type="checkbox"/> Pedestrian Signal MultiPick	<input type="checkbox"/> No Passing Zone
<input type="checkbox"/> Yield Sign MultiPick	<input type="checkbox"/> NDOT Sex	<input type="checkbox"/> C City	<input type="checkbox"/> State	<input type="checkbox"/> Zip
<input type="checkbox"/> Transported By Police	<input type="checkbox"/> Transported By Other	<input type="checkbox"/> Transported By Unknown	<input type="checkbox"/> Transported By Not Transported	<input type="checkbox"/> Transported
<input type="checkbox"/> BB Restraint	<input type="checkbox"/> Injury Severity	<input type="checkbox"/> Airbags	<input type="checkbox"/> Ejected	<input type="checkbox"/> Seating Position
<input type="checkbox"/> Trapped	<input type="checkbox"/> Driver	<input type="checkbox"/> Injury Location Pick1	<input type="checkbox"/> Injury Location Pick2	<input type="checkbox"/> Non Motorist
<input type="checkbox"/> Insured	<input type="checkbox"/> Minor	<input type="checkbox"/> Major	<input type="checkbox"/> None	<input type="checkbox"/> Estimated Speed
<input type="checkbox"/> Speed Limit	<input type="checkbox"/> Sequence Of Events 1	<input type="checkbox"/> Most Harmful Event	<input type="checkbox"/> Suspected Impairment	<input type="checkbox"/> Alcohol
<input type="checkbox"/> Drugs	<input type="checkbox"/> Driver Admission	<input type="checkbox"/> Field Sobriety Test	<input type="checkbox"/> Evidentiary Breath	<input type="checkbox"/> Urine Test
<input type="checkbox"/> Blood Test	<input type="checkbox"/> Blood Test Result	<input type="checkbox"/> AlcoholDrugNotInvolved		

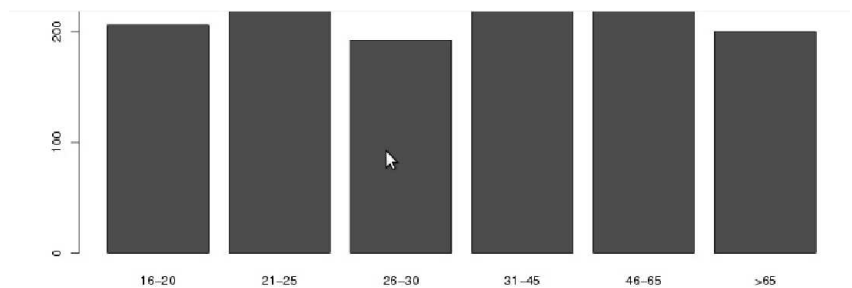
[Submit](#)

[New Dataset](#)

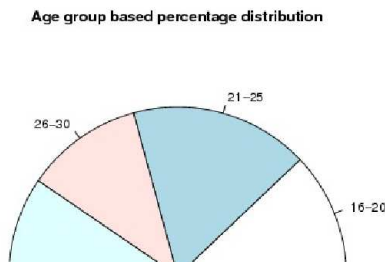
Add new variables: [New Variables](#)

(b) Selecting the variables

Figure 2.10: R-PHP backend: Results in graphical and textual format



 [Save Image](#)



(a) Results in Graphical format

```
[1] "Mean Age of the data is"
[1] 37.77166
[1] "Hospitalized Driver\342(200)231s Injuries by Age Group - This study evaluates the increased risk for injury from a motor vehicle crash by age group, particularly the elderly population. All drivers were assigned to groups covering a span of 5 years (16-20, 21-25, etc.). Factors contributing to the crash, such as use of protective devices were considered. Type of injury, length of stay, and hospital charges were compared for each age group"
newvalue
Age 16 20 2.200000e+01
percent_Age 16 20 4.638415e-01
Age 21 25 3.600000e+01
percent_Age 21 25 7.590133e-01
Age 26 30 2.100000e+01
percent_Age 26 30 4.427577e-01
Age 31 45 4.600000e+01
percent_Age 31 45 9.698503e-01
Age 46 65 5.700000e+01
percent_Age 46 65 1.201771e+00
Age above 65 3.100000e+01
percent_Age above 65 6.535948e-01
Avg_hosp_charges 16 20 6.714434e+04
hosp_charges 16 20 1.188455e+07
Avg_hosp_days 16 20 6.073446e+00
hosp_days 16 20 1.075000e+03
Avg_icu_days 16 20 1.632768e+00
icu_days 16 20 2.890000e+02
count 16 20 1.770000e+02
Avg_hosp_charges 21 25 6.712897e+04
hosp_charges 21 25 1.718502e+07
Avg_hosp_days 21 25 5.468750e+00
hosp_days 21 25 1.400000e+03
Avg_icu_days 21 25 1.785156e+00
icu_days 21 25 4.570000e+02
count 21 25 2.560000e+02
```

(b) Results in Textual format

## CHAPTER 3

### DAYTIME SAFETY BELT USAGE STUDIES

#### Abstract

Safety Belts are very important for the protection of motor vehicle occupants. Several studies in the research literature can be found which emphasize their crucialness. The U.S. government supports safety belt campaigns every year, specifically 'Click It or Ticket'. During this campaign, huge monetary resources are utilized to make the general public aware of importance of safety belts. Therefore, it is essential to conduct safety belt surveys that can estimate the current safety belt usage trends and suggest recommendations to further enhance the campaign success. In this chapter, the theory behind these Daytime Safety Belt Usage Surveys in Nevada has been provided. Further, the collected data is analyzed statistically to compare the safety belt usage before and after the Daytime Safety Belt Usage Survey in 2010.

#### 3.1 Importance of Safety Belts

Safety belts are one of the most important devices designed in the car to protect the driver from the car accidents. According to Martinez (Spring 1998), 'Safety belts are your first line of defense against injuries or death'. When crashes occur, unrestrained drivers are thrown against their steering wheels or ejected from their cars, while unbelted passengers hit the dashboard or go through the windshield. According to the Traffic Safety Marketing 2009, safety belts have saved around 12,713 lives nationwide in 2009. Another documentation provided by the National Center for Statistics and Analysis under NHTSA (2010) revealed that the safety belt usage

in passenger vehicles saved an estimated 13,250 lives in 2008. Aggregating over 2004-2008, it was reported that 75,000 lives have been saved due to the use of safety-belts. Table 3.1 extracted from the above report shows the mentioned trends over 2004-2008.

Table 3.1: Lives saved by Restraint Use from 2004-2008 (Source: FARS 2004-2007 Final File, 2008 Annual Report File)

Year	Lives saved, Age 4 and younger	Lives saved, Age 5 and older	Lives saved if 100% use
	Child restraints	Seat Belts	Seat belts
2004	455	15,548	5,874
2005	424	15,688	5,667
2006	427	15,458	5,468
2007	388	15,223	5,048
2008	244	13,250	4,152

Thus, the facts clearly state the importance of safety belts. To increase the usage of safety belts on a nationwide level, U.S. government organizes a 'Click It or Ticket' (CIOT) Mobilization every year. The 'Click It or Ticket' (CIOT) mobilization, launched in North Carolina during 1993, is a National Highway Traffic Safety Administration mobilization which aims at increasing the use of safety belts among the people in U.S. Targetted advertising and enforcement forms the backbone of this mobilization and monitors closely the safety belt usage of teens and young adults.

### 3.2 Daytime Safety Belt Usage Surveys

As mentioned earlier, the CIOT mobilization is held on a nationwide level in U.S. every year. This usually happens during the month of May, when law enforcement agencies join hands with the statewide safety offices to increase the safety belt usage awareness among the people. Daytime Safety Belt Usage Surveys are conducted before and after this period on statewide levels, These surveys not only look at estimating the safety belt usage rates before and after the mobilization period, but also aim

at producing in-depth results which can be helpful for designing the mobilization in future. Such surveys are also conducted in Nevada every year during the months of April and June. The safety belt usage rates during the Pre-Mobilization and Post-Mobilization periods are estimated and then compared over several categories: Roadway Type (Interstate, Arterial, Collector), Area (Urban/Rural) etc. The next section presents the details of the Daytime Safety Belt Usage Surveys in Nevada during the year 2010.

### 3.3 Daytime Safety Belt Usage Surveys in Nevada

In 2010, Nevada Department of Public Safety (DPS), Office of Traffic Safety (OTS) contracted with the Transportation Research Center (TRC) at UNLV to conduct Pre-Mobilization and Post-Mobilization daytime safety belt usage surveys throughout Nevada. In this chapter, the procedural details for the survey design will first be discussed. Later, a few snapshots of the conducted basic and statistical analysis will be demonstrated and then the conclusions will be briefly discussed.

#### 3.3.1 Site Selection

Nevada developed its initial safety belt use survey design in 1999. The methodology was approved by the National Highway Traffic Safety Administration (NHTSA). Due to a significant change in population, especially in the urban areas of Nevada, the survey methodology was re-designed in 2007 to meet the Code of Federal Regulations (CFR) specified in 23 CFR Part 1340, 1998 (also known as Section 157 surveys). The two stage design process, along with the most recent data, is given in the following section.



### 3.3.2 Survey Design: Stage One

Geographically, Nevada is the seventh largest state with 110,540 square miles. Clark County, which includes the Las Vegas metropolitan area, is the largest county in Nevada in terms of population. It accounts for over 71 percent of the population of the state. The second most populated county is Washoe County. These two counties account for over 87 percent of the state population. Table 3.2 lists the population distribution of Nevada by county.

Table 3.2: List of Counties in Nevada by Population(Source: US Census Bureau, 2008)

County	Pop. 2008	Pop. change 2000-08	Pop. %	Cumulative %
<b>Clark</b>	1,865,746	35.6%	71.75%	<b>71.75%</b>
<b>Washoe</b>	410,443	20.9%	15.79%	<b>87.54%</b>
Carson City	54,867	4.6%	2.11%	89.65%
Lyon	53,022	53.7%	2.04%	91.69%
Douglas	45,180	9.5%	1.74%	93.43%
Elko	47,071	3.9%	1.81%	95.24%
Nye	44,375	35.6%	1.71%	96.95%
Churchill	24,896	3.8%	0.95%	97.9%
Humboldt	17,763	10.3%	0.68%	98.58%
White Pine	9,199	0.2%	0.35%	98.93%
<i>continued on next page</i>				

<i>continued from previous page</i>				
County	Pop. 2008	Pop. change 2000-08	Pop. %	Cumulative %
Pershing	6,291	-6%	0.24%	99.17%
Lander	5,086	-12.2%	0.19%	99.36%
Mineral	4,684	-7.6%	0.18%	99.54%
Storey	4,341	27.7%	0.16%	99.7%
Lincoln	4,898	17.6%	0.18%	99.88%
Eureka	1,628	-1.4%	0.0006%	99.9
Esmeralda	677	-30.3%	0.0002%	99.9
<b>Total</b>	2,600,167	-	<b>~ 100%</b>	-

In Uniform criteria (1998), it was mentioned that the counties which account for less than 15% of the state's population can be eliminated. For Nevada, over 87 percent of it's population lives within two of its counties; Clark and Washoe. Hence, these two counties can be selected as Primary Sampling Units (PSUs) for the observations of safety belt usage in Nevada.

In Table 3.3, 2008 Annual Vehicle Miles Travelled (AVMT) data has been given. Table 3.3 shows that Clark and Washoe counties account for over 80 percent of annual vehicle miles of travel in Nevada. Since Clark and Washoe also satisfy the 85 percent of total population criterion, no further stage 1 sampling is required.

Table 3.3: List of Counties in Nevada with AVMT in 2008(Source:Nevada DOT,2008)

County	AVMT(millions)	Change from 2005-08	AVMT %	Cumulative %
<b>Clark</b>	13,802	3.3%	65.66%	<b>65.66%</b>
<b>Washoe</b>	3,252	-2.13%	15.47%	<b>81.13%</b>
Carson City	360	-10.44%	1.72%	82.85%
Lyon	479	4.2%	2.28%	85.13%
Douglas	518	-7.6%	2.47%	87.6%
Elko	660	0.76%	3.14%	90.74%
Nye	374	-2.6%	1.78%	92.52%
Churchill	290	-1.36%	1.38%	93.9%
Humboldt	305	-7.29%	1.45%	95.35%
White Pine	153	-2.54%	0.73%	96.08%
Pershing	236	-15.4%	1.13%	97.21%
Lander	119	-6.29%	0.57%	97.78%
Mineral	111	-3.47%	0.53%	98.31%
Storey	28	3.7%	0.13%	98.44%
Lincoln	119	7.2%	0.57%	99.01%
Eureka	122	3.39%	0.58%	99.59
Esmeralda	87	-4.39%	0.41%	100
<b>Total</b>	21,021	0.89%	<b>100%</b>	-
<i>continued on next page</i>				

<i>continued from previous page</i>				
County	AVMT(millions)	Change from 2005-08	AVMT %	Cumulative %

### 3.3.3 Survey Design: Stage Two

#### 3.3.3.1 Number of roadway segments

Next, the number of roadway segments to be selected for the observations had to be decided. The average number of road segments for the two counties is 1,095. The quantitative guidelines for selecting the number of roadway segments in a county have been provided in Table 3.4. Since the number of roadway segments are more than 1000, therefore a minimum of 29 roadway segments have to be sampled from each county, totalling 58 roadway segments.

Table 3.4: Deciding the number of roadway segments from each county (Source: Sample survey design in Uniform Criteria 23 CRF, 1998)

Average road segments per county	Sampled roadway segments per county
50	19
60	20
70-80	21
90	22
100	23
200	26
<i>continued on next page</i>	

<i>continued from previous page</i>	
Average road segments per county	Sampled roadway segments per county
300-400	27
500-900	28
<b>More than 1000</b>	<b>29</b>

### 3.3.3.2 Road type stratification

The next step in the design was to distribute the observation sites based on road type stratification. For 2010, the 2008 Annual Vehicle Miles Travelled (AVMT) data obtained from Nevada DOT has been used. The Nevada DOT under U.S. Department of Transportation Federal Highway Administration (2008) publishes AVMT information by county based on functional classifications for rural and urban areas. This classification is shown in Table 3.5

Table 3.5: Roadway Stratification based on Annual Vehicle Miles Travelled

Roadway type	Urban area	Rural area
High Volume	All Principal Arterials	All Principal Arterials
Medium Volume	Collectors	Minor Arterials and Major Collectors
Low Volume	Local roads	Minor Collectors and Local road

Also, functional classification maps (2004) are provided by Nevada DOT for roadway segments falling in rural and urban areas for Clark and Washoe counties. This can be seen in Figure 3.1 and Figure 3.2.

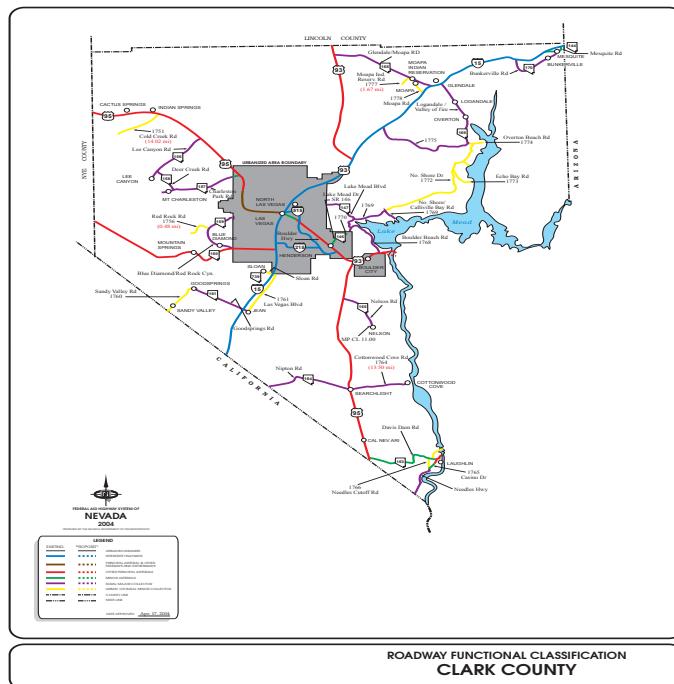


Figure 3.1: Roadway Functional Classification - Clark County

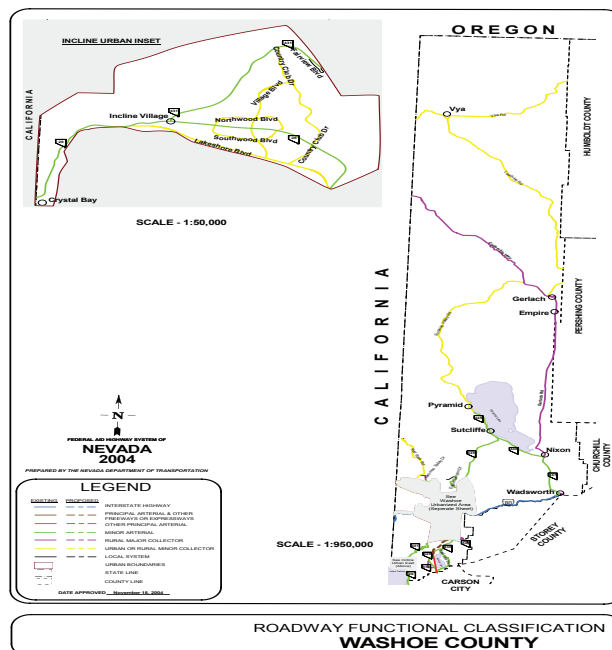


Figure 3.2: Roadway Functional Classification - Washoe County

Thus, the different roadway segments were divided into six strata according to Volume: Urban High, Urban Medium, Urban Low, Rural High, Rural Medium and Rural Low.

### 3.3.3.3 Changes due to 2008 AVMT information

According to the 2008 AVMT data, the stratification of 58 roadway segments into the aforementioned six strata design yielded approximately 0 sites in rural medium volume and rural low volume categories for Clark County. Table 3.6 provides a comparison of site allocation based on 2005 and 2008 data to the 6 strata for the Clark county. Table 3.7 provides a similar comparison for the Washoe county.

Table 3.6: Comparison of AVMT based site allocation - Clark County('m' represents millions, Source: Nevada DOT, 2008)

Strata	2005 AVMT(m)	Sites	2008 AVMT(m)	Sites
Rural High Volume	1300	3	1219	2.56
Rural Medium Volume	252	1	196	<b>0.41</b>
Rural Low Volume	309	1	149	<b>0.316</b>
Total Rural	1861	5	1565	3.29
Urban High Volume	4789	10	5588	11.76
Urban Medium Volume	4343	9	4439	9.34
Urban Low Volume	2367	5	2185	4.59
Total Urban	11499	24	12214	25.7
<i>continued on next page</i>				

<i>continued from previous page</i>				
Strata	2005 AVMT(m)	Sites	2008 AVMT(m)	Sites

Table 3.7: Comparison of AVMT based site allocation - Washoe County('m' represents millions, Source: Nevada DOT, 2008)

Strata	2005 AVMT(m)	Sites	2008 AVMT(m)	Sites
Rural High Volume	240	2	211	1.88
Rural Medium Volume	226	2	226	2.02
Rural Low Volume	142	1	129	1.15
Total Rural	608	5	567	5.06
Urban High Volume	1745	15	1759	15.69
Urban Medium Volume	805	7	721	6.43
Urban Low Volume	190	2	204	1.82
Total Urban	2740	24	2684	23.94

#### 3.3.3.4 Adding more sites: AVMT and Rounding up

As evident in Table 3.6, it was required that more sites be added since the division of 58 roadway segments into the aforementioned six strata leads to approximately 0 medium volume and low volume sites in the rural areas of Clark county. Hence, the number of sites obtained from 2008 AVMT information(last column named 'Sites' in Table 3.6 and Table 3.7) were rounded up. Thus, the final site distribution based on road stratification is shown in Table 3.8

Table 3.8: Distribution of sites



Strata	Clark county (Change)	Washoe county(Change)
Rural HV	3	2
Rural MV	1	<b>3(+1)</b>
Rural LV	1	<b>2(+1)</b>
<b>Total Rural sites</b>	5	<b>7(+2)</b>
<b>Urban HV</b>	<b>12 (+2)</b>	<b>16(+1)</b>
<b>Urban MV</b>	<b>10 (+1)</b>	7
Urban LV	5	2
<b>Total Urban sites</b>	<b>27 (+3)</b>	<b>25(+1)</b>

#### 3.3.3.5 Selection of new sites

For adding 6 more sites, the functional classification maps shown in Figure 3.1 and Figure 3.2 were used. With the help of Table 3.5, the roadway categories were divided based upon area and volume information. A road was randomly chosen for the required category. Then, the Average Annual Daily Traffic(AADT) count information obtained from Nevada DOT (2008) was used to select the site with the highest AADT count information corresponding to the chosen road. This process is shown in Figure 3.3. The newly added sites are shown in Table 3.9. An overview of a few sample locations in Clark county and Washoe county is shown in Figure 3.4. The total of 64 sites for the 2010 survey are available online and can be downloaded by clicking 'Download Full Report for 2010' button under NUTC website (Lakhanpal, 2010).

Table 3.9: New Added Sites

County	Volume	Location	(Comments)
Clark	Urban High	Tropicana at Paradise	WB
		US-95 at Decatur	SB off ramp
	Urban Medium	Fort Apache at Charleston	NB
Washoe	Rural Medium	EastLake Blvd, South of US-395	NB
	Rural Low	Pyramid and Sutcliffe	Both
	Urban High	US-395 at Oddie Blvd	NB off ramp

Figure 3.3: Selecting an Urban High Volume site in Clark county

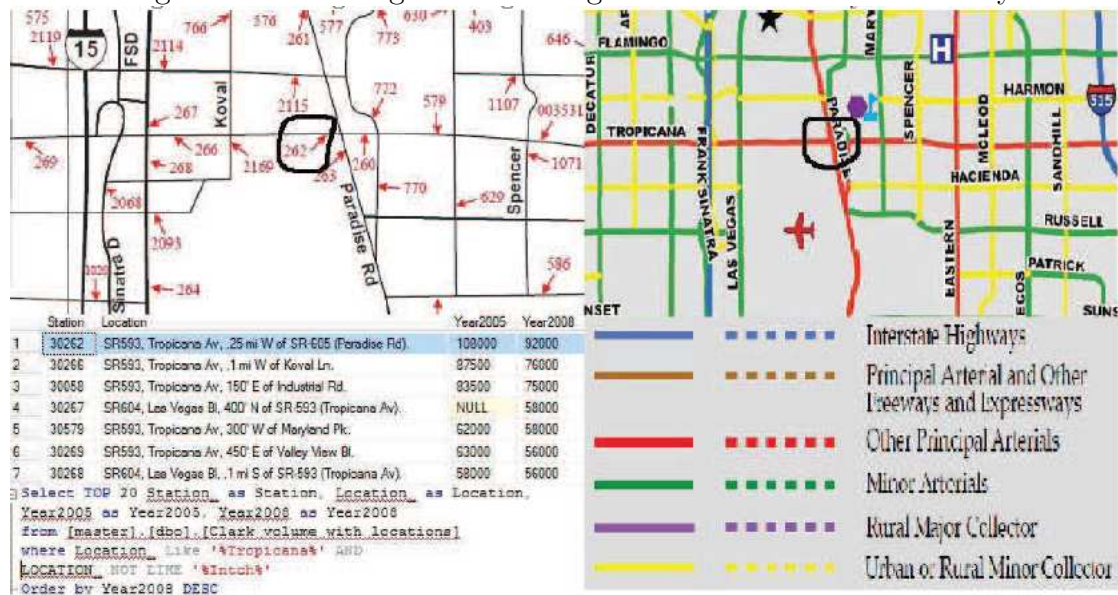
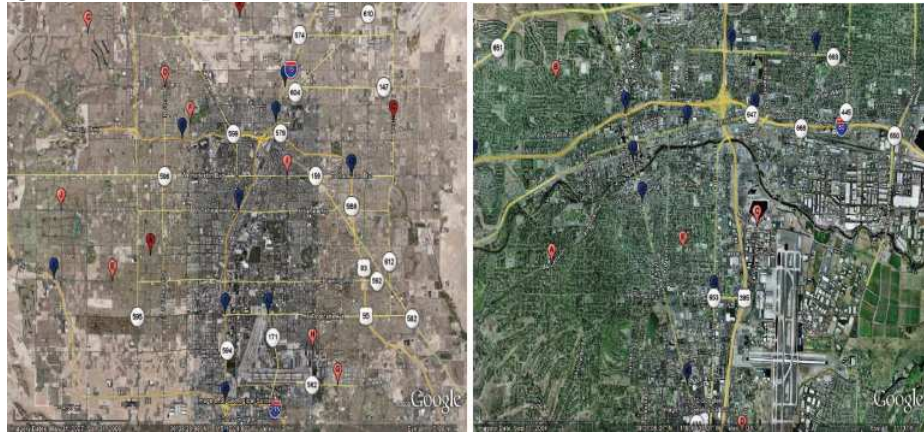


Figure 3.4: Sample observation locations for Clark and Washoe County



#### 3.3.3.6 Day of Week and Time of Day

Six 90-minute blocks of daylight time were identified for observations as follows:

- 7:30 AM - 9:00AM
- 9:00 AM - 10:30AM
- 10:30AM - 12 noon
- 1:00 PM - 2:30 PM
- 2:30 PM - 4:00 PM
- 4:00 PM - 5:30 PM

One observation time period was 40 minutes within any of the aforementioned time blocks. To minimize the travel time and the distance required to conduct the surveys, observation sites were grouped into geographic clusters. A day of the week to begin data collection was assigned to a cluster (using the Random Function in the software program Microsoft Excel). All days of the week (including Saturday and Sunday) were

considered eligible for selection. Within a cluster, each road segment was randomly assigned to the available time slots and a direction of travel to be observed was randomly selected. The supervisor assigned observers to these pre-selected sites.

### 3.3.3.7 Sample Size

Based on the guidelines in Uniform Criteria (1998), rough guidelines for determining sample size were to be provided. Since the proposed observations are time-based, it was difficult to determine exact sample size. However, based on statistical methods, rough estimate could be identified. For a 95 percent confidence level and 1 percent tolerable error, the minimum required number of observations could be estimated, assuming a binomial distribution for the data. In order to properly estimate the needed number of observations, either a usage rate from an earlier study or an estimate of the usage rate is needed. Although previous years' observation show a safety belt usage rate of around 90 percent, since the proposed sites were different from the existing ones, the TRC elected to use a 50 percent usage rate for this calculation instead of generating estimates based on the results of the previous studies. This usage rate was considered a starting point for the calculations since it generated the highest number of needed observations. Using the 50 percent usage rate as a starting point also yields a conservative estimate of the sample size required. A binomial distribution was assumed for the data. Thus, the corresponding equation used to determine the number of observations is shown in Equation 3.1.

$$n = \frac{z^2 P(1 - P)}{e^2} \quad (3.1)$$

where,  $n$  represents the needed number of observations,  $z$  represents the standard normal deviate appropriate for the desired confidence level,  $P$  is initial or anticipated estimated usage rate, and  $e$  represents the tolerable error.

The minimum number of needed observations for the state were calculated to be 9604 for a 95 % confidence level and a 1% tolerable error. Rounding off the minimum number, an estimated 10,000 observations will be acquired from the 64 sites for a single survey. Similar studies by the states of Utah and Wyoming estimated their sample size to be 15,000 observations from 162 sites and 16,500 from 207 sites respectively.

#### 3.3.3.8 Data Collection for Estimation

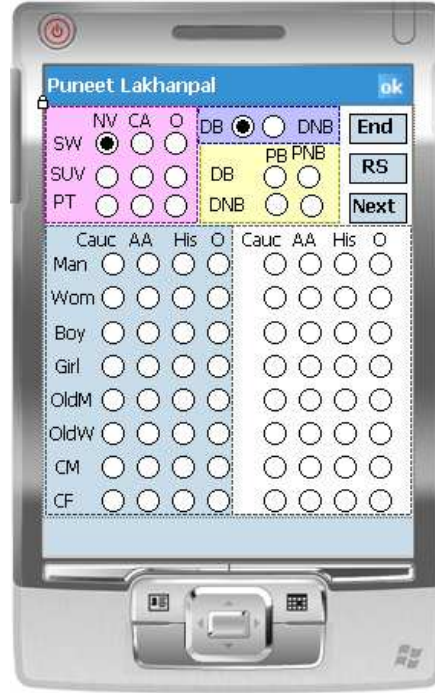
Observers recorded safety belt usage for front seat occupants of each vehicle observed i.e., the drivers and outboard passengers. The following passenger vehicles were observed: Sedans/station wagons, Vans/Sport utility vehicles(SUVs), and pickup trucks. These variables were used to estimate the statewide usage rate.

#### 3.3.3.9 Data Collection Software

In 2010, Transportation Research Center at UNLV developed a new data collection software which replaced the old software templates being used since 2007. The reasons for the replacement and the technical details of the new software will be shown in Chapter 4. This software was built from all open source softwares (totally free): Visual Studio 2008, Software development Kits (Pocket PC 2003, Windows Mobile 5, Windows Mobile 6) and Ultralite database from SQL anywhere (extremely fast and low memory footprint on PDA). The software is shown in Figure 4.5.

Figure 4.5 shows the data collection template for collecting the data on the field.

Figure 3.5: Data collection software developed at TRC in 2010



The variables (Gender and Age) have been combined to give eight variables: Man(20-60 years of age), Woman(Wom, 20-60 years of age), Boy(15-19 years of age), Girl(15-19 years of age), Elderly Man(OldM, >60 years of age), Elderly Woman(OldW, >60 years of age), Younger Boy(CM, <15 years of age) and Younger Girl(CF, <15 years of age). In this data collection template, the state of Nevada is chosen by default since 89.5% of the vehicles noted in 2009 study were registered in Nevada. Similarly, sedans/station wagons have been selected by default. Of course, these defaults can be changed as per the observation. The blue half of data collection template, as shown in Figure 4.5, is for collecting data related to the driver. The white half of the template is used for collecting data according to the observed passenger. The checkboxes B and NB represent 'Belted' and 'Not Belted' respectively. Although this template provides more visual information, the input required from the user is optimally low, keeping the

size constraints in mind. In worst case scenario (Driver, Passenger, State other than NV, Vehicle other than Sedan/Station Wagon and Next), only 4 clicks are required. On the other hand, in best case scenario (only Driver, NV state, Sedan/Station Wagon and Next), only 1 click is required. This is a huge improvement over the previous templates, the details of which are mentioned in Chapter 4. The observers were well trained before the actual data collection on this software. Survey forms in paper were also printed as a backup.

#### 3.3.3.10 Observation Protocols

The exact observation sites, including specific road segments, time of day, day of week and direction of vehicle travel, were determined prior to observers conducting the survey. The observer was not authorized to make any changes to these pre-selected aspects of the survey unless authorized by a supervisor. Observed motor vehicle occupants (either driver or front seat passenger) wearing shoulder belts were only considered as belted occupants, others wearing lap belts or wearing no safety belts were not considered as 'belted'. What follows is a discussion of the methodological protocols for the observations used for this study:

- **The order of observation:** Within the clusters, the order of observation was assigned with the use of a random numbering procedure. For sites outside the clusters, the order was determined by proximity to clustered sites.
- **Traffic direction:** In those cases where the roadway moves in only one direction, no real choice was involved. If a site was situated proximate to a county line, the traffic direction toward the county was associated with the survey. In

all other instances involving a decision of direction, a randomization process was employed.

- **Vehicles observed:** All passenger vehicles were observed and classified on the observation form as sedans/station wagons, vans, sport utility vehicles or pickup trucks.
- **Occupants observed:** The drivers and outboard passengers in the passenger vehicles selected were observed for safety belt usage. The gender of these occupants was recorded along with the safety belt usage information. Children were included in the survey. Any occupant who appeared to be younger than 15 years of age was considered as a Younger Boy. We recognize that this is a subjective determination. Observers were provided training to help make consistent judgment in this regard. Children in child safety seats were NOT considered in the observations. However, if a child was seated in a booster seat, and if he/she was sitting on the front occupant seat (which is very rare), his/her safety belt use was observed and recorded.
- **Traffic conditions and data collection problems** Observers were trained to cope with traffic problems. When traffic is heavy and there were too many vehicles to count visually, counting was done as long as possible and then stopped until the observers count could catch up with the observations. Some vehicles, out of necessity, were skipped under these circumstances. When this occurred, counting resumed after no more than a one-minute pause. Once an observers eyes were locked on a vehicle, a count of that vehicle was entered on the obser-



vation form.

- At sites with more than one lane of traffic in the predetermined direction, observations were made from the lane closest to the observer.
- Field observers were allowed to terminate a pre-selected set of observations if any of the following circumstances arose: 1) extreme weather conditions that would hinder the accuracy of the observations; 2) traffic flow that is heavy enough to endanger the safety of the observer; 3) traffic crashes, traffic conditions, or road construction that would render the observations unfeasible, especially when a detour is involved. If observations at a pre-selected site were to be terminated, the observer was to note the reason and mark the time of termination on the form. The observer was instructed to notify the supervisor about the termination as soon as possible.
- **Site accessibility problems:** If a pre-selected site was not available on the survey date or time, the observer made the following modifications:
  - On mile-posted roads, observations were to be made at a location with a mile point that was one mile higher on the same roadway in the same direction as the assigned traffic flow. If this point was not accessible, more miles were added in one-mile increments, up to three miles. Such changes were noted on the observation form.
  - On non-mile point streets and local roadways, the observer was to proceed in the same direction as the assigned traffic flow in one-quarter mile

increments, not to exceed three-quarters of a mile, until an appropriate observation site was found and so noted on the observation form.

- In cases of road construction or other road obstruction where traffic was detoured, the observer was required to select a site on the detour as close to the original site as possible, no more than two miles away on mile-pointed roadways and no more than one-half mile on non-mile-pointed roadways. The change in site location and the reason for the change was be noted on the observation form.

- **Observations** Safety belt usage and gender characteristics were recorded for drivers and outboard passengers in the front seat in the four identified vehicle types. In addition to observing and documenting this data, observers recorded other data from which additional information could be acquired. Driver and passenger gender were noted to determine usage rates by gender. In-state and out-of-state registered vehicles were noted to identify the usage rate of Nevada registered vehicle occupants vs. those from out-of-state. Observations occurred from the observers vehicle whenever possible. If an observer was unable to observe from his vehicle, s/he was allowed to stand off the roadway, and was required to wear a safety vest for visibility.

#### 3.3.4 Basic data analysis

This section provides a brief overview of the data analysis performed on both the Pre-Mobilization and the Post-Mobilization safety belt usage data. A total of 11,804 vehicles were observed during the Pre-Mobilization and 11,337 vehicles during

the Post-Mobilization survey process. The un-weighted safety belt usage rate for front seat occupants was 90.54% for the Pre-Mobilization survey (14,741 front seat occupants) and 93.63% for the Post-Mobilization survey (14,387 front seat occupants). The data was throughly analyzed and a few tables were generated such as: State Wide Seat Belt Usage in Numbers and Percentage during Pre and Post-Mobilization survey, Seat Belt Usage by drivers and passengers categorized over Age and Gender during Pre and Post-Mobilization survey etc. In this section, only one table each for Pre and Post-Mobilization surveys will be demonstrated to provide a peek into the data analysis. The tables demonstrating the safety belt usage by front seat occupants during Pre and Post-Mobilization surveys are shown in Tables 3.10 and 3.11 respectively. The details of the project are available online (Lakhanpal, 2010).

Table 3.10: Seat Belt Usage by Front Seat Occupants during Pre-Mobilization survey (in %)

Site	20-60		15-19		>60		<15	
	Male	Female	Male	Female	Male	Female	Male	Female
1	97.37	95.45	100.00	(0/0)	86.21	94.12	100.00	(0/0)
2	95.77	98.59	100.00	(0/0)	96.97	100.00	100.00	(0/0)
3	89.17	92.86	(0/0)	0.00	76.47	100.00	100.00	(0/0)
4	88.76	100.00	100.00	100.00	100.00	100.00	100.00	(0/0)
5	89.47	88.89	100.00	100.00	91.67	100.00	100.00	100.00
<i>continued on next page</i>								

<i>continued from previous page</i>								
Site	20-60		15-19		>60		<15	
	Male	Female	Male	Female	Male	Female	Male	Female
6	92.65	97.78	(0/0)	(0/0)	95.83	100.00	100.00	100.00
7	92.62	96.30	100.00	100.00	100.00	100.00	(0/0)	(0/0)
8	76.92	84.93	(0/0)	100.00	100.00	100.00	(0/0)	100.00
9	81.90	89.83	100.00	50.00	93.33	100.00	100.00	(0/0)
10	91.49	95.19	0.00	100.00	92.59	95.45	(0/0)	(0/0)
11	87.64	91.67	100.00	(0/0)	100.00	100.00	(0/0)	(0/0)
12	91.78	92.00	85.71	100.00	85.71	100.00	(0/0)	0.00
13	94.77	97.41	100.00	(0/0)	83.78	95.45	100.00	50.00
14	93.37	87.78	100.00	100.00	100.00	100.00	100.00	100.00
15	80.23	88.37	(0/0)	(0/0)	75.00	100.00	100.00	(0/0)
16	84.15	88.00	(0/0)	50.00	85.71	100.00	(0/0)	100.00
17	97.16	96.30	100.00	100.00	80.00	100.00	100.00	(0/0)
18	84.47	96.00	(0/0)	(0/0)	94.12	90.00	(0/0)	(0/0)
19	85.29	91.95	(0/0)	(0/0)	100.00	100.00	100.00	(0/0)
20	85.82	94.70	50.00	100.00	96.55	97.37	(0/0)	100.00
21	87.50	90.43	(0/0)	100.00	94.44	100.00	100.00	100.00
22	89.72	90.80	(0/0)	100.00	100.00	94.44	50.00	100.00
23	90.08	93.16	(0/0)	(0/0)	100.00	95.24	100.00	100.00
24	94.66	90.59	(0/0)	(0/0)	80.65	100.00	(0/0)	100.00
<i>continued on next page</i>								

<i>continued from previous page</i>								
Site	20-60		15-19		>60		<15	
	Male	Female	Male	Female	Male	Female	Male	Female
25	92.02	97.78	(0/0)	100.00	92.00	100.00	100.00	100.00
26	88.98	98.15	100.00	100.00	80.00	100.00	(0/0)	(0/0)
27	95.93	86.96	100.00	(0/0)	90.48	87.50	100.00	100.00
28	89.33	95.24	(0/0)	0.00	100.00	100.00	100.00	100.00
29	81.58	96.58	100.00	100.00	96.15	96.97	83.33	100.00
30	83.33	92.50	100.00	66.67	66.67	100.00	100.00	(0/0)
31	72.97	76.92	(0/0)	(0/0)	81.82	78.26	100.00	100.00
32	85.71	85.71	50.00	100.00	83.33	100.00	66.67	(0/0)
33	91.07	98.65	100.00	100.00	95.65	100.00	(0/0)	(0/0)
34	95.52	82.50	100.00	(0/0)	92.00	100.00	100.00	(0/0)
35	90.14	96.97	(0/0)	(0/0)	100.00	100.00	66.67	(0/0)
36	91.73	97.44	100.00	100.00	100.00	100.00	100.00	(0/0)
37	77.42	92.00	(0/0)	100.00	93.33	100.00	100.00	(0/0)
38	91.89	93.75	(0/0)	(0/0)	100.00	100.00	(0/0)	(0/0)
39	84.44	75.00	100.00	100.00	83.33	80.00	(0/0)	(0/0)
40	88.04	94.39	100.00	77.78	97.22	84.00	100.00	(0/0)
41	82.20	86.05	70.59	75.00	85.71	82.61	100.00	(0/0)
42	80.00	83.58	(0/0)	100.00	87.50	86.67	100.00	(0/0)
43	79.03	83.33	75.00	25.00	80.00	100.00	100.00	0.00
<i>continued on next page</i>								

<i>continued from previous page</i>								
Site	20-60		15-19		>60		<15	
	Male	Female	Male	Female	Male	Female	Male	Female
44	84.31	89.74	(0/0)	100.00	93.75	92.31	(0/0)	(0/0)
45	91.55	93.81	0.00	100.00	93.75	100.00	100.00	(0/0)
46	97.87	97.94	0.00	100.00	90.00	100.00	(0/0)	(0/0)
47	87.60	89.89	100.00	92.86	92.86	94.12	100.00	(0/0)
48	88.55	95.18	83.33	80.00	93.33	90.24	(0/0)	(0/0)
49	88.99	90.63	100.00	100.00	94.34	87.72	(0/0)	(0/0)
50	88.73	92.45	33.33	100.00	92.86	100.00	0.00	100.00
51	90.00	90.32	100.00	100.00	100.00	100.00	100.00	(0/0)
52	93.85	95.56	100.00	100.00	85.71	85.71	100.00	(0/0)
53	84.96	91.18	100.00	100.00	84.62	93.33	100.00	(0/0)
54	92.92	89.47	100.00	(0/0)	92.86	93.33	100.00	(0/0)
55	85.71	91.07	(0/0)	(0/0)	87.50	90.91	100.00	(0/0)
56	86.80	97.48	100.00	66.67	92.31	100.00	(0/0)	(0/0)
57	91.75	98.25	100.00	100.00	91.30	87.50	50.00	(0/0)
58	89.58	92.41	0.00	57.14	93.33	100.00	100.00	(0/0)
59	88.46	94.29	100.00	71.43	90.32	100.00	100.00	100.00
60	85.29	95.83	100.00	100.00	81.82	100.00	(0/0)	(0/0)
61	84.21	88.50	100.00	90.00	90.00	90.63	100.00	100.00
62	94.81	98.67	100.00	100.00	90.91	100.00	(0/0)	(0/0)
<i>continued on next page</i>								

<i>continued from previous page</i>								
Site	20-60		15-19		>60		<15	
	Male	Female	Male	Female	Male	Female	Male	Female
63	98.11	95.74	100.00	(0/0)	100.00	100.00	(0/0)	(0/0)
64	84.75	86.84	33.33	66.67	90.00	92.31	100.00	100.00
Overall	88.71	92.58	88.89	82.98	91.17	94.53	93.02	92.30

Table 3.11: Seat Belt Usage by Front Seat Occupants during Post-Mobilization survey

Site	<15		15-19		20-60		>60	
	Male	Female	Male	Female	Male	Female	Male	Female
1	(0/0)	(0/0)	100.00	100.00	93.33	100.00	100.00	100.00
2	(0/0)	(0/0)	100.00	100.00	97.19	100.00	100.00	100.00
3	50.00	(0/0)	100.00	100.00	94.82	94.74	100.00	100.00
4	(0/0)	(0/0)	100.00	(0/0)	96.33	97.14	100.00	(0/0)
5	100.00	(0/0)	100.00	100.00	91.53	97.87	81.25	100.00
6	(0/0)	(0/0)	100.00	(0/0)	92.11	90.00	100.00	100.00
7	(0/0)	100.00	100.00	100.00	92.82	95.20	88.89	100.00
8	100.00	100.00	100.00	100.00	85.92	88.42	100.00	100.00
9	100.00	100.00	100.00	75.00	87.58	89.53	100.00	100.00
10	(0/0)	(0/0)	(0/0)	(0/0)	91.34	100.00	94.44	100.00
<i>continued on next page</i>								

<i>continued from previous page</i>								
Site	<15		15-19		20-60		>60	
	Male	Female	Male	Female	Male	Female	Male	Female
11	100.00	(0/0)	80.00	100.00	95.12	96.30	100.00	100.00
12	100.00	(0/0)	100.00	100.00	93.68	94.87	60.00	83.33
13	100.00	(0/0)	100.00	(0/0)	96.52	98.94	94.74	100.00
14	(0/0)	(0/0)	100.00	100.00	94.35	94.17	85.71	100.00
15	100.00	100.00	100.00	(0/0)	92.55	100.00	100.00	100.00
16	100.00	(0/0)	85.71	50.00	90.34	90.43	95.00	100.00
17	100.00	100.00	60.00	75.00	89.29	92.75	87.50	100.00
18	(0/0)	(0/0)	(0/0)	(0/0)	92.31	90.91	100.00	100.00
19	50.00	0.00	100.00	50.00	88.37	97.53	80.00	100.00
20	100.00	(0/0)	100.00	(0/0)	91.95	96.33	89.47	95.83
21	(0/0)	(0/0)	100.00	(0/0)	88.68	100.00	85.71	100.00
22	(0/0)	(0/0)	100.00	100.00	91.67	100.00	100.00	100.00
23	100.00	(0/0)	(0/0)	(0/0)	90.83	94.51	100.00	100.00
24	(0/0)	(0/0)	100.00	100.00	94.08	98.77	91.67	100.00
25	(0/0)	(0/0)	100.00	(0/0)	93.40	97.30	94.74	100.00
26	100.00	(0/0)	0.00	75.00	87.43	93.68	75.00	100.00
27	100.00	100.00	100.00	(0/0)	93.52	97.30	100.00	94.12
28	100.00	(0/0)	(0/0)	(0/0)	95.92	96.74	87.50	94.44
29	(0/0)	(0/0)	(0/0)	100.00	98.48	97.96	100.00	100.00
<i>continued on next page</i>								

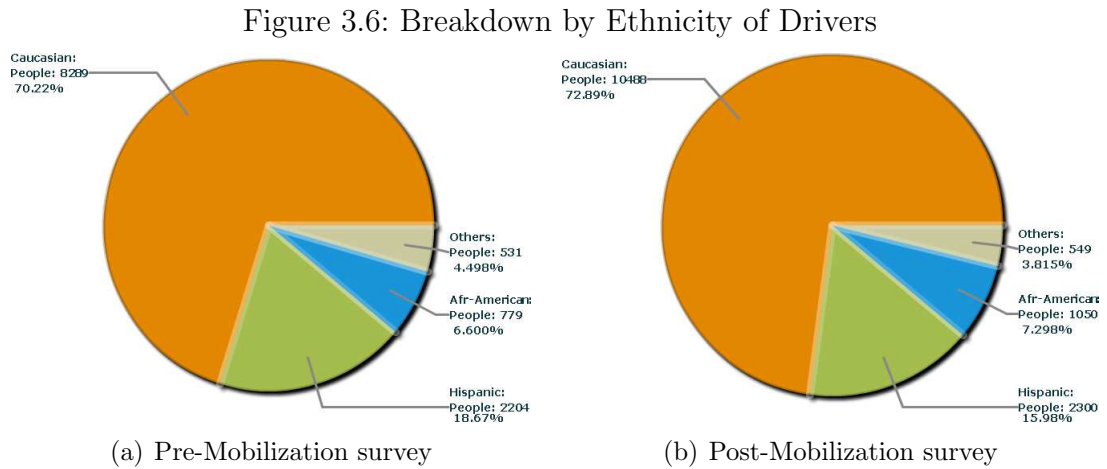


<i>continued from previous page</i>								
Site	<15		15-19		20-60		>60	
	Male	Female	Male	Female	Male	Female	Male	Female
30	(0/0)	100.00	(0/0)	100.00	89.74	90.16	100.00	(0/0)
31	(0/0)	(0/0)	0.00	(0/0)	89.61	93.33	94.74	100.00
32	(0/0)	50.00	100.00	75.00	80.92	83.33	100.00	100.00
33	(0/0)	(0/0)	100.00	100.00	90.00	95.12	93.33	100.00
34	100.00	(0/0)	100.00	(0/0)	95.45	95.24	87.50	92.86
35	(0/0)	(0/0)	100.00	100.00	98.66	97.92	98.46	98.04
36	100.00	(0/0)	100.00	(0/0)	94.48	100.00	100.00	100.00
37	(0/0)	(0/0)	(0/0)	100.00	81.40	84.62	89.19	100.00
38	(0/0)	(0/0)	(0/0)	(0/0)	83.33	100.00	90.00	100.00
39	(0/0)	(0/0)	100.00	(0/0)	73.68	100.00	60.00	100.00
40	100.00	100.00	87.50	84.62	90.28	93.75	100.00	96.00
41	(0/0)	100.00	88.89	72.73	88.03	93.59	100.00	100.00
42	100.00	100.00	71.43	80.00	88.42	94.62	100.00	100.00
43	66.67	100.00	0.00	100.00	94.12	97.18	75.00	100.00
44	(0/0)	50.00	100.00	85.71	86.39	93.68	75.00	94.74
45	50.00	(0/0)	(0/0)	100.00	96.88	92.45	100.00	100.00
46	100.00	(0/0)	100.00	(0/0)	97.30	97.53	100.00	100.00
47	100.00	(0/0)	85.71	100.00	88.14	91.47	100.00	100.00
48	100.00	(0/0)	100.00	100.00	90.84	96.15	93.75	100.00
<i>continued on next page</i>								

<i>continued from previous page</i>								
Site	<15		15-19		20-60		>60	
	Male	Female	Male	Female	Male	Female	Male	Female
49	(0/0)	(0/0)	60.00	91.67	94.02	94.53	100.00	100.00
50	100.00	(0/0)	93.33	94.12	86.89	93.37	100.00	100.00
51	100.00	(0/0)	100.00	100.00	96.63	97.80	100.00	100.00
52	(0/0)	(0/0)	100.00	(0/0)	97.47	98.51	100.00	100.00
53	(0/0)	(0/0)	(0/0)	100.00	93.48	100.00	94.12	100.00
54	(0/0)	(0/0)	(0/0)	100.00	92.16	98.67	100.00	100.00
55	(0/0)	(0/0)	80.00	100.00	92.59	98.00	97.30	100.00
56	100.00	(0/0)	(0/0)	85.71	93.96	93.55	100.00	100.00
57	(0/0)	(0/0)	100.00	(0/0)	90.91	96.30	100.00	100.00
58	100.00	100.00	100.00	50.00	87.62	87.78	94.12	100.00
59	100.00	100.00	100.00	77.78	95.00	96.23	100.00	95.83
60	(0/0)	(0/0)	100.00	(0/0)	91.43	93.22	100.00	93.75
61	(0/0)	(0/0)	50.00	100.00	96.84	87.27	(0/0)	100.00
62	(0/0)	(0/0)	100.00	(0/0)	94.92	96.58	97.30	100.00
63	(0/0)	(0/0)	100.00	100.00	96.34	95.18	90.91	100.00
64	100.00	100.00	90.91	100.00	94.00	93.10	94.12	100.00
Overall	92.86	90.91	89.20	89.01	92.12	95.01	95.48	98.80

The data was also analyzed based upon the state of registration (Nevada, California and Others), vehicle types(Sedan/Station Wagon, Van/SUV, Pickup trucks)

and ethnicity (Caucasian, African-American, Hispanic and Others). The pie chart created for analyzing the data categorized over ethnicities is shown in Figure 3.6.

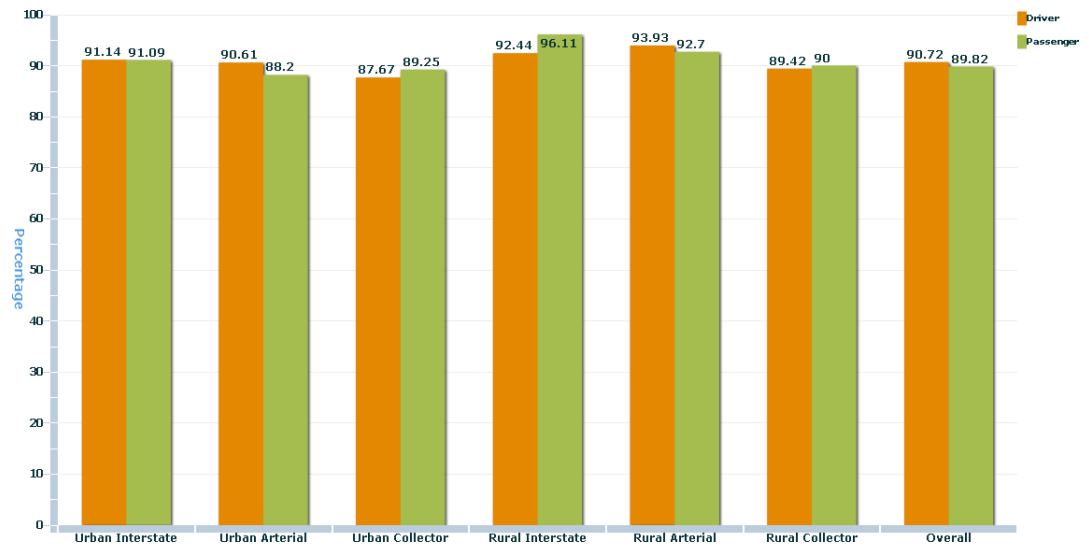


Last but not the least, the safety belt usage data was also analyzed based upon the functional class of streets (Urban Interstate, Urban Arterial, Urban Collector, Rural Interstate, Rural Arterial and Rural Collector) and the area type (Urban, Rural). The safety belt usage for drivers and passengers for different functional classes of streets during the Pre and Post-Mobilization periods is shown in Figure 3.7. Please note that the Figures 3.6 and 3.7 were generated by the Pie Chart and Column Chart components available in Adobe Flex 3, which are further discussed in Chapter 7.

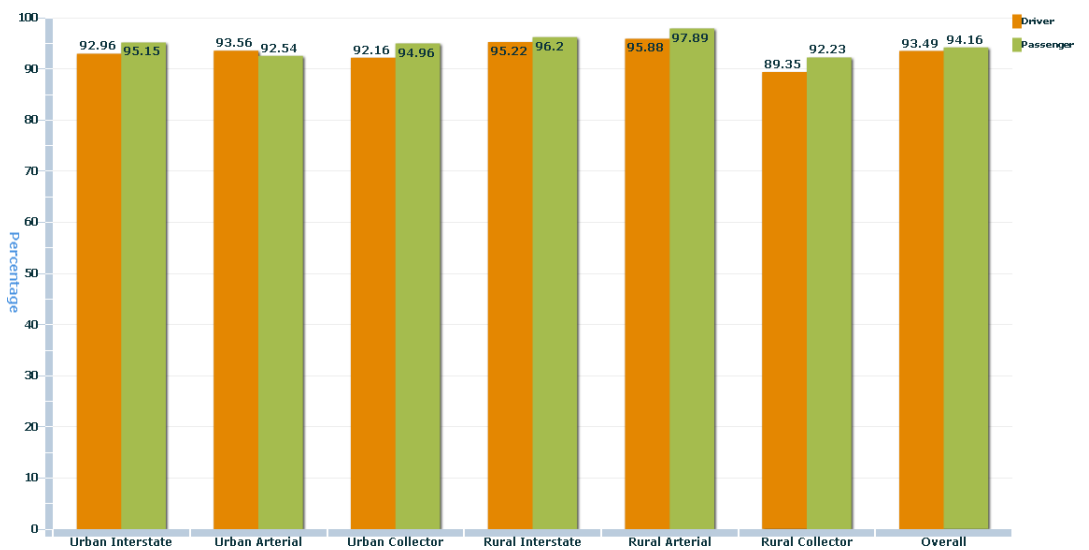
### 3.3.5 Calculating the Weighted Data

The analysis of the safety belt usage data in Nevada has taken the form of aggregate calculations of overall county and state-weighted estimates using a spreadsheet design that incorporates the mathematical formulae. This is done in a three-step calculation process. The first step calculates the safety belt usage rates for the sites

Figure 3.7: Seat Belt Use for Drivers and Passengers for Different Functional Classes of Streets



(a) Pre-Mobilization survey



(b) Post-Mobilization survey

within each county. Those estimates are then used to derive the estimates for each of the counties. Finally, the county estimates are used to derive the overall estimate of safety belt usage for the state as a whole. Based on the research teams communication with the sponsors in 2007, it was decided to restructure the strata of sites to two categories, Rural and Urban, for each county from the six that was used initially before 2007. This was to enable estimating standard error based on bootstrap method. Because the observation sites are selected with a probability proportional to the estimates of vehicle miles traveled within each site, the formula for estimating the safety belt usage rates for the sample sites is given in Equation 3.2.

$$\hat{y}_{ij} = \frac{\sum_{k=1}^{n_{ij}} W_{ijk} VMT_{ijk} B_{ijk} / O_{ijk}}{\sum_{k=1}^{n_{ij}} W_{ijk} VMT_{ijk}} \quad (3.2)$$

where,  $\hat{y}_{ij}$  represents the estimate of safety belt use for the  $j^{th}$  stratum in the  $i^{th}$  county,  $i$  represents county ranging from 1 to number of counties in sample (2),  $j$  represents stratum of road segments ranging from 1 to 2,  $k$  represents the designated sample site ranging from 1 to  $n_{ij}$ ,  $W_{ijk}$  represents the sampling weight of each site within each stratum of each county,  $B_{ijk}$  represents the total number of belted drivers and passengers for the sample site in the stratum and  $O_{ijk}$  represents the total number of observed drivers and passengers for the sample site in the stratum.

For each  $1 \leq i \leq 2$ ,  $1 \leq j \leq 2$  and  $1 \leq k \leq n_{ij}$ ,  $W_{ijk}$  can be computed as given in Equation 3.3.

$$W_{ijk} = \frac{\sum_{l=1}^{n_{ij}} VMT_{ijl(2005)}}{n_{ij} VMT_{ijk(2005)}} \quad (3.3)$$

where,  $VMT_{ijk(2005)}$  represents daily vehicle miles traveled in the base year (i.e. 2005) for the road segment on which site  $k$  of stratum  $j$  in county  $i$  lies. Equation 3.3 calculates the sampling weight of site  $k$  in stratum  $j$  of county  $i$ , i.e., the inverse of the inclusion probability of site  $k$  in stratum  $j$  of county  $i$ , and where  $n_{ij}$  is the number of sites selected from stratum  $j$  of county  $i$ .

The estimates  $\hat{y}_{ij}$  for the sites are then used to create the estimates for the counties. Note that  $B_{ijk}$  and  $O_{ijk}$  will include all of the data collected. Because the road segments are allocated to the urban and rural strata with a probability proportional to the number of road segments in each stratum, the formula for estimating safety belt use in each county is shown in Equation 3.4.

$$\hat{y}_i = \frac{\sum_{j=1}^2 VMT_{ij} \sum_{k=1}^{n_{ij}} W_{ijk} VMT_{ijk} B_{ijk} / O_{ijk}}{\sum_{j=1}^2 VMT_{ij} \sum_{k=1}^{n_{ij}} W_{ijk} VMT_{ijk}} \quad (3.4)$$

where,  $VMT_{ij}$  represents the combined VMT of all roads (both selected and not) in stratum  $j$  of county  $i$ .

Finally, the statewide estimate of safety belt use will be calculated according to the Equation 3.5.

$$\hat{y} = \frac{\sum_{i=1}^2 VMT_i \hat{y}_i}{\sum_{i=1}^2 VMT_i} \quad (3.5)$$

where,  $VMT_i$  represents the total daily vehicle miles traveled for all the road segments

(both selected and not) in county  $i$  and is calculated as shown in Equation 3.6.

$$VMT_i = \sum_{j=1}^2 VMT_{ij} \quad (3.6)$$

The whole collected data is processed in MYSQL database server and queried through scripts written in PHP programming language. PHPMyAdmin IDE is used to make the PHP queries to the MYSQL server which makes the data processing very easy and efficient. The queries produced the output in a comma separated value (CSV) format which contained the following fields: Sitename,  $B_{ijk}$ ,  $O_{ijk}$ . These fields were later concatenated with the weights  $W_{ijk}$  obtained according to the Equation 3.3. The concatenation of the weights and further processing to calculate the weighted safety belt usage results and the bootstrap results were done in 'R', a statistical software package. Note that the most recent VMT data (currently available for the year 2009) would be used in the equations above, except for  $VMT_{ijk(2005)}$  and  $VMT_{ijl(2005)}$  in calculating  $W_{ijk}$ , where the VMT for the year 2005 would be used. For the 6 new added sites, the AADT data for 2005 was consulted to obtain the VMT for those sites.

#### 3.3.5.1 Sampling Error

As discussed in the previous sections, data were collected from roads falling in 6 strata:

- Rural High Volume: 1, 2, 3, 33 and 34
- Rural Medium Volume: 35, 36 and 37

- Rural Low Volume: 5, 38 and 39
- Urban High Volume: 6-17 and 47-62
- Urban Medium Volume: 18-27 and 40-46
- Urban Low Volume: 28-32 and 63-64

Since the number of sites in some of the strata are very small, and since we must use non-parametric bootstrap involving sampling with replacement, we had to classify the 64 sites into the following four strata:

- Urban Clark (5 sites)
- Urban Washoe ( $5 + 2 = 7$  sites)
- Rural Clark ( $24+1 = 25$  sites)
- Rural Washoe ( $24 + 3 = 27$  sites)

The non-parametric bootstrap procedure used in this report is briefly described below:

1. Input data is read.
2. For each of the four strata, bootstrap sampling (sampling with replacement) was used to select the original number of sites; e.g., 5 sites were selected with replacement for Urban Clark stratum.
3. The statewide rate of safety belt usage was calculated using the formula for the stratified estimate.



4. Steps 1-3 are repeated 1000 times, which will yield 1000 values of combined  $\hat{y}_{ij}$  the estimated statewide rate of safety belt usage.
5. The standard deviation (sd) of the 1000  $\hat{p}$  values from Step 4 is calculated. This sd is the standard deviation of the statewide rate of safety belt usage.
6. An approximate 95% confidence interval of the statewide rate of safety belt usage can be calculated from the following formula:

$$\hat{p}_{combined} \pm 1.96 \times sd(\hat{p}_{combined})$$

A program in the language R was written for the bootstrap method outlined above. The output fields were calculated as shown in Equations 3.2, 3.3, 3.4 and 3.5.

The code was run 3 times, with 1000 bootstrap simulations in each run. The following results were obtained:

1. Standard deviation in First Run: 0.005423 (Pre-Mobilization) and 0.00495978 (Post-Mobilization)
2. Standard deviation in Second Run: 0.0054926 (Pre-Mobilization) and 0.004904294 (Post-Mobilization)
3. Standard deviation in Third Run: 0.005515 (Pre-Mobilization) and 0.004836132 (Post-Mobilization)

An approximate 95% confidence interval from the data for statewide rate of safety belt usage was found to be (89.361992%, 91.487808%) during the Pre-Mobilization

and (92.1888731%, 94.1331069%) during Post-Mobilization.

### 3.3.6 Statistical Analysis

#### 3.3.6.1 Unweighted Analysis

In order to reliably state that the safety belt usage by the total front seat occupants increased from Pre-Mobilization survey to Post-Mobilization survey, statistical analysis was done on the data for all the 64 locations. The input data is shown in table 3.12.

Table 3.12: Input Data: Percentage Belted Total Front Seat Occupants

Site	Pre-Mobilization % belted	Post-Mobilization % belted
1	93.5185185185185	97.5308641975309
2	96.969696969697	98.2014388489209
3	89.4941634241245	94.8640483383686
4	92.7536231884058	96.6216216216216
5	91.0958904109589	93.5251798561151
6	95.1672862453531	92.4050632911392
7	94.6666666666667	93.854748603352
8	81.1827956989247	87.6984126984127
9	86.1904761904762	88.8461538461538
10	92.5925925925926	94.3127962085308
<i>continued on next page</i>		

<i>continued from previous page</i>		
Site	Pre-Mobilization % belted	Post-Mobilization % belted
11	89.9328859060403	95.364238410596
12	91.3934426229508	92.6666666666667
13	94.3019943019943	97.2222222222222
14	91.9463087248322	94.3089430894309
15	83.3333333333333	95.8333333333333
16	84.8249027237354	90.6705539358601
17	96.652719665272	90.0826446280992
18	87.741935483871	93.3333333333333
19	89.4230769230769	90.7488986784141
20	91.6434540389972	94.1908713692946
21	90.3846153846154	92.8571428571429
22	90.8695652173913	95.7894736842105
23	92.1428571428571	93.9516129032258
24	92.0454545454546	95.6862745098039
25	94.2372881355932	95.3051643192488
26	91.4438502673797	89.1696750902527
27	91.1671924290221	95.7528957528958
28	92.5465838509317	95.6204379562044
29	91.3946587537092	98.7421383647799
<i>continued on next page</i>		

<i>continued from previous page</i>		
Site	Pre-Mobilization % belted	Post-Mobilization % belted
30	86.013986013986	90.4109589041096
31	76.4367816091954	91.4285714285714
32	85.6353591160221	82.0754716981132
33	94.6902654867257	92.3076923076923
34	91.6083916083916	93.4210526315789
35	94.0789473684211	98.3516483516484
36	95.3947368421053	97.6923076923077
37	89.3203883495146	86.7647058823529
38	93.2203389830508	90
39	83.5616438356164	83.7209302325581
40	91.0958904109589	92.4242424242424
41	82.5174825174825	90.295358649789
42	82.3754789272031	91.2442396313364
43	81.5436241610738	94
44	87.6777251184834	88.6666666666667
45	93.0817610062893	96.0199004975124
46	96.7078189300412	98.1481481481482
47	89.6797153024911	90.7356948228883
48	90.9967845659164	93.7037037037037
<i>continued on next page</i>		

<i>continued from previous page</i>		
Site	Pre-Mobilization % belted	Post-Mobilization % belted
49	90.5775075987842	94.0509915014164
50	89.3333333333333	90.7960199004975
51	91.1764705882353	97.9757085020243
52	94.3548387096774	98.3333333333333
53	87.9310344827586	96.1538461538462
54	92.3076923076923	95.7547169811321
55	88.2926829268293	95.4081632653061
56	91.3513513513514	94.1860465116279
57	93.0348258706468	95.1219512195122
58	89.8263027295285	88.695652173913
59	91.5451895043732	95.4415954415954
60	89.7435897435897	93.4131736526946
61	87.1313672922252	92.9032258064516
62	96.5714285714286	96.3087248322148
63	97.4576271186441	96.0199004975124
64	85.5072463768116	94.0425531914894

### **R-Code and Analysis Results :**

In this section we include the results of paired tests for comparing the Pre-Mobilization and Post-Mobilization safety belt usage data. We have done this comparison using paired tests on the following variables:

1. Difference in safety belt usage percentages

$$Diff = Post\_Mobilization\%belted - Pre\_Mobilization\%belted$$

2. Percent increase  $PC$  in the safety belt usage calculated as

$$PC = 100 \times \frac{Post\_Mobilization\%belted - Pre\_Mobilization\%belted}{Pre\_Mobilization\%belted}$$

The hypotheses tested are:

$$H_0 : \mu_{Diff} \leq 0$$

$$H_1 : \mu_{Diff} > 0$$

and

$$H_0 : \mu_{PC} \leq 0$$

$$H_1 : \mu_{PC} > 0$$

where  $\mu$  represents the mean of the variable. The test of the above hypotheses depends on the probability distribution of the variable. If the variable is normally distributed, then the best test to use is the well-known t-test. We therefore performed normality test on  $Diff$  and  $PC$ . The Lilliefors Goodness of Fit (GOF) test of normality available in the software package ProUCL was used for this purpose (Agency, 2011).

## Results

Figure 3.8 shows the result of normality test for *Diff*. Since the P-value for the normality test for *Diff* is 0.303, the assumption of normality of *Diff* cannot be rejected.

Figure 3.8: GOF Test of Normality for *Diff*

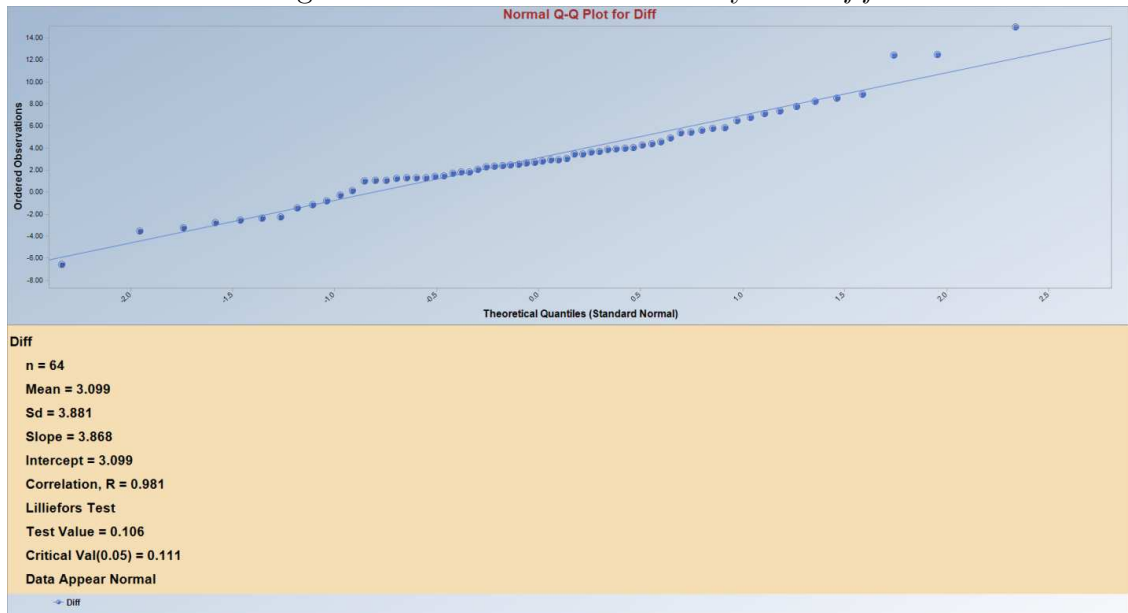


Figure 3.9: Histogram for Diff

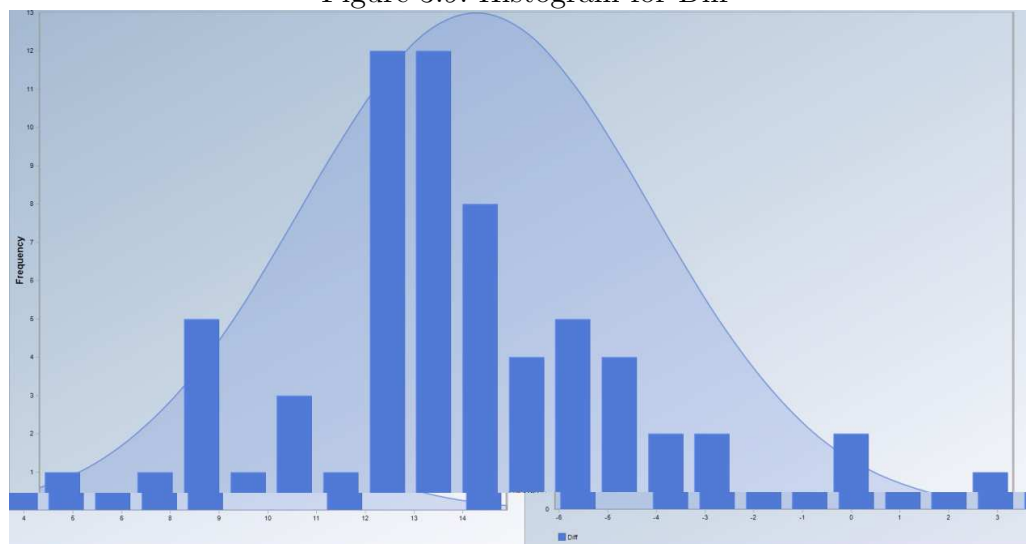


Figure 3.10 shows the result of normality test for *P*; since the P-value in this case is .027 the assumption of normality of *PC* is rejected at 5% test size. Figure 3.11

shows the histogram of  $PC$  values

Figure 3.10: GOF Test of Normality for  $PC$

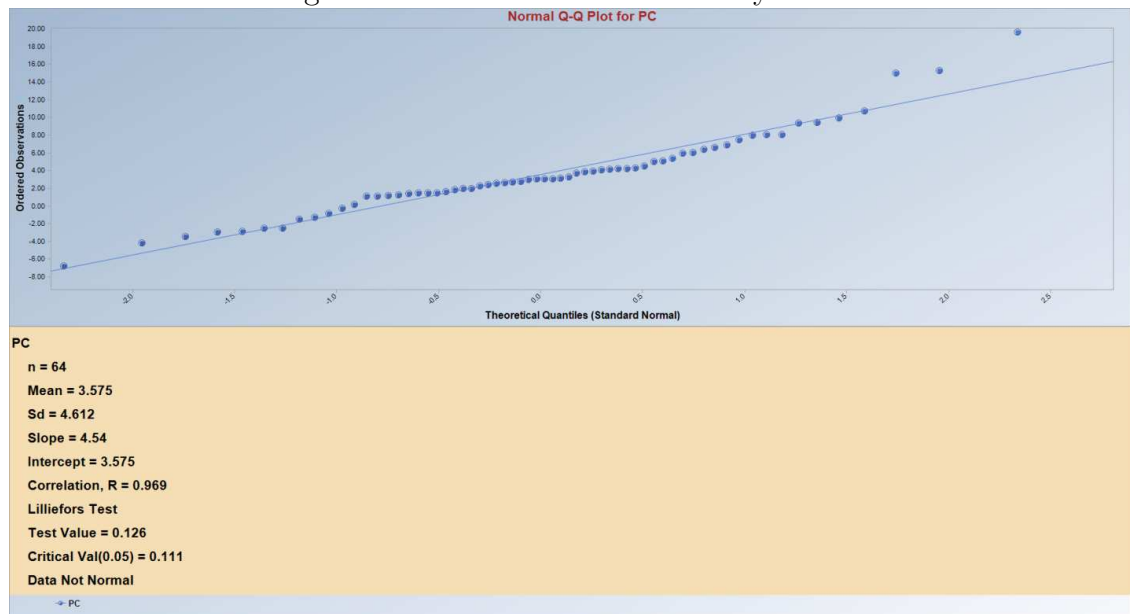
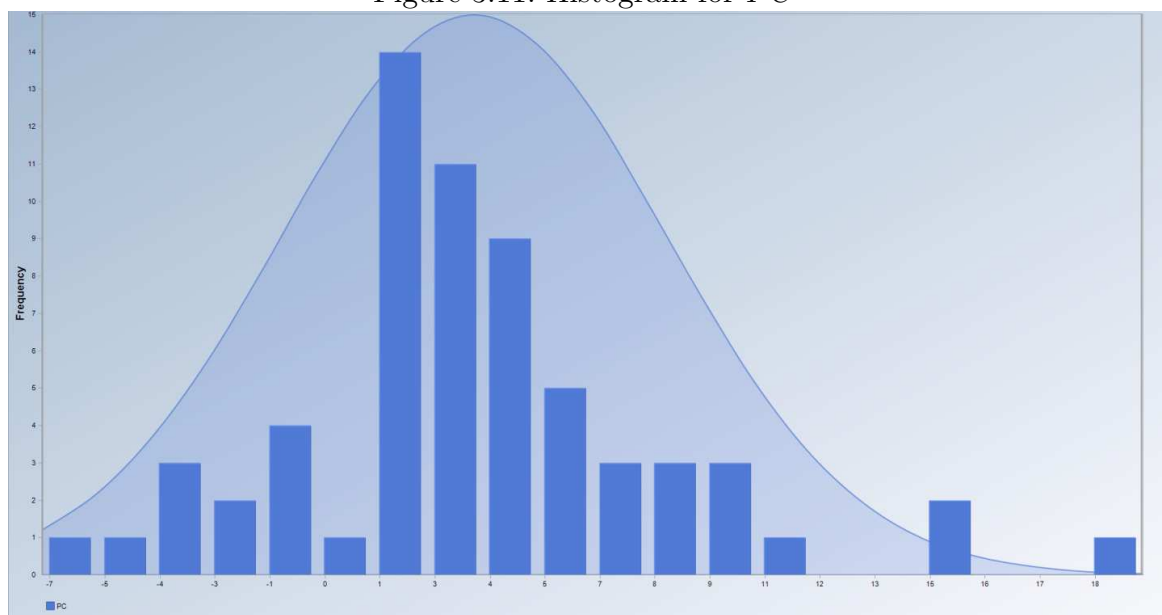


Figure 3.11: Histogram for  $PC$



Normality of the variable is required for using the 1-sample t-test. The normality of the sample mean of the variable, however, is all that we need. The normality of



the sample mean can be verified by drawing bootstrap samples, computing the mean of each sample, and then drawing the histogram of these sample means. The R-code used for this purpose is shown in Listing 3.1.

Listing 3.1: R code

```
x <- read.csv("Pre vs Post Aug 15 2010.csv",header=TRUE)

diff <- x[,4]
pc <- x[,5]

B <-1000
Diff_bar <-matrix(NA, nrow = 1000, ncol = 1)
PC_bar <-matrix(NA, nrow = 1000, ncol = 1)
# Draw B bootstrap samples
for (i in 1:B)
{
d <- sample(diff, replace=TRUE)
p <- sample(pc, replace=TRUE)

Diff_bar[i] <- mean(d)
PC_bar[i] <- mean(p)
}
layout(1:2)
hist(Diff_bar, nclass=20)
hist(PC_bar, nclass=20)
```

The histograms of bootstrap sample means of *Diff* and *PC* are shown in Figures 3.12 and 3.13, respectively. Figures 3.12 and 3.13 show that the sample mean of *Diff* and *PC* are approximately normal, and we therefore can use the 1-sample t-test to test the hypotheses. The t-tests were run in ProUCL; the t-test results are shown in

Tables 3.13 and 3.14. The P-values for both *Diff* and *PC* were close to 0, indicating that there is a significant increase in safety belt usage and also in percent-increase for each site. The 95% confidence intervals for the mean *Diff* is (2.13, 4.07) and that for *PC* is (2.43, 4.73).

Figure 3.12: Histogram for *Diff\_bar*

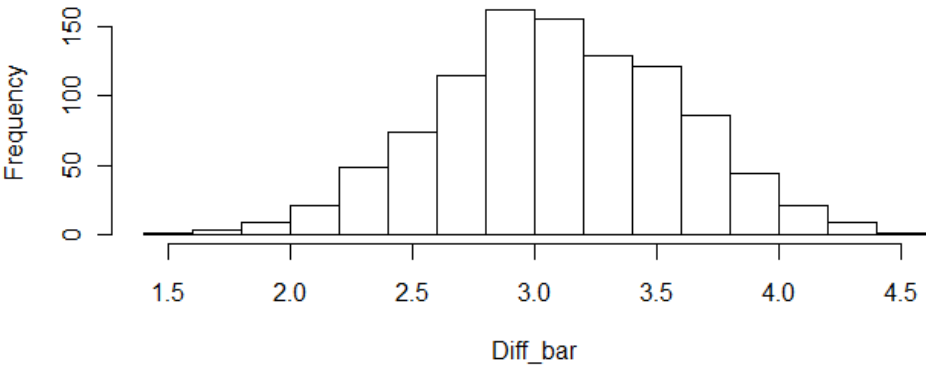


Figure 3.13: Histogram for *PC\_bar*

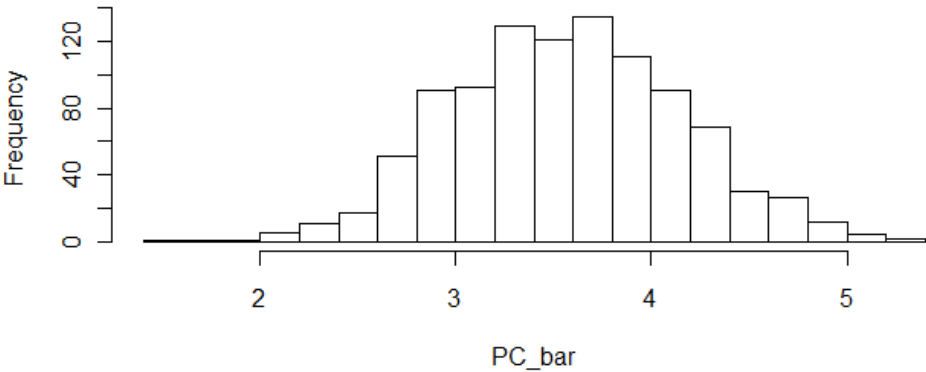


Table 3.13: Single Sample t-Test for *Diff*

Raw Statistics	Values
Number of Valid Observations	64
Number of Distinct Observations	64
Minimum	-6.57
Maximum	14.99
Mean	3.099
Median	2.771
SD	3.881
SE of Mean	0.485
$\hat{\theta}$ : Site Mean $\leq 0$ (Form 1)	
Test Value	6.389
Degrees of Freedom	63
Critical Value (0.05)	1.669
P-Value	1.1474E-8
Conclusion with Alpha = 0.05 Reject H0, Conclude Mean $> 0$	
<i>continued on next page</i>	

Raw Statistics	Values
P-Value < Alpha (0.05)	

Table 3.14: Single Sample t-Test for  $PC$

Raw Statistics	Values
Number of Valid Observations	64
Number of Distinct Observations	64
Minimum	-6.798
Maximum	19.61
Mean	3.575
Median	3.089
SD	4.612
SE of Mean	0.577
H0: Site Mean $\leq 0$ (Form 1)	
Test Value	6.201
Degrees of Freedom	63
Critical Value (0.05)	1.669
<i>continued on next page</i>	

Raw Statistics	Values
P-Value	2.4057E-8
Conclusion with Alpha = 0.05 Reject H0, Conclude Mean > 0 P-Value < Alpha (0.05)	

### 3.3.6.2 Weighted Analysis

The 95% confidence intervals for the weighted safety belt usage percentages for the State of Nevada were found to be:

- Pre-Mobilization: (89.36%, 91.49%)
- Post-Mobilization: (92.18%, 94.13%)

Since the entire 95% confidence interval for Post-Mobilization safety belt usage fell to the right of the 95% confidence interval for Pre-Mobilization safety belt usage, we concluded that the statewide safety belt usage went up after the Mobilization.

## CHAPTER 4

### DATA COLLECTION SOFTWARES FOR SAFETY BELT STUDIES

#### Abstract

Every year, US government conducts nationwide 'Click It or Ticket' mobilization during the month of May to increase public awareness of safety belt usage among the people. Throughout US, a lot of money is spent on advertising safety belts through the radio/tv, directing law enforcement and conducting surveys before and after this mobilization. These surveys form an integral part of the Daytime Safety Belt Usage studies. More than 10,000 observations in the two mobilization periods (before and after) are performed by trained observers who stand by the side of the roadways on intersections, freeway ramps and sometimes on freeways as well. These observers have different data collection templates to collect the same variables: safety belt usage and characteristics of front seat occupants, the vehicle type and the state of registration. In some U.S. states, paper is still the main data collection template, where in other states, personal digital assistants are commonly used. However, the design of the data collection template is of utmost importance, especially when the observations are performed on roadways where the vehicles are passing by at 75 m.p.h ! In such situations, the importance of a well designed graphical user interface comes into picture and a question naturally arises: which software template should be used which provides speed, accuracy and flexibility at the same time. In this chapter, five data collection templates have been compared on the basis of aforementioned criteria. Out of these five templates, the 2010 PDA design (see Figure 4.5) and the iPhone design (see Figure 4.11) have been newly developed and proposed in this thesis. The

comparison reveals that the newly developed designs allow the observers to collect much more data at higher speeds without sacrificing the accuracy.

#### 4.1 Data Collection Templates

In this section, we provide five templates for collecting the safety belt usage data. One of these templates is paper which has been used in US for a long time. The second template was designed in 2006 through a behavioral study by Vivoda & Eby (2006). The rest three templates have been designed at the Transportation Research Center, UNLV. One of these templates was designed on a Personal Digital Assistant(PDA) and has been used from 2007-09. Another PDA design was presented in 2010 with a matrix format data collection template described later in this section. The same template was then programmed on iPhone 4 to make further strides in data collection techniques. All these five templates have been presented with their advantages and shortcomings in this section. .

##### 4.1.1 Template 1: Paper & Pen

The standard paper and pen template is the first data collection tool being used in U.S. This template provides a basic layout to the trained observers who either tick a mark on the corresponding column or enter a letter alphabet as shown in Figure 4.1. As shown in Figure 4.1, the observers have been trained to enter 1 in the last column *Vehicle Type* if the observed vehicle is a Sedan or a Station Wagon. Similarly, N is entered as a single letter alphabet in the *State* column. These shortcuts have been assigned to increase the speed of data collection at every site. In spite of these shortcuts, several entries are required for a single vehicle on this paper and pen

template. In the worst case when both front seat occupants are present in a single vehicle, the observer has to enter a letter alphabet or a single tick mark 8 times. Also, the observers need a cardboard to stand on the roadsides and in highly windy states like Nevada, the cardboard-paper duo might face a few challenges. Nevertheless, the paper and pen template is still a widely used tool to collect safety belt usage data.

Figure 4.1: Template 1: Paper & Pen

Site # \_\_\_\_\_ End Time (for Sheet): \_\_\_\_\_  
City: \_\_\_\_\_ Weather: \_\_\_\_\_

Vehicle	Driver			Driver Agegroup			Passenger			Passenger Agegroup			State	Vehicle type
	SB	No SB	Ethnicity	15-19	20-60	>60	SB	No SB	Ethnicity	<15	15-19	20-60		
1														
2														
3														
4														
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														
18														
19														
20														
21														
22														
23														
24														
25														

SB: Seat Belt Used  
No SB: No Seat Belt Used  
For both Driver and Passenger information:  
Mark M (for Male) or F (for Female) in either of SB or No SB field.  
Ethnicity: Mark C (Caucasian), A (African American), H (Hispanic) or O (Others)  
Agegroup: Check on corresponding Agegroup  
State: State of Vehicle Registration. Mark N (Nevada), C (California), or O (Others)  
Vehicle type: Mark 1 (Sedan/Station Wagon), 2 (Pick-up), or 3 (Van/SUV)

#### 4.1.2 Template 2: 2006 PDA Design (Vivoda & Eby, 2006)

This PDA design was programmed in 2006 to compare the safety belt data collection through paper and the PDAs. This software required HanDBase to be purchased, which is a fast relational database manager for the PDAs and offers standard data



queries and synchronization features. The exact design used in the study is shown in Figure 4.2.

Figure 4.2: The design used in 2006 study (Vivoda & Eby, 2006)

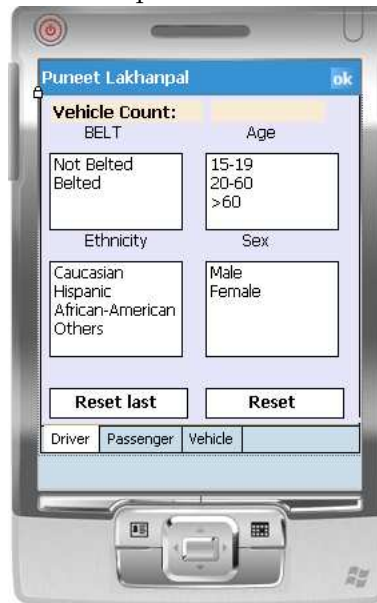
The figure displays a three-tabbed user interface for data collection. The tabs are labeled 'Driver', 'Passenger', and 'Vehicle'.  
 - The **Driver** tab contains three sections: 'Belt' with a list of options (Not Belted, Belted, B Back, U Arm), 'Age' with a list (4-15, 16-29, 30-59, 60+), and 'Sex' with a list (Male, Female). A 'Prev Veh' button is at the bottom left.  
 - The **Passenger** tab starts with a checkbox labeled 'No Passenger'. Below it are similar sections for 'Belt', 'Age', and 'Sex'.  
 - The **Vehicle** tab includes a 'Type' dropdown menu with options: Passenger Car, Van/Minivan, SUV, and Pickup Truck. Below this is a 'Commercial' dropdown with options: Commercial and Not Commercial. At the bottom are three buttons: 'Next Vehicle', 'End Site', and 'Cancel'.

The above figure shows a tabbed design with three pages: driver, passenger and vehicle. On all these pages, the user has to tap a choice with a stylus and then move on to the next tab. Going in a sequential manner, the design required the observers to click the *No Passenger* checkbox if there was no passenger in the vehicle. Thus, one could not skip over the *Passenger* tab even if the vehicle had no front seat passenger. After entering the vehicle details, the observer needs to tap the *Next Vehicle* button to start fresh data collection for the next vehicle.

Following these guidelines, a design similar to Figure 4.2 was programmed in C# language through Visual Studio 2008 Professional Edition IDE. This edition is free for university students and is available online (Microsoft, 2011). Also, Ultralite database, an extremely compact and low memory footprint database was used for relational database management and was available free of cost under the Sql Anywhere 10 developer edition. Currently, Sybase has launched Sql Developer 12 edition and is available online (Sybase, 2011). Figure 4.3 shows the programmed design similar to the 2006 design presented in Figure 4.2. The programmed design was deployed on

HP IPaq PDAs running Windows Mobile 6 platform.

Figure 4.3: Template 2: 2006 PDA design



Comparing the designs presented in Figure 4.2 and 4.3, the 2006 PDA design was implemented with a few modifications. Firstly, the three tab bars for *Driver*, *Passenger* and *Vehicle* were moved to the bottom of the page because of such restrictions in Visual Studio. Secondly, the variables and the choices amongst these variables were modified according to the Daytime Safety Belt Studies project in Nevada. For example: The *Ethnicity* variable was added to both *Driver* and *Passenger* tabs. Thirdly, an automatically updating vehicle counter was added to the top of the page to keep the observer informed of the current count. Lastly, *Reset* and *Reset Last* buttons were also provided to do the jobs similar to their names.

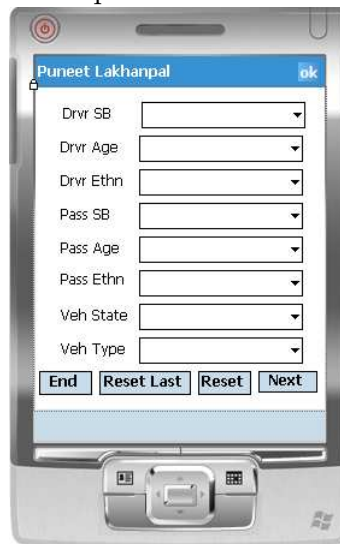
In the worst case scenario when both the front seat occupants have to be observed, an observer needs to tap 4 times each on the *Driver* and the *Passenger* tab and twice on the *Vehicle* tab. Assuming no mistake was made and counting the Next button

tap on the *Vehicle* tab, a total of 11 taps are required by a stylus in this design for recording the data for a single vehicle. However, in the best case when only a driver is observed in the vehicle, 8 taps will be required including the *Next* button.

#### 4.1.3 Template 3: 2007-09 PDA Design

Another PDA design shown in Figure 4.4 was being used from 2007-09 for the Daytime Safety Belt Studies at Transportation Research Center, UNLV. The original design was developed in ArcPad 7.0 software on the PDA units employing Pocket PC 2003 operating system and did not have the *End*, *Reset* and *Reset Last* buttons shown in Figure 4.4. These buttons were recently added to the original design to compare the five templates mentioned in this chapter. Hence, the original design was re-programmed in C#, again using Ultralite database and was deployed on much recent HP IPaq PDAs running Windows Mobile 6 platforms.

Figure 4.4: Template 3: 2007-09 PDA Design



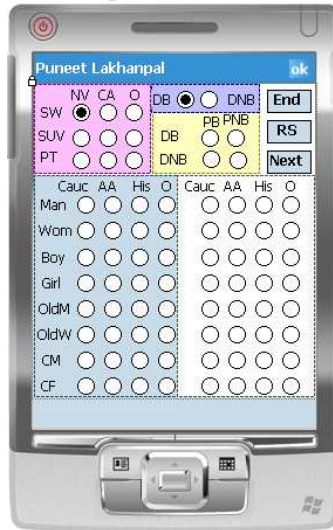
In the worst case scenario when both the front seat occupants have to be observed, an observer needs to tap twice on every drop down menu shown in Figure 4.4. As-

suming no mistake was made and counting the Next button tap on the screen, a total of 17 taps are required by a stylus in this design for recording the data for a single vehicle. However, in the best case when only a driver was observed in the vehicle, 11 taps would be required by the observer including the *Next* button tap.

#### 4.1.4 Template 4: 2010 PDA Design

The PDA design shown in Figure 4.5 was designed in 2010 at Transportation Research Center, UNLV and was used for the Daytime Safety Belt Studies project in Nevada during 2010. The programming was again done in C# and Ultralite using the Visual Studio development environment. The software was deployed on HP IPaq PDAs running Windows Mobile 6 platform. Note that the development for the PDA devices in Sections 4.1.2, 4.1.3 and 4.1.4 required installation of Windows Mobile 6 Professional SDK downloadable from Microsoft Download Center (2007).

Figure 4.5: Template 4: 2010 PDA Design



A theory entitled 'One-Click Solution on Spatially Unconstrained Space' was developed at Transportation Research Center, UNLV during 2010. The design shown in

Figure 4.5 is motivated from this theory. According to this theory, if unlimited space is available for designing a graphical user interface, then the data for theoretically any number of variables can be collected with a single click on that user interface.

The above statement is quite obvious if only one variable exists. However, if two variables are required in the study, lets say  $X = (X_1, X_2, X_3, X_4 \dots X_{m-1}, X_m)$  and  $Y = (Y_1, Y_2, Y_3, Y_4 \dots Y_{n-1}, Y_n)$ , a one-click solution can be still be obtained by aligning both the variables in a matrix format. Mutually exclusive radio buttons can be used to represent one pair of variables  $(X_i, Y_j)$  where,  $i = (1, 2, \dots, m)$  and  $j = (1, 2, \dots, n)$ . In the case of unconstrained design surface, it does not matter whether radiobuttons are used or checkboxes are used to represent the variables. The obtained matrix design is showed in Figure 4.6.

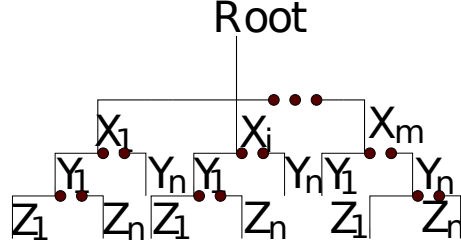
Figure 4.6: Matrix design for two variables

	$X_1$	$X_2$	$X_3$	...	$X_{m-1}$	$X_m$
$Y_1$	●	●	●	...	●	●
$Y_2$	●	●	●	...	●	●
$Y_3$	●	●	●	...	●	●
$Y_4$	●	●	●	...	●	●
$Y_5$	●	●	●	...	●	●
...	●	●	●	...	●	●
...	●	●	●	...	●	●
$Y_{n-1}$	●	●	●	...	●	●
$Y_n$	●	●	●	...	●	●

If another variable  $Z = (Z_1, Z_2, Z_3, Z_4 \dots Z_{p-1}, Z_p)$  is considered along with  $X$  and  $Y$ , one-click solution can still be obtained. Basically, a hierarchy would be obtained for all the variables, whether it be 2 variables, 3 variables or  $t$  variables where  $t = (1, 2, \dots, \infty)$ . Due to the drawing constraints, only three variables have been shown

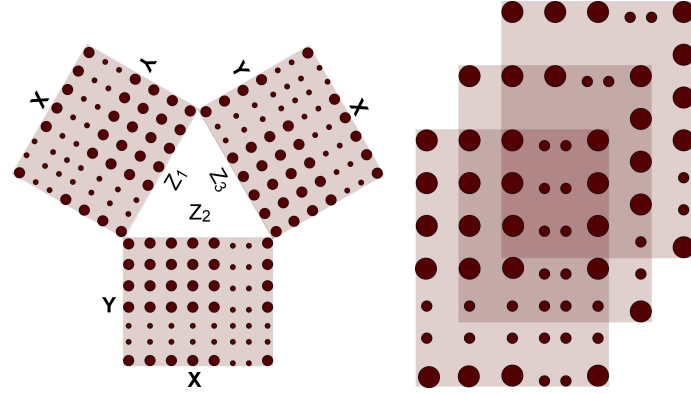
in Figure 4.7.

Figure 4.7: Hierarchial structure for theoretically  $\infty$  variables (3 shown)



If  $p$  represents the number of fields in the  $Z$  variable, the structure can also be represented in form of other algebraic structures. For example: If  $p = 3$ , either all the 3 fields  $Z_1, Z_2, Z_3$  can be arranged in a triangular format or can simply be arranged as 3 copies of the matrix derived from the Figure 4.6. This is represented in the Figure 4.8.

Figure 4.8: Possible structures for 3 variables in data collection

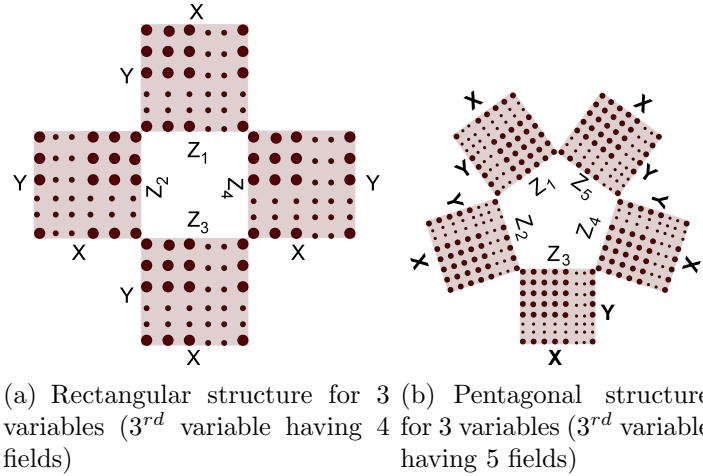


(a) Triangular structure for 3 variables ( $3^{rd}$  variable having 3 fields) (b) Planar structure for 3 variables ( $3^{rd}$  variable having 3 fields)

Similarly, rectangular and pentagonal structures can be obtained for 4 and 5 fields in the  $Z$  variable respectively. These structures are shown in Figure 4.9.

Thus, enhancing the variables or the fields within a variable only increases the copies of the preceding-level structure (i.e. when variables =  $k$ , structure repeated =

Figure 4.9: More possible structures for 3 variables in data collection



structure obtained in  $k - 1$  level). In any case, one-click symmetric solution is always obtained.

Now, for Daytime Safety Belt Studies, the design surface of PDA is quite small to build structures which can attain one-click solution to enter the data for 10 variables (Safety Belt usage, age, gender and ethnicity for both front seat occupants, state of registration of the vehicle and the vehicle type). The HP IPAQ models (210) being used are 3 inches wide and 5 inches long. However, the screen is slightly more than 2 inches wide and 3 inches long. Thus, an optimal solution on this constrained space is bound to have more than one click.

In the design presented in Figure 4.5, the following shortcuts were taken to obtain an optimal click solution on the constrained PDA space:

1. Combining the variables:

A user interface designer can combine any number of variables to reduce the number of required clicks. In such a process, he would give another name to the combined variable. However, it is important that the combined name should

not fall into the category of technobabble: jargon which only the developers can understand but the most users cannot. In other words, the new name should be easily recognizable to any user. For example: Combining age (20-60) and gender (Male) variables will make sense since 'Man' is a very common term. Similarly, women, elderly men, elderly women etc. are terms which, although are combination of age and gender variables, but are commonly used in day-to-day life. On the other hand, it is better to combine two variables in matrix format rather than displaying all the permutations of the two variables in a single. For example: Figure 4.10 shows the variables 'Vehicle Type' and 'State of Registration' combined in a 3x3 matrix format. It is much better than creating 9 radio buttons (3x3) and putting a label on each of them.

Figure 4.10: Combining the variables

	NV	CA	O	DB	<input checked="" type="radio"/>	DNB
SW	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	DB	<input type="radio"/>	PNB
SUV	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	DB	<input type="radio"/>	<input type="radio"/>
PT	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	DNB	<input type="radio"/>	<input type="radio"/>

## 2. Static Default selections:

A lot of user clicks can be saved by keeping some fields of the necessary variables to be selected as default. The selection of the default can be static and be based upon the data collected in previous years. For example: 2009 safety belt usage surveys in Nevada reported 52.5% vehicles to be Sedan/Station Wagons throughout the state out of the three vehicle categories (Sedan/Station Wagon, Van/SUV and Pickup). Thus, at every site, this field can be selected by default. Similarly, 89.7% of the observed vehicles belonged to Nevada. Thus, the NV



state of registration can be selected by default. Figure 4.10 shows 'NV', 'SW' within the pink panel (NeVada, Sedan/Station Wagon) to be selected by default, thereby making use of the prior data. Last but not the least, since the driver information is always needed and around 89.4% of the drivers were belted in the 2009 survey, the drivers can be considered to be belted by default. Figure 4.10 shows this in which 'DB' within the violet panel (driver belted) has been chosen by default. Hence, these proposed measures can greatly reduce the number of clicks. Of course, these default categories can be changed in any scenario other than the above.

### 3. Reset button:

To obtain a minimum click solution on the PDA, the radio buttons were laid out in a matrix format on a single screen. This does not leave much space for incorporating every single control that a developer wants. In other words, the *Reset* and *Reset Last* functionality were clubbed together into a single button control named *RS* placed at nearly the top of the PDA screen. When the observer tapped the *RS* button with a stylus, an alert was shown which gave two options to the observers to either reset the current screen or retrieve the last entry inserted into the database and update it.

In the worst case when the observed vehicle is something other than a Nevada Sedan/Station Wagon and both front seat occupants are present, 5 taps will be required including the tap of the *Next* button. However, in the best case scenario when only a belted driver is observed driving a Nevada Sedan/Station Wagon, only 2 taps

will be required including the *Next* button tap.

#### 4.1.5 Template 5: iPhone

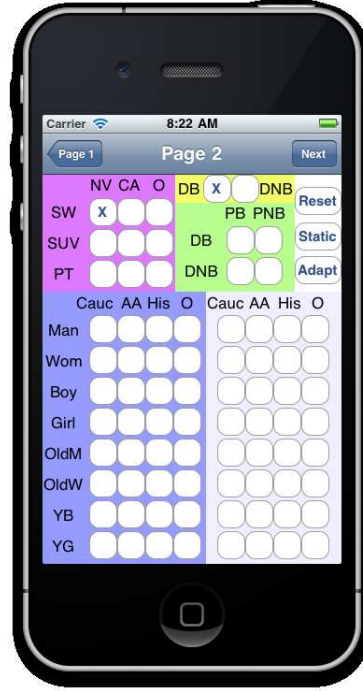
With the boom of smartphones like iPhone 4 and Google Android, it is necessary to use the capabilities of the smartphones and act accordingly. Nowadays, almost everyone has a smartphone. However, we rarely see anyone with a PDA in hand and using it on an everyday basis. More importantly, most of the PDA's are devoid of internet access due to the inavailability of sim card slots in them. However, smartphones clearly win in this area too. Hence, an iPhone application was developed due to their obvious advantages in the following areas:

1. The iPhone application can be easily downloaded once available in the App Store, making the data collection software much more accessible to everyday user.
2. The iPhone application can easily transfer the data through the wi-fi/3G/4G internet connection to the data servers. Thus, it would reduce the efforts to download the data to a laptop first and later upload it online.

It was decided that the data collection template shown in Figure 4.5 will be used for developing the iPhone application too. Thus, the developed template in iPhone 4 is shown in Figure 4.11.

Since the same template is used in iPhone 4 as the 2010 PDA template shown in Figure 4.5, 5 clicks are required in the worst case and 2 clicks in the best case, as mentioned in Section 4.1.4. Note that in this study, a stylus was not used for an iPhone. However, if a stylus would have been used, the scenario of entering the data

Figure 4.11: Template 5: iPhone



entries through an iPhone would be very similar to entering data in the 2010 PDA design (for which stylus is required) since the user interface screen is the same.

## 4.2 Comparison Study of five templates

In this section, all the five data collection templates in Figures 4.1, 4.3, 4.4, 4.5 and 4.11 have been compared. First, the plan for the study has been provided and then results have been discussed. Additionally, three statistical tests have been performed on the collected data to ensure its reliability.

### 4.2.1 Study Procedure

The following plan was implemented for comparing the data collection templates:

1. We have 6 different strata for Daytime Safety Belt Usage studies: Rural High, Rural Medium, Rural Low, Urban Medium, Urban High, Urban Low. There-

fore, to cover all these regions, one site was chosen from each strata in Clark county area. The sites where data collection was performed for the comparative study are:

- (a) Rural High : I-15 between Henderson and Arden
- (b) Rural Medium : US 95 between 157 and Indian Springs
- (c) Rural Low : SR 161 at I-15
- (d) Urban High: I-15 at Sahara
- (e) Urban Medium: Maryland at Charleston
- (f) Urban Low: Torrey Pines at Spring Mountain

2. Data collection was performed on the following dates and times:

- (a) March 16<sup>th</sup> - March 17<sup>th</sup>: 9 a.m. to 5.00 p.m.
- (b) March 21<sup>st</sup>- March 23<sup>rd</sup>: 9 a.m. to 5.00 p.m.

Therefore, the data was collected for an overall time period of 5 days. Friday and weekends were intentionally neglected since the traffic pattern during these days can be totally different than the other weekdays.

3. Data Collection procedure: Two methods of data collection could have been used.

- (a) Go to only one site every day and collect data using all templates for 40 minutes each. Although this approach was convenient, this approach was not used since the traffic pattern could have been different during different

time periods. For example: The % data collected using paper during 12-12.40 pm could have been totally different than % data collected using PDA during 3-3.40 pm. Hence, in order to avoid the time variations (and of course rush hour variations), this method was not used.

- (b) Go to 6 chosen sites every day (for 5 days) and collect data using only template per day. For example: Choose paper for data collection on one day and use only that to collect data at all the 6 sites. Next day, use another template and repeat the same procedure. A time schedule was followed for this method. For example: Lets say the data was collected using paper at Site A during 10-10.40 am. Next day, another template was used to collect the data at same site A during 10-10.40 am. This took care of the time variations and the rush hour effects.

Hence, the second method was used for conducting the comparative study.

4. Apart from the usual variables collected in the Daytime Safety Belt Usage Surveys, actual traffic count during the 40 minutes data collection period was also observed using a smartphone app called *T-Counter* shown in Figure 4.12.

#### 4.2.2 Comparison 1: Speed

In Sections 4.1.1, 4.1.2, 4.1.3, 4.1.4 and 4.1.5, the number of stylus taps (on PDAs and iPhone) or marks (on paper) in the best and the worst case scenarios were discussed. This gives a rough idea as to how these data collection templates should perform in the real world. In this section, the results of testing these data collection templates on six pre-determined sites have been discussed. Site selection for

Figure 4.12: Traffic Counter

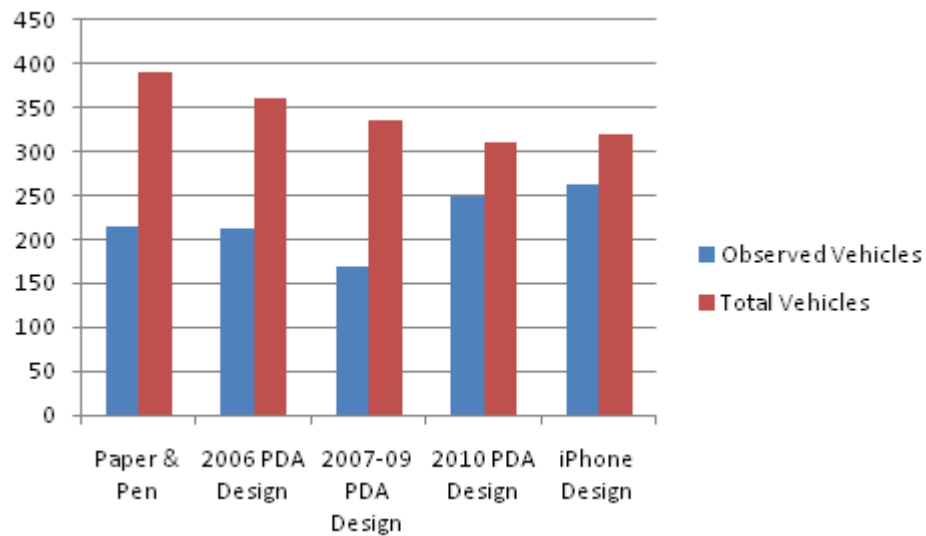


the comparative study was performed in such a manner that one roadway segment belonging to each strata (rural/urban and high/medium/low volume) was chosen. Since every roadway segment differed either in terms of the area or the traffic volume levels, interesting results were obtained while comparing the speed of these templates. For comparing the speed of the data collection templates, percentage data collected was considered to be the only measure. In other words, speed of a data collection template at a particular site referred to how many data entries could be made on a template out of the number of vehicles that passed by an observer. For this study, three observers were used. Two observers made the data entries in the templates while one observer tapped on a counter shown in Figure 4.12 to count the actual number of vehicles passing by. All the three observers were guided not to consider tinted vehicles in the study.

#### 4.2.2.1 I-15 between Henderson and Arden

This site is a Rural High Volume Site. At this site, the data was collected standing on the side of a freeway. As mentioned before, two observers collected safety belt usage data while one observer counted the number of non-tinted vehicles passing by. The results are shown in Figure 4.13.

Figure 4.13: Speed comparison: I-15 between Henderson and Arden



(a) Data collected in numbers



(b) Data collected in percentage

Figure 4.13 suggests that comparable percentage of data was collected using both the iPhone and the 2010 PDA design. This is an expected phenomena, since the structure of user interface for data collection is same in both these designs. However, there is a significant difference between the data collected through newer templates (iPhone and 2010 PDA design) and comparatively older templates (Paper & Pen, 2006 PDA design, 2007-09 PDA Design). The 2007-09 design showed in Figure 4.4 collected the least amount of data. Also, the 2006 design showed in Figure 4.3 performed better than the paper template. Such a performance of all the five templates suggests that there can be a considerable difference in the data collection speed at high volumes, especially on freeway locations.

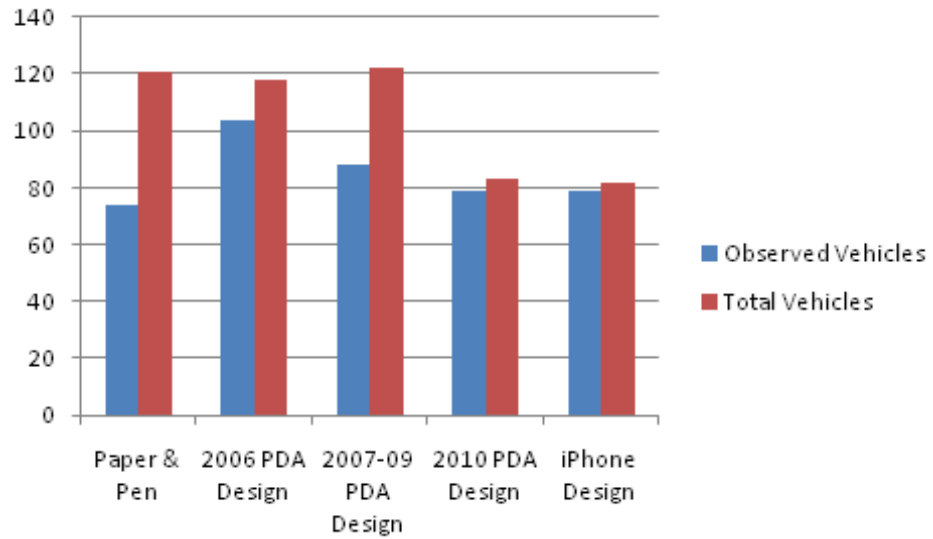
#### 4.2.2.2 US-95 between SR 157 and Indian Springs

This site is although a Rural Medium site, data at this site has to be collected while standing at the side of US-95 freeway. Only two lanes were observed at this site, one lane by each observer. The data collected through the five templates at this site is shown in Figure 4.14.

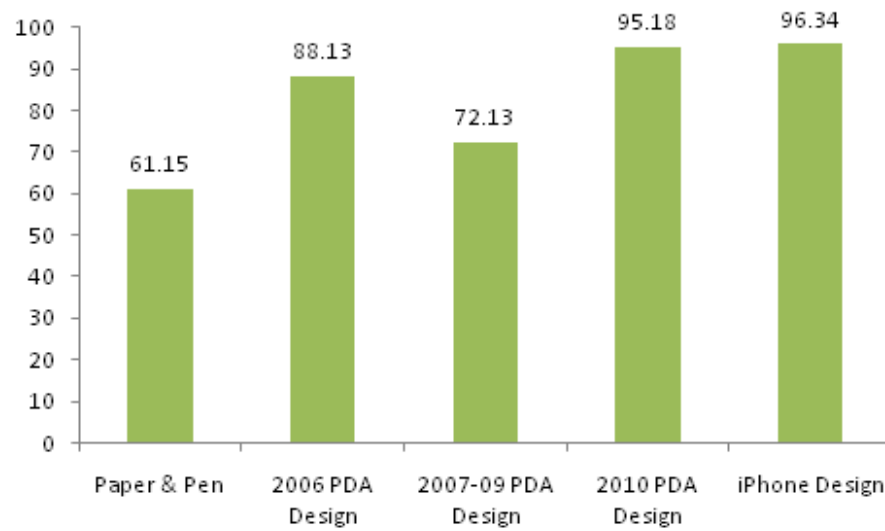
In Figure 4.14, it can be observed that the total vehicles passing by during the data collection through iPhone and 2010 PDA design are comparatively lower than the total vehicles passed during paper, 2006 PDA design and 2007-09 PDA design. Still, the high percentage of data entered through iPhone and 2010 PDA design suggests their faster speeds in comparison with other templates.



Figure 4.14: Speed comparison: US-95 between SR 157 and Indian Springs



(a) Data collected in numbers

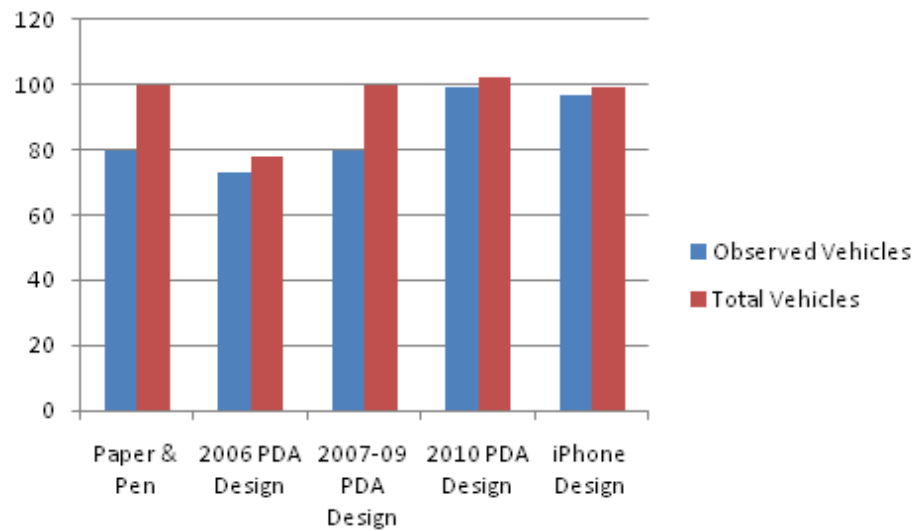


(b) Data collected in percentage

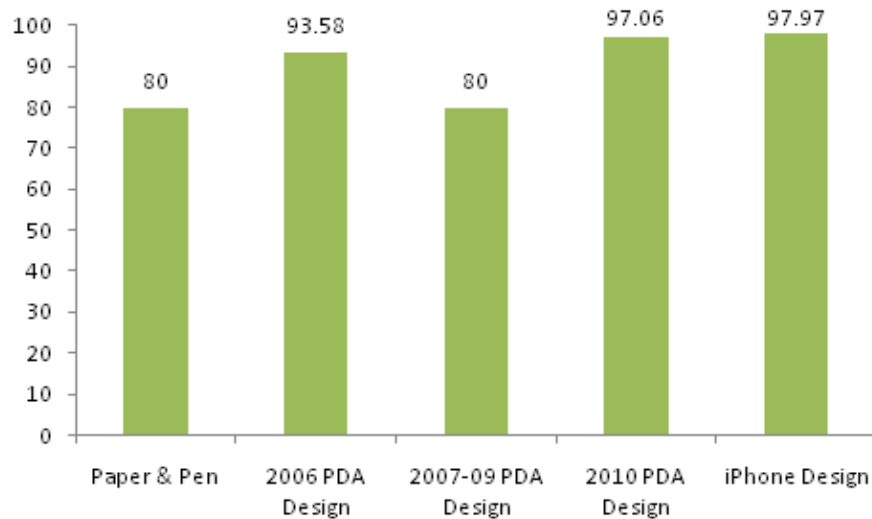
#### 4.2.2.3 SR-161 at I-15

This site is a Rural Low Volume Site. At this site, the observers could easily stand in a triangular region in the middle of the road, but only while wearing safety vests. The data collection at this site was performed in all the directions, due to the low volume at this site. The results at this site are shown in Figure 4.15.

Figure 4.15: Speed comparison: SR-161 at I-15



(a) Data collected in numbers



(b) Data collected in percentage

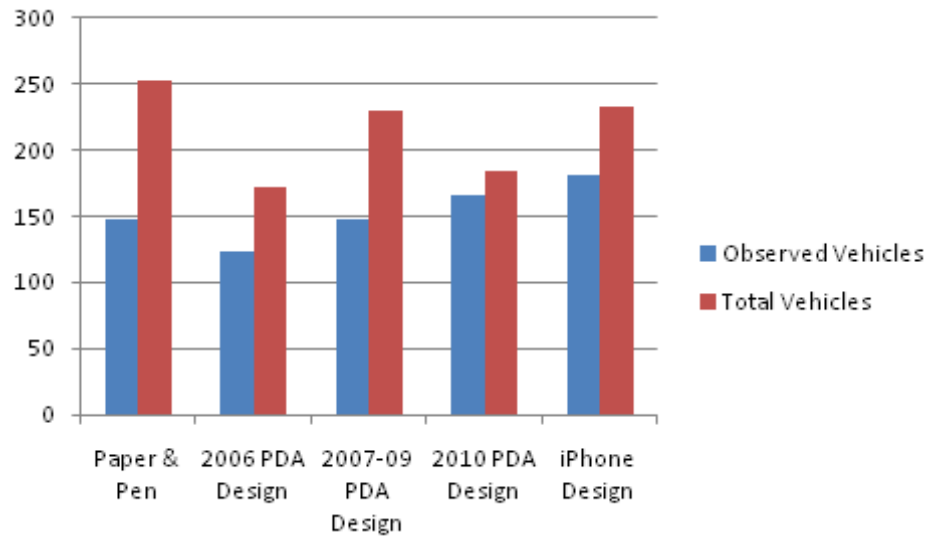
Figure 4.15 suggests that high percentages of data were collected at this site using all the data collection templates. However, the 2006 PDA design, 2010 PDA design and iPhone design showed exceptionally high rates of data collection. Still, the data collection percentage in aforementioned three PDA templates was higher than the Paper & Pen template. Again, lower data collection rates were shown by the 2007-09 PDA design.

#### 4.2.2.4 I-15 at Sahara

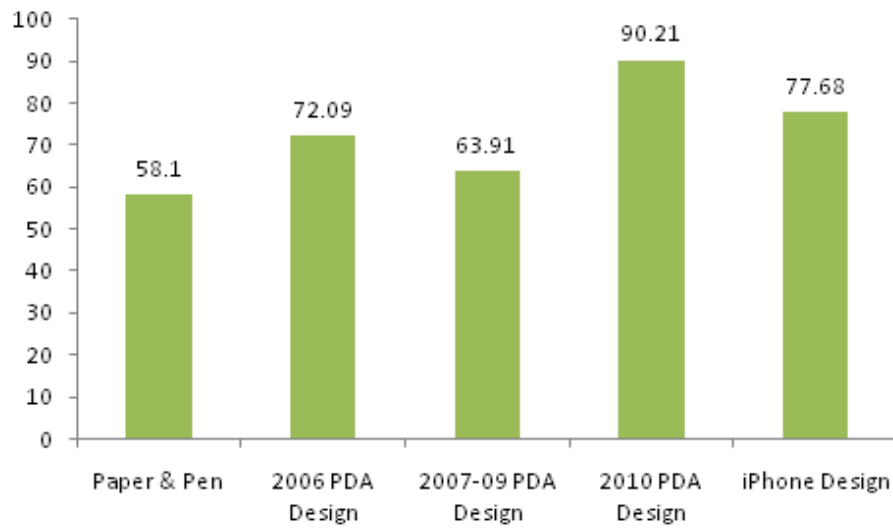
This site is an Urban High volume site. At this site, observers could stand close to a traffic light. Therefore, the vehicles stopped at this traffic light and data collection could easily be performed. Only two lanes were observed at this site, one lane by each observer. The results of data collection at this site is shown in Figure 4.16.

Figure 4.16 suggests that the 2010 PDA design outperformed the iPhone design at this particular site. A possible explanation is that it was a bit difficult entering the data entries without a stylus in iPhone. This was because all the button controls to be triggered were on the same screen and sometimes, touching one button caused a tap on some other button. Therefore, while this data entry was corrected, some vehicles passed which lead to a reduction in data collection percentage for the iPhone design. Still, the performance of the iPhone design was better than the rest of the templates (excluding 2010 PDA design). It is worth noticing that although the percentages of data collected through Paper & Pen template and 2007-09 PDA design are comparable, still the Paper & Pen template showed poor data collection performance at this site.

Figure 4.16: Speed comparison: I-15 at Sahara



(a) Data collected in numbers

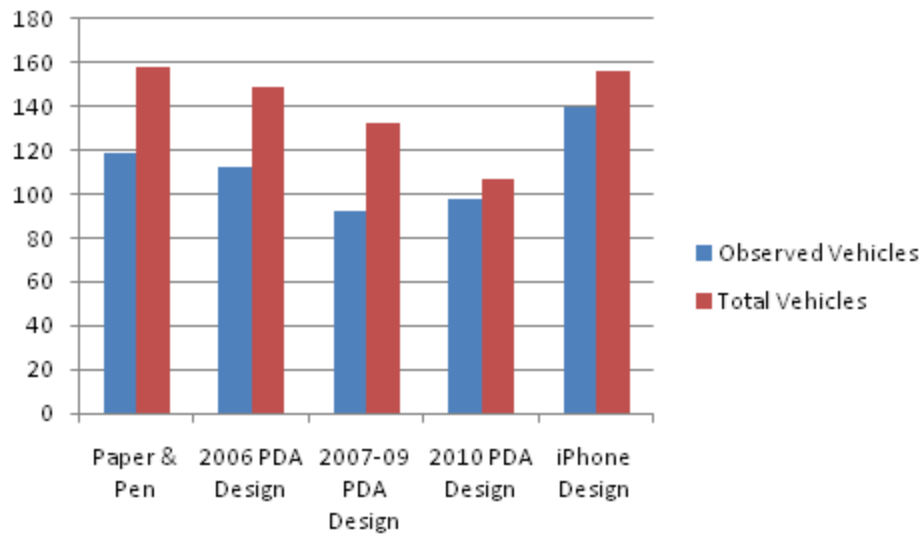


(b) Data collected in percentage

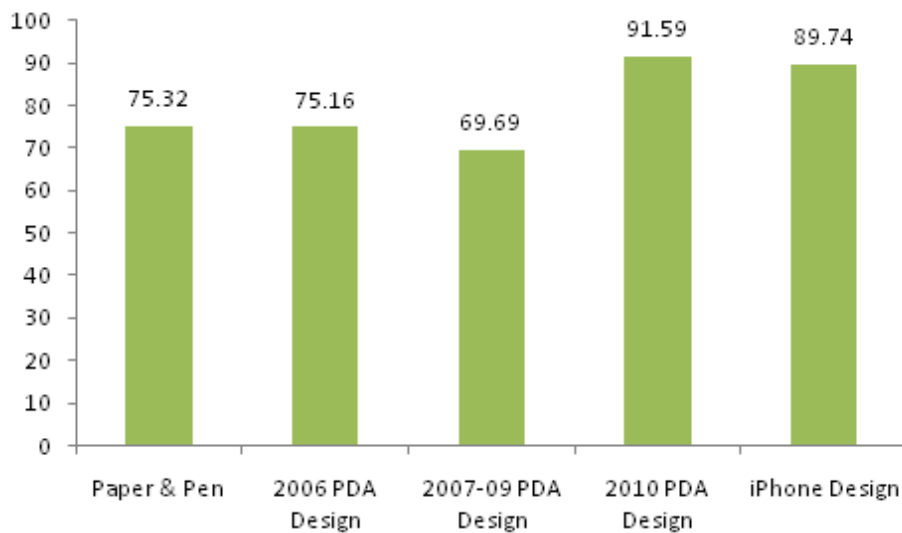
#### 4.2.2.5 Maryland at Charleston

This site is an Urban Medium volume site. At this site, the observers could stand at the intersection of two roadways and therefore, next to a traffic light. Only two lanes were observed at this site, one lane by each observer. The results of data collection at this site are shown in Figure 4.17.

Figure 4.17: Speed comparison: Maryland at Charleston



(a) Data collected in numbers



(b) Data collected in percentage

As observed in Figure 4.17, close to 90% data was observed in both, the iPhone design and the 2010 PDA design. The Paper & Pen template and the 2006 PDA design showed similar performances while the least percentage of data was collected through the 2007-09 PDA design.

#### 4.2.2.6 Torrey Pines at Spring Mountain

This site is an Urban Low volume site. At this site, the observers could again stand at the intersection of two roadways and therefore, next to a traffic light. Only two lanes were observed at this site, one lane by each observer. The results of data collection at this site are shown in Figure 4.18.

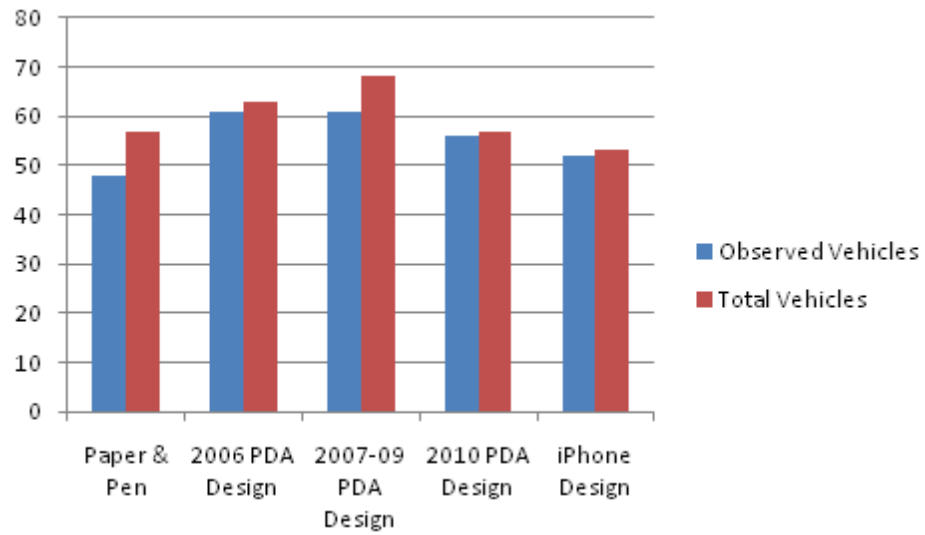
As observed in Figure 4.17, over 95% of data was observed through iPhone, 2010 PDA and the 2006 PDA design. The Paper & Pen template and the 2006 PDA design showed similar performances while the least percentage of data was collected through the 2007-09 PDA design. The Paper & Pen template performed well at this site, allowing the observers to enter close to 85% of the data. However, this performance was clearly the lowest among all the five tested templates.

#### 4.2.2.7 Overall Results

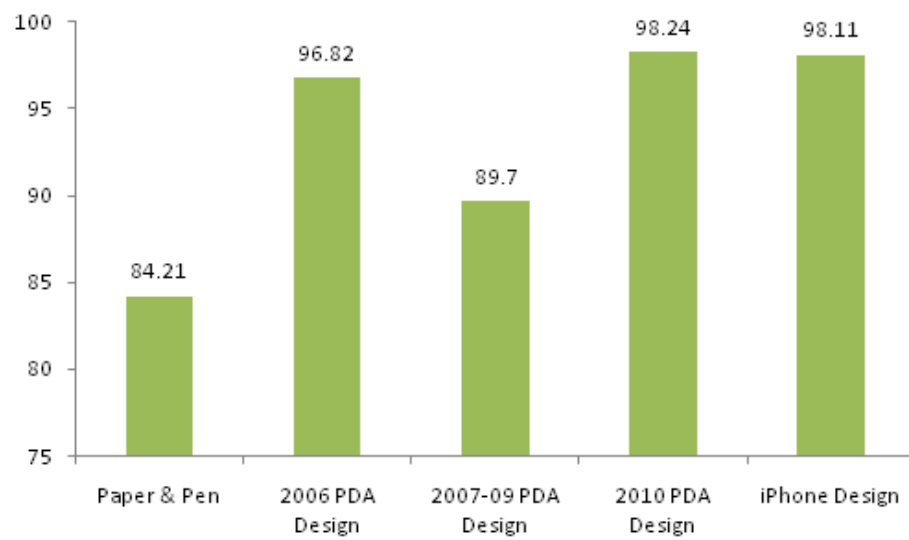
Figure 4.19 shows the comparison of the speed of the five templates when the data from all the six sites was combined.

In Figure 4.19, it can be observed that the iPhone and the 2010 PDA design allowed the observers to collect the most percentage of data (around 88%). On the contrary, the Paper & Pen template and the 2007-09 PDA design showed poor performances (around 64%) on the speed grounds amongst the other templates.

Figure 4.18: Speed comparison: Torrey Pines at Spring Mountain

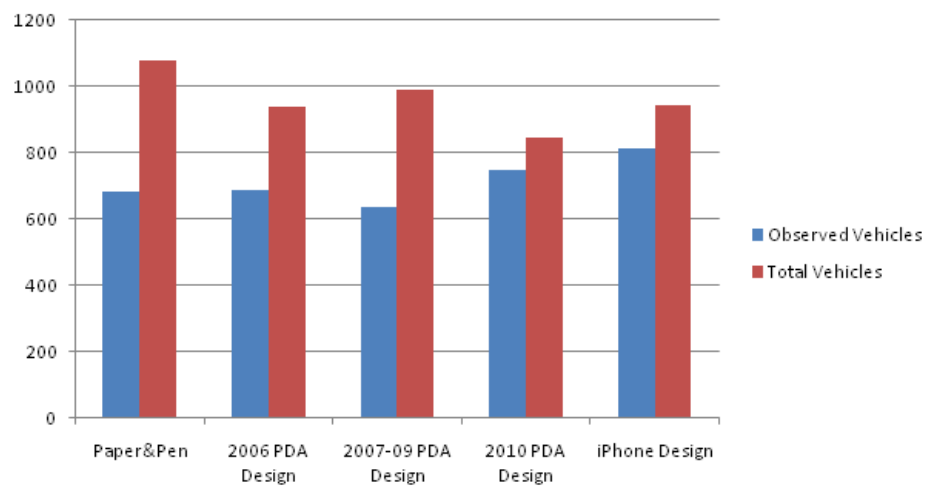


(a) Data collected in numbers

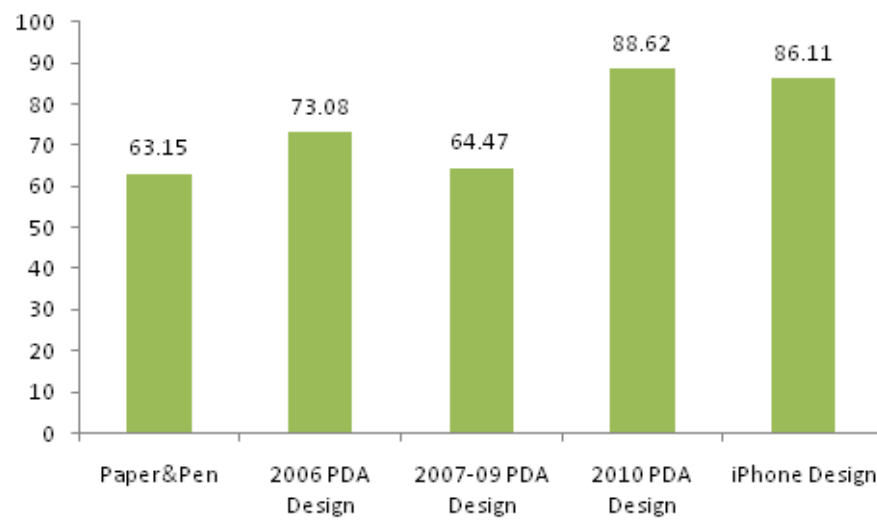


(b) Data collected in percentage

Figure 4.19: Speed comparison: Overall Results



(a) Data collected in numbers



(b) Data collected in percentage



### 4.2.3 Comparison 2: Accuracy

Before coming to any conclusions about which design is better and which is not, it is very important to have a statistical analysis comparing accuracy of different templates. In this section, we mainly focus upon conducting statistical analysis to see if the new designed templates (See Figures 4.5 and 4.11) and the older templates (See Figures 4.1, 4.3 and 4.4) have significant deviation. We conducted three different statistical tests to test our hypothesis that the data collected from all the designs is comparable.

#### 4.2.3.1 Chi-Square ( $\chi^2$ ) Analysis

It gives a measure of discrepancy between the observed and expected frequencies and is denoted by the statistic  $\chi^2$ . This statistic is given by the Equation 4.1, where  $j$  represent the number of categories within a variable. For example: If the variable considered is *Gender*, then this variable could be divided into *Male* and *Female* categories. Hence, in this case  $j = 2$ .

$$\chi^2 = \sum_j \frac{(O_j - E_j)^2}{E_j} \quad (4.1)$$

If the value of  $\chi^2 = 0$ , Equation 4.1 suggests that the observed and theoretical frequencies will agree exactly; otherwise if  $\chi^2 > 0$ , the above statement is not true. The larger the value of  $\chi^2$ , the greater the discrepancy between the observed and expected frequencies.

In practice, the expected frequencies are computed on the basis of a hypothesis  $H_0$ . If under this hypothesis, the computed value of  $\chi^2$  is greater than the critical

value, then it is concluded that the observed frequencies differ significantly from the expected frequencies and hence,  $H_0$  can be rejected at the corresponding level of significance; otherwise, it is not rejected. This procedure of hypothesis testing is called chi-square test of hypothesis.

In our study, we test the following hypothesis  $H_0$ :

$H_0$ : There is no significant difference between the data obtained from different templates for variables like safety belt status of the driver, driver characteristics, state of registration and vehicle type.

$H_1$ : There is significant difference between the data obtained from different templates for variables like safety belt status of the driver, driver characteristics, state of registration and vehicle type.

For our case, the expected data frequencies will be comprised of three templates: Paper & Pen (Fig 4.1), 2006 PDA design (Fig 4.3) and 2007-09 PDA design (Fig 4.4). On the other hand, the 2010 PDA design (Fig 4.5) and the iPhone design (Fig 4.11) will be considered as observed frequencies. Thus from this test, we compare the deviation in the data collected from the 2010 PDA design and iPhone design to that of the earlier techniques.

Tables 4.1, 4.2, 4.3 show the different  $\chi^2$  coefficients, comparing two designs at a time. The values of the  $\chi^2$  coefficient shows that the deviation in the data between the new (iPhone and 2010 PDA) and the older designs is not statistically significant. In other words, the computed value of came out to be less than the critical value and thus we could not reject the hypothesis.

Table 4.1: 2010 PDA Design vs. Paper & Pen : $\chi^2$  results

Variable	2010		Paper		$\chi^2$ parameters		
	%	N	%	N	$\chi^2$	<i>df</i>	Probability
Driver's Safety Belt Use							
Belted	91.3	683	93.1	635	1.6	1	0.206
Not belted	8.7	65	6.9	47			
Driver's Sex							
Male	73.3	548	71.4	487	0.614	1	0.433
Female	26.7	200	28.6	195			
Driver's Age group							
15-19	1.2	9	1.5	10	2.28	2	0.32
20-60	83.4	624	80.3	548			
60+	15.4	115	18.2	124			
Driver's Ethnicity							
African-American	5.2	39	6.9	47	10.2	3	0.002
Caucasian	62.7	469	67.7	462			
Hispanic	21.8	163	15.7	107			
Others	10.3	77	9.7	66			
Vehicle Type							
Pickup	19.6	146	17.5	119	3.6	2	0.165
<i>continued on next page</i>							

<i>continued from previous page</i>							
Variable	2010		Paper		$\chi^2$ parameters		
Sedan	51.7	387	56.7	387			
Van/SUV	28.7	215	25.8	176			
State							
NV	71.3	533	71.1	485	4.9	2	0.086
CA	18.7	140	21.7	148			
Others	10.0	75	7.2	49			

The value of  $\chi^2$  for all the variables such as safety belt status, driver characteristics, vehicle type and state of registration is less than the  $\chi^2_{crit}$  value. Thus, the hypothesis  $H_0$  cannot be rejected that the 2010 PDA design is comparable to the Paper & Pen template on the accuracy grounds.

Table 4.2: 2010 PDA design vs 2006 PDA's design:  $\chi^2$  results

Variable	2010		2006		$\chi^2$ parameters		
	%	N	%	N	$\chi^2$	<i>df</i>	Probability
Driver's Safety Belt Use							
Belted	91.3	683	95.3	655	9.24	1	0.002
Not belted	8.7	65	4.7	32			
Driver's Sex							
<i>continued on next page</i>							

continued from previous page							
Variable	2010		2006		$\chi^2$ parameters		
Male	73.3	548	73.8	507	0.053	1	0.818
Female	26.7	200	26.2	180			
Driver's Age group							
15-19	1.2	9	0	0	9.49	2	0.009
20-60	83.4	624	86.5	594			
60+	15.4	115	13.5	93			
Driver's Race							
African-American	5.2	39	7.1	49	2.48	3	0.479
Caucasian	62.7	469	62.4	429			
Hispanic	21.8	163	20.5	141			
Others	10.3	77	10.0	68			
Vehicle Type							
Pickup	19.6	146	17.3	119	1.55	2	0.461
Sedan	51.7	387	51.7	355			
Van/SUV	28.7	215	31	213			
State							
NV	71.3	533	70.2	482	2.32	2	0.314
CA	18.7	140	21.4	147			
Others	10.0	75	8.4	58			
continued on next page							

<i>continued from previous page</i>			
Variable	2010	2006	$\chi^2$ parameters

In Table 4.2, the value of  $\chi^2$  for all the aforementioned variables is also less than the  $\chi^2_{crit}$  value. Again, the hypothesis  $H_0$  cannot be rejected that the 2010 PDA design has comparable accuracy to the 2006 PDA design. In other words, there is a very little deviation between the data for all the safety belt variables obtained from these two designs.

Table 4.3: 2010 PDA design vs. 2007-09 PDA Design:  $\chi^2$  results

Variable	2010		2007-09		$\chi^2$ parameters		
	%	N	%	N	$\chi^2$	<i>df</i>	Probability
Driver's Safety Belt Use							
Belted	91.3	683	91.8	585	0.123	1	0.725
Not belted	8.7	65	8.2	52			
Driver's Sex							
Male	73.3	548	73	465	0.0122	1	0.912
Female	26.7	200	27	172			
Driver's Age group							
15-19	1.2	9	2	13	13.5	2	0.001
20-60	83.4	624	88.9	566			
<i>continued on next page</i>							

<i>continued from previous page</i>							
Variable	2010		2007-09		$\chi^2$ parameters		
60+	15.4	115	9.1	58			
Driver's Race							
African-American	5.2	39	8	51	9.52	3	0.002
Caucasian	62.7	469	56.1	357			
Hispanic	21.8	163	22.6	144			
Others	10.3	77	13.3	85			
Vehicle Type							
Pickup	19.6	146	15.2	97	4.81	2	0.09
Sedan	51.7	387	52.9	337			
Van/SUV	28.7	215	31.9	203			
State							
NV	71.3	533	67.8	432	12.8	2	0.002
CA	18.7	140	25.6	163			
Others	10.0	75	6.6	42			

Similar to Table 4.2, Table 4.3 also suggests that the  $H_0$  hypothesis cannot be rejected that the 2010 PDA design has comparable accuracy to the 2007-09 PDA design.

Table 4.4: iPhone design vs Paper & Pen:  $\chi^2$  results

Variable	iPhone		Paper		$\chi^2$ parameters		
	%	N	%	N	$\chi^2$	df	Probability
Driver's Safety Belt Use							
Belted	96.9	787	93.1	635	11.7	1	0.001
Not belted	3.1	25	6.9	47			
Driver's Sex							
Male	71.1	577	71.4	487	0.022	1	0.88
Female	26.7	200	26.2	180			
Driver's Age group							
15-19	1.6	13	1.5	10	1.12	2	0.57
20-60	82.3	668	80.3	548			
60+	16.1	131	18.2	124			
Driver's Race							
African-American	8.5	69	6.9	47	4.72	3	0.194
Caucasian	62.7	509	67.7	462			
Hispanic	18.7	152	15.7	107			
Others	10.1	82	9.7	66			
Vehicle Type							
Pickup	18.5	150	17.5	119	0.725	2	0.696
<i>continued on next page</i>							



<i>continued from previous page</i>							
Variable	iPhone		Paper		$\chi^2$ parameters		
Sedan	54.5	443	56.7	387			
Van/SUV	27	219	25.8	176			
State							
NV	73.2	594	71.1	485	4.87	2	0.088
CA	17.7	144	21.7	148			
Others	9.1	74	7.2	49			

In Table 4.4, it can be observed that the value of  $\chi^2$  for safety belt status is more than its  $\chi^2_{crit}$  value. For other variables such as driver characteristics, vehicle type and state of registration, the value of  $\chi^2$  comes out to be less than the  $\chi^2_{crit}$  value. As we see some deviation in the safety belt status, nothing could be concludes about the accuracy between the iPhone design and the Paper & Pen template based on  $\chi^2$ -test.

Table 4.5: iPhone design vs 2006 PDA's design:  $\chi^2$  results

Variable	iPhone		2006		$\chi^2$ parameters		
	%	N	%	N	$\chi^2$	<i>df</i>	Probability
Driver's Safety Belt Use							
Belted	96.9	787	95.3	655	2.54	1	0.111
Not belted	3.1	25	4.7	32			
<i>continued on next page</i>							

<i>continued from previous page</i>							
Variable	iPhone		2006		$\chi^2$ parameters		
Driver's Sex							
Male	71.1	577	73.8	507	1.4	1	0.237
Female	28.9	235	26.2	180			
Driver's Age group							
15-19	1.6	13	0	0	13.5	2	0.001
20-60	82.3	668	86.5	594			
60+	16.1	131	13.5	93			
Driver's Race							
African-American	8.5	69	7.1	49	1.52	3	0.678
Caucasian	62.7	509	62.4	429			
Hispanic	18.7	152	20.5	141			
Others	10.1	82	10.0	68			
Vehicle Type							
Pickup	18.5	150	17.3	119	2.96	2	0.82
Sedan	54.5	443	51.7	355			
Van/SUV	27	219	31	213			
State							
NV	73.2	594	70.2	482	3.23	2	0.199
CA	17.7	144	21.4	147			
<i>continued on next page</i>							

<i>continued from previous page</i>							
Variable	iPhone		2006		$\chi^2$ parameters		
Others	9.1	74	8.4	58			

Table 4.5 suggests that since the  $\chi^2$  for all the variables is less than the  $\chi^2_{crit}$  value, the hypothesis  $H_0$  cannot be rejected that the iPhone design is comparable in accuracy to the 2006 PDA design.

Table 4.6: iPhone design vs. 2007-09 PDA Design:  $\chi^2$  results

Variable	iPhone		2007-09		$\chi^2$ parameters		
	%	N	%	N	$\chi^2$	<i>df</i>	Probability
Driver's Safety Belt Use							
Belted	96.9	787	91.8	585	18.3	1	0.000
Not belted	3.1	25	8.2	52			
Driver's Sex							
Male	71.1	577	73	465	0.665	1	0.415
Female	28.9	235	27	172			
Driver's Age group							
15-19	1.6	13	2	13	15.3	2	0.000
20-60	82.3	668	88.9	566			
60+	16.1	131	9.1	58			
<i>continued on next page</i>							

<i>continued from previous page</i>							
Variable	iPhone		2007-09		$\chi^2$ parameters		
Driver's Race							
African-American	8.5	69	8	51	8.64	3	0.034
Caucasian	62.7	509	56.1	357			
Hispanic	18.7	152	22.6	144			
Others	10.1	82	13.3	85			
Vehicle Type							
Pickup	18.5	150	15.2	97	5.33	2	0.07
Sedan	54.5	443	52.9	337			
Van/SUV	27	219	31.9	203			
State							
NV	73.2	594	67.8	432	14.7	2	0.001
CA	17.7	144	25.6	163			
Others	9.1	74	6.6	42			

In Table 4.6, the  $\chi^2$  values for safety belt status, driver's age group and the state of registration came out to be more than the  $\chi^2_{crit}$  value. For other variables such as the driver's race, driver's ethnicity and vehicle type, the  $\chi^2$  values  $< \chi^2_{crit}$  value. Thus, nothing could be concluded about the accuracy comparison between the iPhone design and 2007-09 PDA design based on  $\chi^2$ -test.

Table 4.7: 2010 PDA design vs. iPhone Design:  $\chi^2$  results

Variable	2010		iPhone		$\chi^2$ parameters		
	%	N	%	N	$\chi^2$	df	Probability
Driver's Safety Belt Use							
Belted	91.3	683	96.9	787	22.5	1	0.000
Not belted	8.7	65	3.1	25			
Driver's Sex							
Male	73.3	548	71.1	577	0.94	1	0.332
Female	26.7	200	28.9	235			
Driver's Age group							
15-19	1.2	9	1.6	13	0.642	2	0.725
20-60	83.4	624	82.3	668			
60+	15.4	115	16.1	131			
Driver's Race							
African-American	5.2	39	8.5	69	7.9	3	0.048
Caucasian	62.7	469	62.7	509			
Hispanic	21.8	163	18.7	152			
Others	10.3	77	10.1	82			
Vehicle Type							
Pickup	19.6	146	18.5	150	1.25	2	0.536
<i>continued on next page</i>							

<i>continued from previous page</i>							
Variable	2010		iPhone		$\chi^2$ parameters		
Sedan	51.7	387	54.5	443			
Van/SUV	28.7	215	27	219			
State							
NV	71.3	533	73.2	594	12.8	2	0.002
CA	18.7	140	17.7	144			
Others	10.0	75	9.1	74			

In table 4.7, the value of  $\chi^2$  for safety belt status is observed to be more than the  $\chi^2_{crit}$  value. For other variables such as driver characteristics, vehicle type and state of registration, the value of  $\chi^2$  is less than the  $\chi^2_{crit}$  value. Due to the deviation in the safety belt status, nothing could be concluded about the accuracy comparison between the iPhone design and 2010 PDA design based on the  $\chi^2$ -test.

As a proper pattern could not be observed among the results of various designs for the  $\chi^2$ -test, the hypothesis  $H_0$  can neither be rejected, nor accepted for all the variables. Hence, there is a need to conduct more statistical analysis to comment about the accuracy relationships among the five designs.

#### 4.2.3.2 t-Test

The t-Test assesses whether the means of two groups are statistically different from each other. This analysis is appropriate whenever we want to compare the means of two groups.

Table 4.8 shows the t-Test analysis results for the five different data sets and on

each variable of the data set.

Table 4.8: Comparison for different PDA designs: t-Test results

Variable	Paper	2006	2007-09	2010	iPhone
Driver's Safety Belt Use					
t-value	1.1599	1.1027	1.1951	1.2104	1.0656
$df$	1	1	1	1	1
p-value	0.453	0.469	0.444	0.449	0.479
Driver's Sex					
t-value	2.3356	2.1009	2.1741	2.1494	2.3743
$df$	1	1	1	1	1
p-value	0.2575	0.2828	0.2745	0.2772	0.2538
Driver's Age group					
t-value	1.3889	1.3713	1.1975	1.3136	1.3428
$df$	2	1	2	2	2
p-value	0.2993	0.4011	0.3538	0.3195	0.3114
Driver's Race					
t-value	1.7403	1.9514	2.3194	1.9177	1.9592
$df$	3	3	3	3	3
p-value	0.1802	0.1461	0.1031	0.1510	0.145
<i>continued on next page</i>					

<i>continued from previous page</i>					
Variable	Paper	2006	2007-09	2010	iPhone
Vehicle Type					
t-value	2.789	3.3384	3.0578	3.4795	3.0605
<i>df</i>	2	2	2	2	2
p-value	0.1081	0.0792	0.0924	0.0736	0.0922
State					
t-value	1.7227	1.7741	1.8423	1.7427	1.6613
<i>df</i>	2	2	2	2	2
p-value	0.2271	0.2181	0.2068	0.2235	0.2385

From Table 4.8, it can be observed that the t-value is less than the critical value. As the t-test assesses the means of different designs for various variables, it can be observed that all the means are comparable to their  $t$  value. From all these tests, since the same results were obtained, therefore, the hypothesis  $H_0$  cannot be rejected, Thus, it could be concluded the 2010 PDA Design and the iPhone Design have comparable accuracy to the Paper & Pen template, 2006 PDA Design and 2007-09 PDA Design.

#### 4.2.3.3 Analysis of Variance

The main aim of analysis of variance (ANOVA) is to test significant differences among two or more groups. We define a hypothesis and then compute a value corresponding to that hypothesis. If the F-value is greater than the critical value, then we can reject the null hypothesis  $H_0$ ; otherwise we cannot reject the hypothesis.

In our case, data sets from different templates would act as different groups.



Tables 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, 4.16, 4.17, 4.18, 4.19 and 4.20 show the F-test analysis for the different designs.

Table 4.9: Driver's Safety Belt Use: For all designs

Variable	Paper	2006	2007-09	2010	iPhone
Belted	635	655	585	683	787
Not Belted	47	32	52	65	25

Table 4.10: F-Test Results for Safety Belt Use

Source of Variation	$df$	$\sum(.)^2$	$\overline{(.)}^2$	F-value
Between	4	9219.4	2304.8	$F = \frac{2304.8}{198052} = 0.012$
Within	5	990260	198052	

From Table 4.10, it can be observed that the value of test statistic for safety belt status among all the designs (i.e. the  $F$ -value) is less than the  $F_{crit}$ -value for  $df = (4, 5)$ . Thus, the hypothesis  $H_0$  cannot be rejected that the data for safety belt status was collected with comparable accuracy from all the designs.

Table 4.11: Driver's Sex: For all designs

Gender	Paper	2006	2007-09	2010	iPhone
Male	487	507	465	548	577
Female	195	180	172	200	235

Table 4.12: F-Test Results for Driver's Sex

Source of Variation	$df$	$\sum(.)^2$	$\overline{(.)}^2$	F-value
Between	4	9219.4	2304.8	$F = \frac{2304.8}{60234} = 0.038$
Within	5	301170	60234	

From Table 4.12, it can be observed that the  $F$  value for driver's gender among all the designs is less than the  $F_{crit}$ -value for  $df = (4, 5)$ . Thus, the hypothesis  $H_0$  cannot be rejected that the data for driver's gender was collected with comparable accuracy from all the designs.

Table 4.13: Driver's Age Group: For all designs

Age Group	Paper	2006	2007-09	2010	iPhone
15-19	10	0	13	9	13
20-60	548	594	566	624	668
60	124	93	58	115	131

Table 4.14: F-Test Results for Driver's Age

Source of Variation	$df$	$\sum(.)^2$	$\overline{(.)}^2$	F-value
Between	4	6151.8	1538	$F = \frac{1538}{101350} = 0.015$
Within	10	1013500	101350	

From Table 4.14, the  $F$ -value for driver's age comes out to be less than the  $F_{crit}$ -value for  $df = (4, 10)$ . Thus, the hypothesis  $H_0$  cannot be rejected that the data for driver's age was collected with comparable accuracy from all the designs.

Table 4.15: Driver's Ethnicity: For all designs

Ethnicity	Paper	2006	2007-09	2010	iPhone
African American	47	49	51	39	69
Caucasian	462	429	357	469	509
Hispanics	107	141	144	163	152
Others	66	68	85	77	82

Table 4.16: F-Test Results for Driver's Ethnicity

Source of Variation	$df$	$\sum(.)^2$	$\overline{(.)}^2$	F-value
Between	4	4609.7	1152.4	$F = \frac{1152.4}{33843} = 0.034$
Within	15	507640	33843	

For driver's ethnicity, Table 4.14 suggests that  $F$ -value  $< F_{crit}$ -value for  $df = (4, 15)$ . Thus, the hypothesis  $H_0$  cannot be rejected that the data for driver's ethnicity was collected with comparable accuracy from all the designs.

Table 4.17: Vehicle type: For all designs

Vehicle Type	Paper	2006	2007-09	2010	iPhone
Pick Up	119	119	97	146	150
Sedan	387	355	337	387	443
Van/SUV	176	213	203	215	219

Table 4.18: F-Test Results for Vehicle Type

Source of Variation	$df$	$\sum(.)^2$	$\overline{(.)}^2$	F-value
Between	4	6151.8	1538	$F = \frac{1538}{17476} = 0.088$
Within	10	174760	17476	

For vehicle type, Table 4.14 shows that  $F\text{-value} < F_{crit}\text{-value}$  for  $df = (4, 10)$ . Thus, the hypothesis  $H_0$  cannot be rejected that the data for vehicle type was collected with comparable accuracy from all the designs.

Table 4.19: State of Registration: All designs

State	Paper	2006	2007-09	2010	iPhone
NV	485	482	432	533	594
CA	148	147	163	140	144
Others	49	58	42	75	74

Table 4.20: F-Test Results for State of Registration

Source of Variation	$df$	$\sum(.)^2$	$\overline{(.)}^2$	F-value
Between	4	6151.8	1538	$F = \frac{1538}{57239} = 0.027$
Within	10	572390	57239	

From Table 4.14, the  $F$ -value for state of registration comes out to be less than the  $F_{crit}$ -value for  $df = (4, 10)$ . Thus, the hypothesis  $H_0$  cannot be rejected that the data for state of registration was collected with comparable accuracy from all the designs.

As the  $F$ -value is less than the  $F_{crit}$ -value for all the variables among different designs, hence the hypothesis  $H_0$  cannot be rejected. In other words, the data ob-

tained from various designs shows a very small variation. Thus, all the designs are comparable on the basis of accuracy.

Overall, it has been observed two out of three statistical tests suggest that the hypothesis  $H_0$  cannot be rejected. Therefore, it can be concluded that all the five data collection designs collect the data with comparable accuracy. With this, we finish the chapter on the data collection softwares for Daytime Safety Belt Usage Studies.

## CHAPTER 5

### DESIGNING FEEDBACK MOTIVATED TRAFFIC SAFETY CAMPAIGNS

#### Abstract

In September 2009, National Center for Statistics and Analysis (2009) published National Occupant Protection Use Survey, according to which safety belt usage was 84% in United States. This is a huge increase over a nationwide usage rate of 65% in 1994. Over the years, the safety belt usage rates have witnessed a constant increase. However, this progress in safety belt usage rates, and hence the success in saving far more lives, has been due to the intense efforts of the U.S. Government. Since 1994, U.S. government has been conducting 'Click It Or Ticket'(CIOT) Mobilization which includes aggrandized law enforcement and paid media adveristing. Every year, the U.S. government spends millions of dollars on conducting these campaigns to increase public awareness about the use of safety belts. The history itself speaks of the success of CIOT mobilization, however it comes at a huge price. Hence, it is imperative that instead of just crunching numbers, a conceptual framework is developed which looks at the details of how the campaigns affect people. Furthermore, this framework should have the capabilities to efficiently measure the people's behaviour towards safety belts and thrust the campaigns in the coming years. In this chapter, a conceptual feedback framework has been presented which fulfils the aforementioned tasks. Additionally, a model motivated from the theories behind the diffusion of diseases in humans has been presented which aims to capture the behavior of how campaign information diffuses among the people.

## 5.1 History and success of CIOT Mobilization

Despite the documented use of safety belts, nearly one in five Americans don't regularly wear a safety belt while driving or riding in a motor vehicle (Traffic Safety Marketing). To combat this situation and help people realize the importance of safety belts, 'Click It or Ticket' mobilization was developed to reduce highway fatalities due to the non-use of safety belts. This mobilization is conducted on an annual basis by National Highway Traffic Safety Administration (NHTSA) in affiliation with several law enforcement agencies, traffic safety advocates and State Highway Safety Offices. Traffic Safety Marketing also provides invaluable information about the history of CIOT mobilization. Here are a few important details of CIOT mobilization:

### 1. *1993:*

The CIOT mobilization was first created in North Carolina in 1993. It combined 3000 enforcement checkpoints, paid advertising and earned media during which 58,000 citations were issued for safety belt violations. This resulted in a huge increase of safety belt usage rate from 65 % to 84 % by July 1994.

### 2. *2000:*

In November 2000, South Carolina joined the cause and conducted a campaign featured by 2-week increased law enforcement, earned media and paid advertising supported by \$500,000 grant from National Safety Councils Air Bag & Seat Belt Safety Campaign. This resulted in hiking the safety belt usage rate from 65% to 79%.

### 3. *2001:*

In May 2001, Alabama, Florida, Georgia, Kentucky, Mississippi, North Carolina, South Carolina and Tennessee joined hands and collectively participated in 5-week earned media campaign, a \$ 3.6 million 2-week paid advertising campaign and 2 weeks of intensive law enforcement. This resulted in a huge increase in safety belt usage rate in every state.

4. *2002:*

In May 2002, eighteen more states including Nevada conducted nationwide CIOT campaign and was supported by Federal grants and \$ 10 million received for paid advertising. This resulted in approximately 250,000 safety belt use citations during the enforcement period in 18 states.

5. *2003:*

In 2003, 43 states, District of Columbia and Puerto Rico participated in nationwide CIOT mobilization, thereby using \$ 8 million from Federal grants for advertising campaigns. This resulted in an overall increase from 75% to 79% in 2003.

6. *2004-Present:*

From 2004, the nationwide safety belt usage rates have surpassed the 80% line, and have been motivated by rural programs focusing on safety belt usage in pickup trucks and \$ 8 million Federal advertising media buy every year.

Hence, it is evident from the historical facts that CIOT mobilization has been influential in maintaining high safety belt usage rates. However, it cannot also be ignored



that the CIOT mobilization has been constantly backed by Federal and other grants by safety offices and agencies. These grants fulfil a major chunk of the advertising media buying requirements. Hence, it is important to understand these advertising channels and implement a control on them.

## 5.2 Media Advertising

In this section, the advertising channels supporting the CIOT mobilization have been explored. Figure 5.1 shows that 70% of 341 Nevadans in 2010 actually learnt about the Safety Belt Law Enforcement through television. This figure also reveals that only 3.9% of those Nevadans heard about the the campaign from the Police Enforcement. These numbers suggest that television is a good source to inform the people about the importance of safety belts.

During the CIOT mobilization, special messages are broadcasted at different times on several tv stations. A few of them (courtesy: Television Monitoring Services, 2003) are shown below:

### 1. *KTNV-ABC LAS VEGAS, NV* - JUN 27 2010 6:00PM

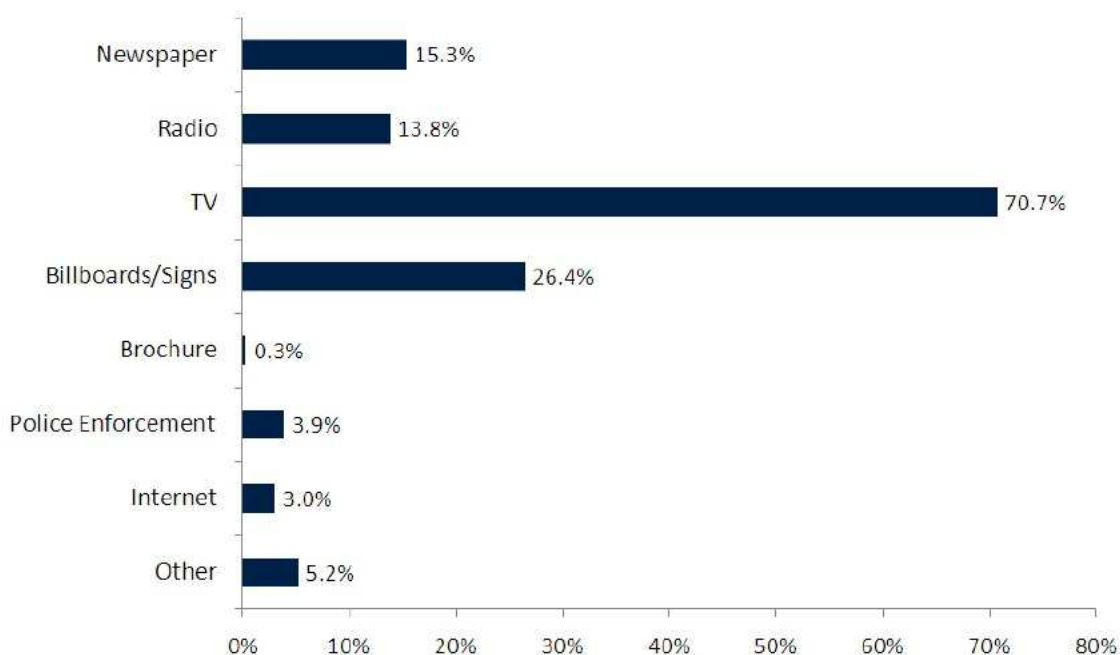
It's part of a special sting to find out offenders. Not just drunk drivers, but authorities, your seatbelt on. North Las Vegas Police will be running a 'Click It Or Ticket' campaign today from 11a.m. to 9 p.m and be warned. There will be a zero tolerance policy in place. Metro is making more arrests, weekend.

#### **Statistics:**

Run time: 0:26, Nielsen Audience: 19,522, Calculated Ad Equivalency: \$433,

Calculated Publicity Value: \$1,299, 30-Second Ad Equivalency: \$500

Figure 5.1: Where Nevadans read, saw or heard about Seat Belt Law Enforcement (n = 341)



## 2. KVBC-NBC LAS VEGAS, NV - JUN 27 2010 6:00AM

Here at home North Las Vegas police officers are trying to keep our roads safe, officers will be saturating the valley today, we're told they will be out from about 11 this morning until 9 p.m. This is part of the Click It Or Ticket Speeding campaign, officers say they will have zero tolerance for anyone driving recklessly.

### Statistics:

Run Time: 0:20, Nielsen Audience: 15,539, Calculated Ad Equivalency: \$267, Calculated Publicity Value: \$801, 30-Second Ad Equivalency: 4\$00

## 3. KOLO-ABC RENO, NV - JUN 10 2010:6:00AM

As the day progresses, Sparks police say, they stopped nearly two-hundred-and-eighty vehicles during the recent 'Click It Or Ticket' Campaign. The stops

were made between May twenty-fourth and June sixth. Almost one hundred citations were given out for seatbelt and car-seat violations.

**Statistics:**

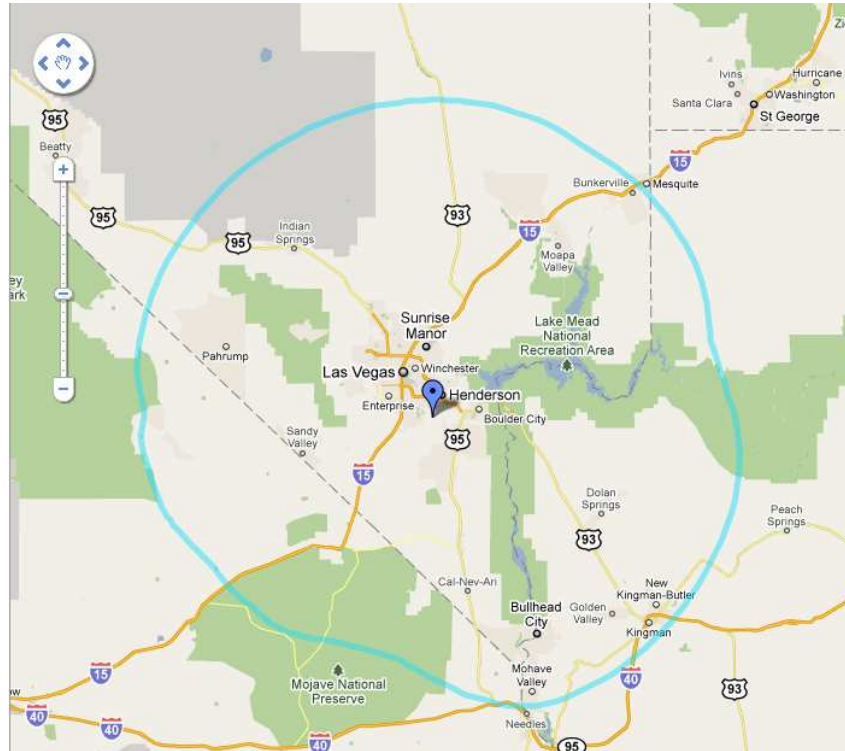
Run Time: 0:19, Nielsen Audience: 5,662, Calculated Ad Equivalency: \$89,  
Calculated Publicity Value: \$267, 30-Second Ad Equivalency: \$140

The statistics mentioned above use data from Nielson Media Research and help the campaign designers to quantify the performance of a particular message and a tv channel. Other TV channels used for advertising under CIOT mobilization include: KLAS-CBS, KBLR-TELEMUNDO, KVVU-FOX, KINC-UNIVISION and KTVN-CBS. For campaign design, although Nielson and other ratings are specifically important, it is also crucial to know their coverage areas. Figure 5.2 shows the coverage area of KLAS-TV (Las Vegas) and KTNV-TV (Las Vegas). These graphs have been produced by doing a query on Federal Communications Commission (2010).

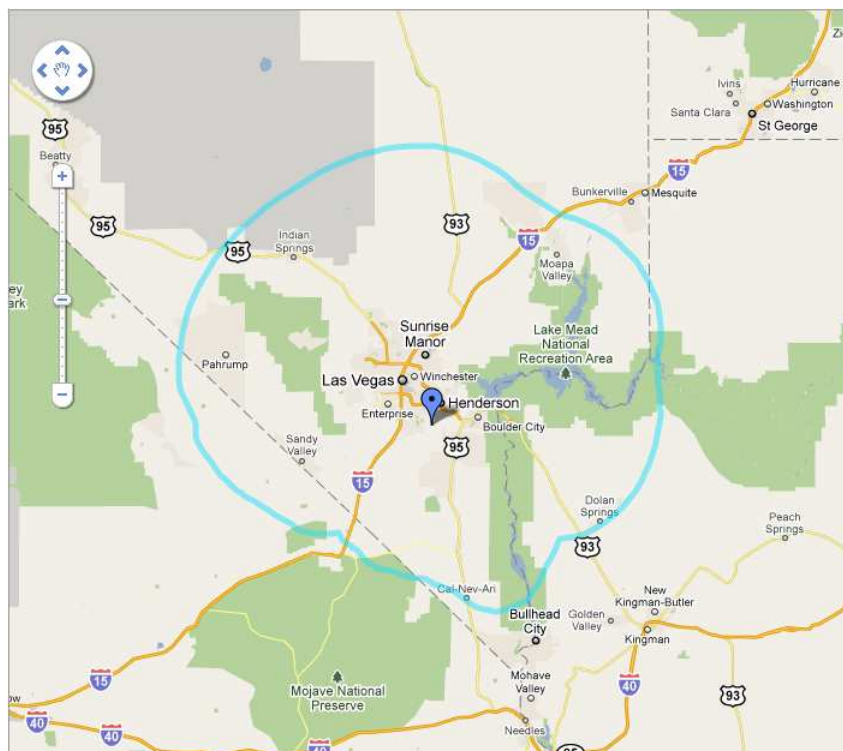
### 5.3 Feedback Framework for Campaign Design

Section 5.1 speaks of the indisputable success of CIOT mobilization in terms of increasing the safety belt usage. However, it can be seen that every year, millions of dollars are spent in augmenting the law enforcement and advertising messages through several media channels to influence the safety behavior of the people. Moreover, the time period of CIOT mobilization is also fixed (for example: May in Nevada). What would happen if the mobilization was launched during some other time ? Also, what television messages and what television channels will work the best for spreading

Figure 5.2: Coverage Areas of Televisions

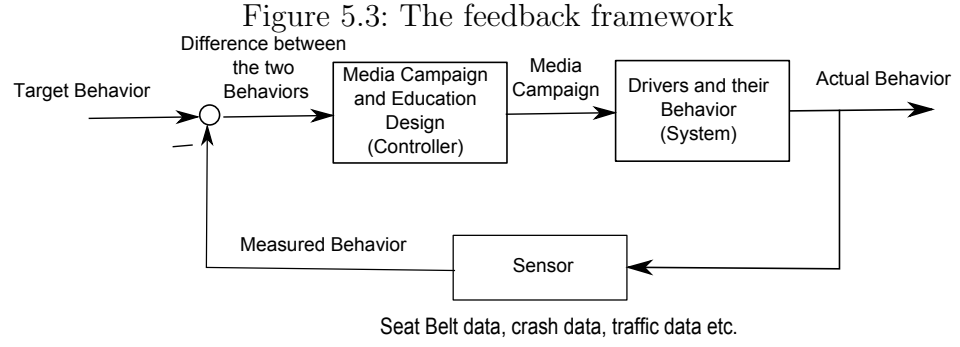


(a) KLAS-TV, Las Vegas



(b) KTNV-TV, Las Vegas

the mobilization information around ? Above all, how are the people influenced by the media advertisements assumed to positively affect the safety behavior ? To answer all these questions effectively, a conceptual framework is required which takes into account the current behavior of people towards safety belt usage through efficiently designed sensors, matches it against a target behavior, studies the spread of CIOT mobilization information through people and provides a feedback to control the campaigns in the near future. This feedback framework, which aims to reduce the costs incurred during the campaign, while simultaneously reaching a target behavior amongst the people is shown in Figure 5.3.



Thus, it is important to know how different people receive the campaign information and how they respond to it. For this purpose, a conceptual SEICRM model has been proposed in the next section, which aims to model the spread of campaign information among the people.

#### 5.4 Conceptual framework for SEICRM model

In this section, a conceptual model has been proposed to capture the dynamics of the spread of the information related to the campaigns through the people. The

spread of this information is quite analogous to the spread of an infection among the humans. Thus, we borrow a few ideas from the deterministic epidemiological models of infections in humans and apply it in our case. Firstly, the terminology required in the proposed model has been discussed. Then, the SEIRCM model is presented in the form of an equation. Then, the meaning of the parameters in the model equation are discussed followed by an in-depth analysis of the equation.

#### 5.4.1 SEIRCM categories

This section defines six categories to be used in our model for studying the campaign information diffusion. These categories stand for: Susceptible(S), Exposed(E), Infected(I), Carrier(C), Recovered(R) and Totally Immune(M).

Consider an individual who has not yet been exposed to a particular CIOT message, either broadcasted through television or through law enforcement agencies. Now, this individual can either come across that message and get influenced from it (thus contract an infection) or choose to totally ignore it. Thus, that particular individual can go either way, which cannot be determined until he has actually experienced the CIOT message. Such an individual will fall into a category called *Susceptible (S)*, who have not yet been exposed to the CIOT message (yet unexposed to the pathogen). Another category, called *Infected (I)*, comprises of individuals who have seen that CIOT message, wear the safety belts and spread the word around (currently infected by the pathogen). The third category of individuals are those who were previously infected with the campaign information but have stopped spreading the campaign information. This scenario might arise in several situations. One such

situation is when driving experience is involved. For example, as the new drivers, aware of the campaign information, gain experience or drive in the zones where the speed limits are low, they become confident and stop using safety belts followed by a change in behavior towards preaching about the safety belt use. This category of individuals, called *Recovered*(R), won't be affected anymore by the inoculation of campaign information (infection). These three categories namely *Susceptible* (S), *Infected* (I) and *Recovered* (R) form the basic modeling technique in Epidemiology and hence, will be used as the pedestal to build the model for campaign information diffusion.

The basic SIR model for campaign information diffusion can be further refined by adding the *Exposed* (E), *Carrier* (C) and *Totally Immune* (M) categories. Consider that a CIOT campaign is launched with broadcasting 'Buckle Up' message on the television. At the same time, the law enforcement agencies also increase their activities, giving safety belt citations and telling the people to buckle up. However, during the initial stages of this campaign, there is a possibility that an individual who comes across these CIOT messages, either through television or through law enforcement agencies, himself gets influenced but is not yet capable of spreading the campaign related information locally. For him, watching a television message or getting a safety belt citation might just be one time coincidence. This might occur in a case, specifically during an initial stage, when he has not come across the CIOT messages so much that he can spread the word around. In such a scenario, although the individual is infected by the campaign information, he is not yet infectious to let others know about the campaign (Keeling & Rohani, 2007). Such an individual

will be said to belong to the *Exposed (E)* category. Secondly, since modeling the campaign information flow is a complex phenomena, there is a requirement to take into account the time factor as well. It is a general tendency in humans to forget incidents with time. As the time passes by, the newer information overcomes the past one. Consider an example of an individual who saw the CIOT messages and became infected for a short period of time. As he comes across newer information unrelated to the campaign, the rate at which he spreads the campaign related information decreases and becomes more and more interested in the contemporary events of life. Such an individual still has the campaign information in mind, staying infected for a longer time period, but the rate of transmission has subdued. This individual will be called a chronic carrier of the campaign information and is said to belong to the *Carrier* category. In epidemiological models with carrier categories, it is assumed that a susceptible individual can be infected either by an individual in *Infected* category or *Carrier* category, without any distinction (Keeling & Rohani, 2007). Thus, the same shall be true for the models associated with studying the campaign information diffusion. Lastly, a *Totally Immune (M)* category has been introduced which comprises of individuals whose behavior towards the safety belts is highly negatively biased. These individuals seldom have any effect from the campaign information and thus, almost never tend to wear safety belts, primarily because of certain myths or rebellious behaviour. However, certain probability of them becoming susceptible still exists, in case an external event like a crash happens. However, this probability is very low on an average basis.



### 5.4.2 Proposed SEICRM Model

The proposed model for studying campaign information diffusion is shown in Equation 5.1. The transmission from one category into another has been shown in Figure 5.4.

$$\begin{aligned}
\frac{\partial X}{\partial t} &= \nu + \omega Z + \Psi M - \frac{\beta X}{N}(Y + \epsilon C) - \mu X + D_X \nabla^2 X \\
\frac{\partial E}{\partial t} &= \frac{\beta X}{N}(Y + \epsilon C) + \alpha(x, y, t)u - \sigma E - \mu E + D_E \nabla^2 E \\
\frac{\partial Y}{\partial t} &= \sigma E - (\mu + \sigma)Y + D_E \nabla^2 Y \\
\frac{\partial C}{\partial t} &= \gamma q Y - (\mu + \Gamma)C + D_C \nabla^2 C \\
\frac{\partial Z}{\partial t} &= \gamma(1 - q)Y + \Gamma C - (\mu + \omega + \phi)Z + D_Z \nabla^2 Z \\
\frac{\partial M}{\partial t} &= \nu - \mu M + \phi Z - \Psi M
\end{aligned} \tag{5.1}$$

### 5.4.3 Explanation of the SEICRM model

People gain information about the campaigns either directly through televisions and elevated law enforcement, or through interaction with each other. Assuming there are no telephonic conversations, the campaign information diffuses within the population either through the interactions within the same location or through the geographical spread of the individuals. The spatial models in Epidemiology well capture the host population characteristics and quantify the rate of spread of a pathogen

within the population. Therefore, an analogy can again be drawn from the spatial Epidemiology models to study campaign information diffusion.

#### 5.4.3.1 General information on Spatial Models

Several spatial models have been explored in Epidemiological studies. A few of them include Coupled Lattice Models, Cellular Automata Models, Continuous-space-continuous-population models implemented through partial differential equations etc. The details of these models have been provided by Keeling & Rohani (2007). In the coupled lattice models, the population is divided into grid sites and the individuals within each grid site are grouped together to form a sub-population. Only the neighbouring grid sites can interact with each other, leading to the spatial spread of the infection. This results in a wave like spread of the infection, where the infection has to get to the neighbours before spreading to the rest of the population (Keeling & Rohani, 2007). In cellular automata models, this situation is further simplified to allow only one individual at each grid site. Thus, either every grid site can be empty or have a single individual. In both these models, the interactions are so limited that they don't allow the populations to mix randomly. However, in practicality, the people can't be fixed at lattice sites and therefore, the space cannot be discretized. Therefore, continuous-space-continuous-populations tend to better model the random interactions within the continuous population, where the density of individuals is specified at all locations.

#### 5.4.3.2 Parameter Definitions in Equation 5.1

Continuous-space-continuous-population models are well implemented through partial differential equations, specifically called as reaction diffusion equations. The *SEICRM* model proposed here is a modified version of a reaction diffusion model with six categories. In the proposed SEICRM model, it is assumed that the infectious (I) or the carrier individuals (C) can transmit the campaign information only to susceptibles (S) at their current location and that all individuals can move randomly throughout the state.

In Equation 5.1, the variables  $X$ ,  $E$ ,  $Y$ ,  $C$ ,  $Z$  and  $M$  are all functions of both space (x,y) and time (t). These six variables represent the local densities of susceptible, exposed, infected, carrier, recovered and totally immune individuals and the total population  $N$  can be written as  $N = X + E + Y + C + Z + M$ . Since  $X$ ,  $E$ ,  $Y$ ,  $C$ ,  $Z$  and  $M$  are multidimensional variables, the rate of change of individuals is therefore calculated by finding the partial derivatives, as shown in Equation 5.1.

Geographically, Nevada is the seventh largest state with 110,540 square miles. In terms of population, Nevada was ranked 35th largest state (2000 census data). According to US census bureau (2005), Nevada was the fastest-growing state between April 1, 2000 and July 1, 2005. With a 21 percent growth rate, total population of Nevada had reached 2.4 million. Moreover, where the nation's population grew only by 5 percent between the given period, the population in North Las Vegas, Nevada (53 percent) grew ten times faster. Thus, in our model representing the diffusion of campaign information, it is essential to model the demography as well. Since the population size ( $N$ ) is not fixed, a fixed birth rate  $\nu$  independent of population size

is incorporated into the rate of change of susceptibles in Equation 5.1. Also, the individuals can suffer natural mortality in any category and is represented by  $\mu$ .

The term  $\beta$  represents the transmission rate and is known to be the product of transmission probability and the contact rates. The rate at which the new exposed category people are produced (which are infected but not yet infectious) is represented by  $\frac{\beta XY}{N}$ . The transmission of campaign information between individuals is frequency dependent, the details of which are provided are provided by Keeling & Rohani (2007). In our SEICRM model,  $\beta$  is a function of the control variable  $u$  defined later and time  $t$ , to reflect the changed behavior in transmission of campaign information during the campaign periods (specifically May every year).

The term  $\sigma$  represents the time delay introduced to slow down the transmission of an individual from the *Susceptible* to the *Infected* category. In other words, the term  $\frac{1}{\sigma}$  signifies the average duration of an individual staying in the *Exposed* category.

Due to the incorporation of the *Carrier* category, the variables  $\epsilon$ ,  $q$  and  $\Gamma$  have been introduced.  $\epsilon$  represents the reduced rate at which the chronic individuals belonging to the *Carrier* category transmit the campaign information to other susceptibles. Next,  $q$  represents that fraction of the infected individuals who join the *Carrier* category and transmit the campaign information for longer time periods at reduced rates. Therefore,  $(1 - q)$  fraction of the infected individuals recover and stop using the safety belts for a while. Furthermore, the term  $\Gamma$  represents the rate at which the individuals leave the *Carrier* category and stop using safety belts for a while, before again becoming susceptible to the campaign information at a rate  $\omega$ .

In our model, the recovered individuals might also become *Totally Immune* at a

rate  $\phi$ . As mentioned before, the *Totally Immune* individuals will not be affected by the campaign information unless an external event like a crash happens. In such a case, a *Totally Immune* individual might also become susceptible at a rate of  $\Psi$ , which otherwise would remain very low.

The term  $\nabla^2 X$  (and similarly other  $\nabla^2(s)$ ) in Equation 5.1 models the local spatial diffusion of individuals and represents the change in the rate of change of  $X$  individuals. Since our space in the campaign information diffusion study is two dimensional, the diffusion terms for susceptibles can be written as:

$$\nabla^2 X = \frac{\partial^2 X}{\partial x^2} + \frac{\partial^2 X}{\partial y^2} \quad (5.2)$$

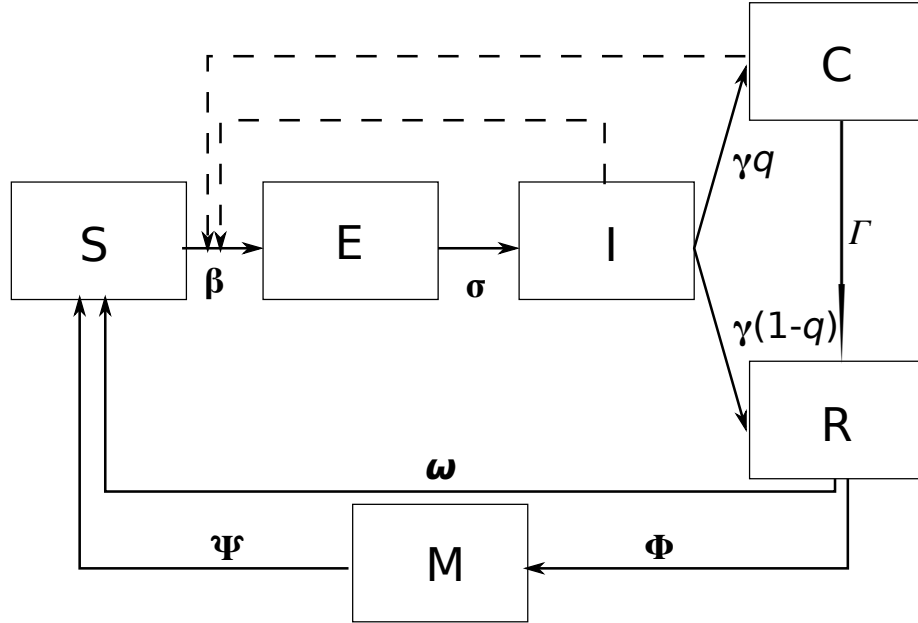
In general, the variables  $D_X$ ,  $D_E$ ,  $D_Y$ ,  $D_C$  and  $D_Z$  have been kept different for each category, representing different diffusion rates of individuals. However, they could also have been assumed to be same since all the individuals can be assumed to travel and spread the campaign information at same rates.

Last but not the least,  $u$  is a digital control variable with only two states: 0 and 1. The value of this control variable is 1 during the campaign period and stays zero otherwise.  $u$  is used in conjunction with  $\alpha(x, y, t)$  which represents the spatial coverage areas of the television channels as shown in Figure 5.2. These different television channels act as several patches on the surface of Nevada and directly influence the number of infected (infact exposed) individuals during the campaign periods.

#### 5.4.3.3 Further analysis of Equation 5.1

In section 5.4.3.2, theory behind partial differential equations, diffusion term and other parameters in Equation 5.1 has been discussed. In this section, the transmission of the campaign information between the six categories has been discussed. Figure 5.4 provides a flow diagram to present the conceptual description presented in Equation 5.1. In this figure, solid arrows have been used to show the movement of campaign information between the categories. According to Keeling & Rohani (2007), the progression from  $S$  to  $E$  category is determined by three factors: the numbers of infecteds, the population structure and the probability of transmission given contact. Since the level of individuals infected with the campaign information influences the rate at which the susceptibles move to the  $E$  category, this phenomenon is represented by the dotted arrows in Figure 5.4.

Figure 5.4: Proposed Model for Diffusion of Campaign Information



Keeling & Rohani (2007) also provides the explanation of the transmission term from *Susceptible* to *Exposed* category (actually Infected, *Exposed* category just represents a time delay). Let a susceptible individual have  $\kappa$  contacts per unit time. Out of these contacts, let  $I = \frac{Y}{N}$  be the contacts with infected individuals having the campaign information. During a time interval  $\delta t$ , the number of contacts of a susceptible individual with infecteds is  $\kappa \frac{Y}{N} \delta t$ . Let  $c$  be the probability that after a contact, a susceptible turns into an infected individual with campaign information. Then,  $1 - c$  represents the probability that the susceptible individual ignores the campaign information and hence does not participate in further spreading the information, thus not becoming infected. Since all the contacts were independent, the probability  $(1 - \delta q)$  that a susceptible individual ignores the campaign information inspite of all its contacts with  $\kappa \frac{Y}{N} \delta t$  infected individuals can be written as:

$$1 - \delta q = (1 - c)^{(\frac{\kappa Y}{N})\delta t} \quad (5.3)$$

Now, if a parameter  $\beta$  is written as  $\beta = -\kappa \log(1 - c)$  and substituted in Equation 5.3, then the probability of transmission of campaign information in time  $\delta t$  can be written as shown in Equation 5.4.

$$\delta q = 1 - e^{-\frac{\beta Y \delta t}{N}} \quad (5.4)$$

Using taylor expansion of  $e^x$ , dividing both sides of Equation 5.4 by  $\delta t$  and taking the limit  $\frac{\delta q}{\delta t}$  as  $\delta t$  goes to zero, the transmission rate per susceptible individual is given

by Equation 5.5.

$$\frac{dq}{dt} = \frac{\beta Y}{N} \quad (5.5)$$

Equation 5.5 represents the per capita probability of acquiring the infection and thus spreading the campaign information. Thus, the total rate of transmission to the entire susceptible population is given by Equation 5.6.

$$\frac{dX}{dt} = -\frac{dq}{dt}X = -\frac{\beta XY}{N} \quad (5.6)$$

In our SEICRM model, since  $X$  is a multidimensional variable,  $\frac{d}{dt}$  operator can be replaced by  $\frac{\partial}{\partial t}$ .

From Figure 5.4, it can be observed that the *Susceptible* population increases by the natural birth, the influx of individuals from the *Recovered* category at a rate  $\omega$  and the arrival of individuals from the *Totally Immune* category at a small rate  $\Psi$ . However, the *Susceptible* individuals move to the *Exposed* category after coming into contact with *Infected* or *Carrier* categories at the rate of  $\frac{\beta Y}{N}$  and  $\frac{\epsilon \beta C}{N}$  respectively. Some of the susceptible individuals might also die due to natural reasons at a rate  $\mu$ . The population of *Susceptible* is also influenced by the local diffusion of susceptible population through space.

Figure 5.4 also suggests that a time delay is introduced by the *Exposed* category between the transfer of individuals from the *Susceptible* to *Infected* category. The *Exposed* population is increased due to the transfer of individuals from the *Susceptible* category mentioned above. Also, the people under the patch (coverage area) of a



television station broadcasting campaign message come under the *Exposed* category, out of which a few individuals might eventually transfer into the *Infected* category at a rate of  $\sigma$ . Similarly, the natural death and diffusion terms affect the *Exposed* population as shown in Equation 5.1.

The *Infected* population, coming from the *Exposed* category after certain time period, are also influenced by natural death and local diffusion of the infected individuals. Furthermore, a  $q$  proportion of infected individuals might go into the *Carrier* category at a rate of  $\gamma$ , thereby transmitting the campaign information at a low rate for long time periods. On the other hand,  $(1 - q)$  proportion of those infected individuals choose to go into the *Recovered* category, where they refuse to further spread the campaign information.

The individuals in *Carrier* category stay chronically infectious for long time periods but either eventually move to the *Recovered* category or yield to the natural mortality at a rate of  $\mu$ . The diffusion term also affects the *Carrier* population in a way represented in Equation 5.1.

According to Figure 5.4, the recovered individuals cannot stay immune to the campaign forever and eventually move into the *Susceptible* category at a rate of  $\omega$ . However, these *Recovered* individuals might also choose to become *Totally Immune* at a rate of  $\phi$ , following their rebellious behavior. The *Recovered* population also succumbs to its natural death at a death rate  $\mu$  and is affected by the local diffusion of the recovered individuals.

Lastly, the *Totally Immune* population increases at a rate of  $\phi$  from the *Recovered* population but might also become *Susceptible* at a rate of  $\Psi$ , following a serious event

that might force them to rethink the necessity of the safety belt usage. Since this category does not participate in the dynamics of the campaign information diffusion, it is naturally influenced by the demographic processes and the diffusion term is ignored.

## CHAPTER 6

### IPHONE APPLICATION FOR DAYTIME SAFETY BELT USAGE STUDIES

#### Abstract

In Chapter 4, a single data collection design was implemented both on PDAs as well as on iPhone4. While developing a C# application on PDA is easy and requires a basic understanding of just the involved framework and the run flow, the development of an iPhone application for the first time can be quite challenging. This is because the attractive iPhone multitouch interface comes with a large pool of frameworks and memory management concepts which can be tough for a new developer to handle. Therefore, in order to comprehend the involved iPhone concepts in building interesting applications and to stay in the race of swiftly emerging smartphone market, a detailed account of the iPhone application built for the Daytime Safety Belt Usage Studies has been provided. Figure 6.1 shows a screenshot of the final iPhone application.

#### 6.1 Overview and Requirements

iOS application development has witnessed a tremendous growth in the market. The iOS SDK (software development kit) can be used to create applications on iPhones, Ipod Touch and Ipads. At the time of writing this thesis, iOS 4.3 has hit the market and things have gotten better and better with every release. For iOS development, the developers need to have an Intel-based Macintosh running Snow Leopard OS X. The most recent and popular IDE for iOS development provided by Apple is XCode 3.2.6 which requires MAC OS X 10.6.6 or later, although Apple is

Figure 6.1: Screenshot of the iPhone application



(a) First page

(b) Second page

also planning to release XCode 4 soon. Apple allows the developers to sign up and download 4.1 GB bundle of iOS SDK and Xcode 3.2.6 from the iOS Dev Center (2011b).

The iOS SDK includes a simulator that allows the developers to build and run the iPhone and iPad apps on Mac. However, hardware dependent features like iPhone's accelerometer or camera are not supported in the iOS simulator. Moreover, the developers need to sign up for Standard (\$99/year) or Enterprise (\$299/year) programs to deploy the applications on the actual iOS devices (like Iphone) or distribute them on the Apple's App Store. More information about the two programs can be found at Apple Developer Programs website (Apple, 2011a).

## 6.2 Model-View-Controller Paradigm

The Model-View-Controller paradigm is an architectural pattern used in software engineering. According to this pattern, the functionality of an application is divided into three parts:

1. Model:

Classes holding the application's data

2. View:

Renders the model into a form (consisting of windows, controls etc.) appropriate for user interaction. Multiple views can exist for a single model for several purposes.

3. Controller:

The application logic which decides how to handle user inputs and binds the model and view together.

According to the MVC pattern, any object created must be identifiable to belong to one category with a null intersection with the other two categories. For example: An object that implements a button must not contain the code to process the user interaction with the button.

## 6.3 Differences between iOS and Mac OS X development

There are a few key differences between iOS and Mac OS X development. This section covers these differences and presents them in an enumerated format.

1. One active application:

At a given time, only one application can be active in iOS. With the release of iOS 4, background processing is allowed but this is also restricted to a few usages. On the contrary, several applications can run on Mac OS X.

2. One window:

iOS applications are given only one window to work with. However, Mac OS X contains the ability to create and control multiple windows.

3. Restricted access:

iOS applications are allowed to access resources only in a specific area called an application's *sandbox*. This is where an iOS application stores documents, preferences and other data that needs to be stored. On the other hand, Mac OS X allows the developers to access a lot more resources.

4. Response time:

iOS developers need to make sure that application is launched, preferences are set, data is loaded and the main view is shown within a few seconds. Similarly, the data must be saved within 5 seconds from the point iOS home button is tapped. If these restrictions are not followed, the application process is automatically killed. This restriction is not there on Mac OS X.

5. System Resources:

On Mac OS X, chunks of memory not in use can be swapped to be written to a disk. This allows Mac OS X developers to ask for extra memory for an application than what is actually available. The memory swapping is not

available in iOS and hence, the iOS developers have to restrict their applications within the constrained system resources.

#### 6.4 Cocoa and Cocoa Touch Frameworks

Cocoa and Cocoa Touch frameworks are used for Mac OS and iOS development respectively. Both of them are implemented primarily in Objective C language. The Cocoa framework consists of libraries, APIs and runtimes, thereby forming the development layer of Mac OS X. Using Cocoa enables the developers to inherit behaviors and appearances of Mac OS X. Cocoa is known to be primarily a combination of AppKit and Core Foundation frameworks which cover a wide ground of applications on Mac related to networking, graphics, audio processing etc. A few frameworks available in Cocoa are mentioned below:

1. Audio and Video:

Core Audio, Core Midi, Core Video.

2. Data Management:

Core Data.

3. Networking and Internet:

Bonjour, Directory Services, Kerberos.

4. Graphics and Animation:

Core Animation, Core Image, OpenGL, Quartz, Quicktime, QTKit.

5. Cocoa Bridges to Scripting Languages:

AppleScript, Python, Ruby.

6. User applications:

Address Book, Calendar Store, Instant Message.

Cocoa Touch framework shares the patterns used on Mac but is built mainly with a focus on touch-based interfaces. It is known to be a combination of UIKit and Foundation frameworks, where UIKit provides the tools to provide graphical capabilities. UIKit is built upon Foundation framework in Mac OS X, which includes file handling, networking, string building etc. A few frameworks and their applications in Cocoa Touch can be seen below:

1. Core Animation:

Examples and APIs: OpenGL ES API for 3D animations, Quartz 2D API, Photo Library etc.

2. Core Audio:

Examples and APIs: Media Player, HTTP Live Streaming, AVFoundation.

3. Core Data:

Examples and APIs: Sqlite, XML files, Calendar Access, Sharing data amongs apps.

In 2006, Apple announced the release of Objective C 2.0, a revision to Objective C to include the concepts of Garbage Collection, Properties and Fast Enumeration. The details about these new concepts can be found in M. Dalrymple (2009).



## 6.5 Building the Iphone Application

In this section, a step-by-step approach has been adopted to provide the details of how the iPhone application was built. As the code snippets are provided, iOS development literature from the iOS developer library and/or books will be consulted and the details will be provided.

### 6.5.1 Step1: Creating the Project SafetyIphone

This section assumes that XCode has been downloaded and installed by clicking on the *.dmg* obtained from the Apple developer website as shown in Section 6.1. Once the Xcode is launched, a welcome screen will show up which will give the developer options to either create a new project or open previous opened recent projects. This welcome screen is shown in Figure 6.2.

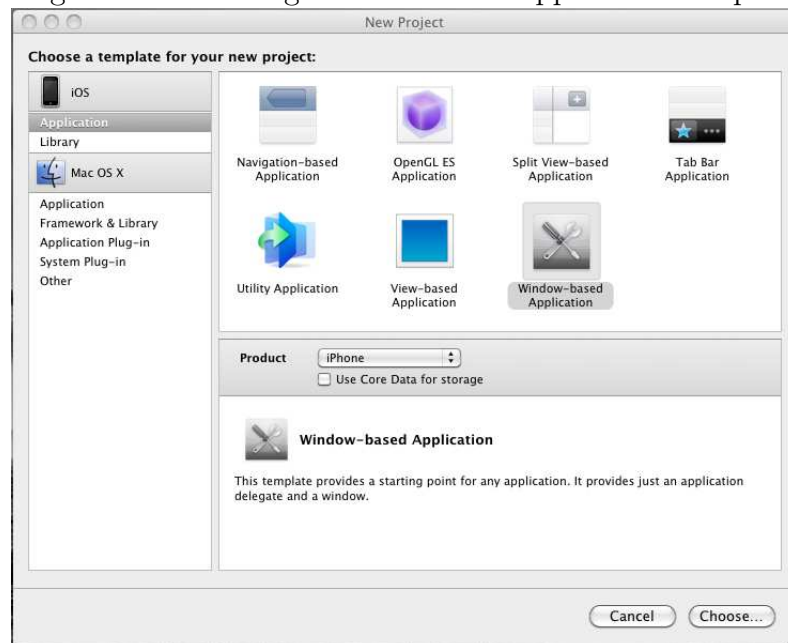
Figure 6.2: Welcome screen in Xcode



After clicking on *Create a new Xcode project* button as shown in Figure 6.2, the developer will be asked which template Xcode should select. These templates make lives easier for the developers as the developers don't have to build connections between

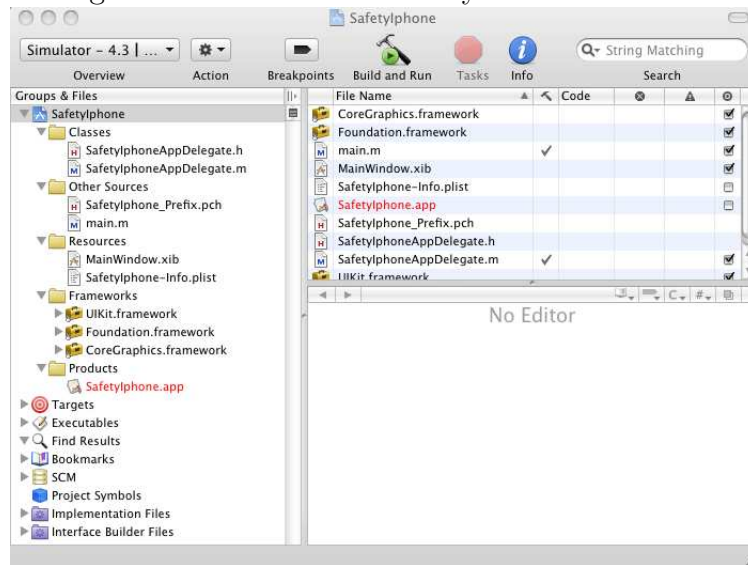
the low level objects, create autorelease pools and delegate files, or write the code from the very beginning. In fact, based upon the templates chosen, the developers are provided with commented out methods that are necessary for the implementation of a Mac or iOS project. The available templates in iOS SDK are shown in Figure 6.3. Although the easier way to go is to select a View based application (which won't be discussed here and is thoroughly explained in Mark et al. (2011)), a Window-based Application will have intentionally been chosen to understand the intricacies involved in iOS application development. As mentioned in the summary of Window-based applications, this template will only provide a window and an application delegate, which will be explained later in this chapter.

Figure 6.3: Selecting Window-based application template



Lets name the project *SafetyIphone* and save it under the */Users/<username>/Documents* folder. Figure 6.4 shows the files created automatically by the Xcode development environment.

Figure 6.4: Files automatically created in Xcode



### 6.5.2 Step2: Adding resources to the SafetyIphone project

The Daytime Safety Belt Usage survey project required that the observers collect safety belt usage data at pre-determined locations. Ten variables were collected for every vehicle which included: safety belt status, age, gender and ethnicity of the front row occupants, state of registration of the vehicle and the vehicle type. Hence, it was important that the iPhone application built for the project had capabilities to store the data in a well structured format. Although iOS SDK allows the developers to save the data in text files, a well structured sqlite data allows them to run queries on the fly and publish results without any hassle. In order to add sqlite capabilities to the iPhone project, *libsqlite3.dylib* library needs to be added to the following pre existing iPhone frameworks:

1. *UIKit framework*:

This framework provides classes for the construction and management of the iOS application's user interface. It provides an application object, event handling

capabilities and user interface elements for the touch enabled iOS interfaces.

Figure 6.5 provides a hierarchial structure of the several classes provided by the UIKit framework.

## 2. *Foundation framework:*

According to the Foundation Framework reference in the iOS developer library (2010a), this framework includes NSObject(root object class), classes for several datatypes like strings, collection classes for data storage, system classes like date and classes providing information about the communication ports. The set of classes under this framework is huge and is shown in Figure 6.6.

## 3. *Coregraphics framework:*

This framework, written in C programming language, is based on Quartz drawing engine and is required for 2D rendering of objects. It handles color and image management, PDF document creation and is responsible for nice visual patters, gradients and shadings. The iOS developer library does not demonstrate the classes in CoreGraphics framework in a graphical format. More information about this framework can be found in the iOS developer library (2009a).

As mentioned earlier, *libsqlite3.dylib* library needs to be added to the project to enable the Sqlite database support in iPhone applications. In order to add this library, follow these steps:

1. Right click the Frameworks folder in the *SafetyIphone* project and select Add  
— > Existing Files.

Figure 6.5: Classes in UIKit framework (Adapted from iOS developer library, 2010d)

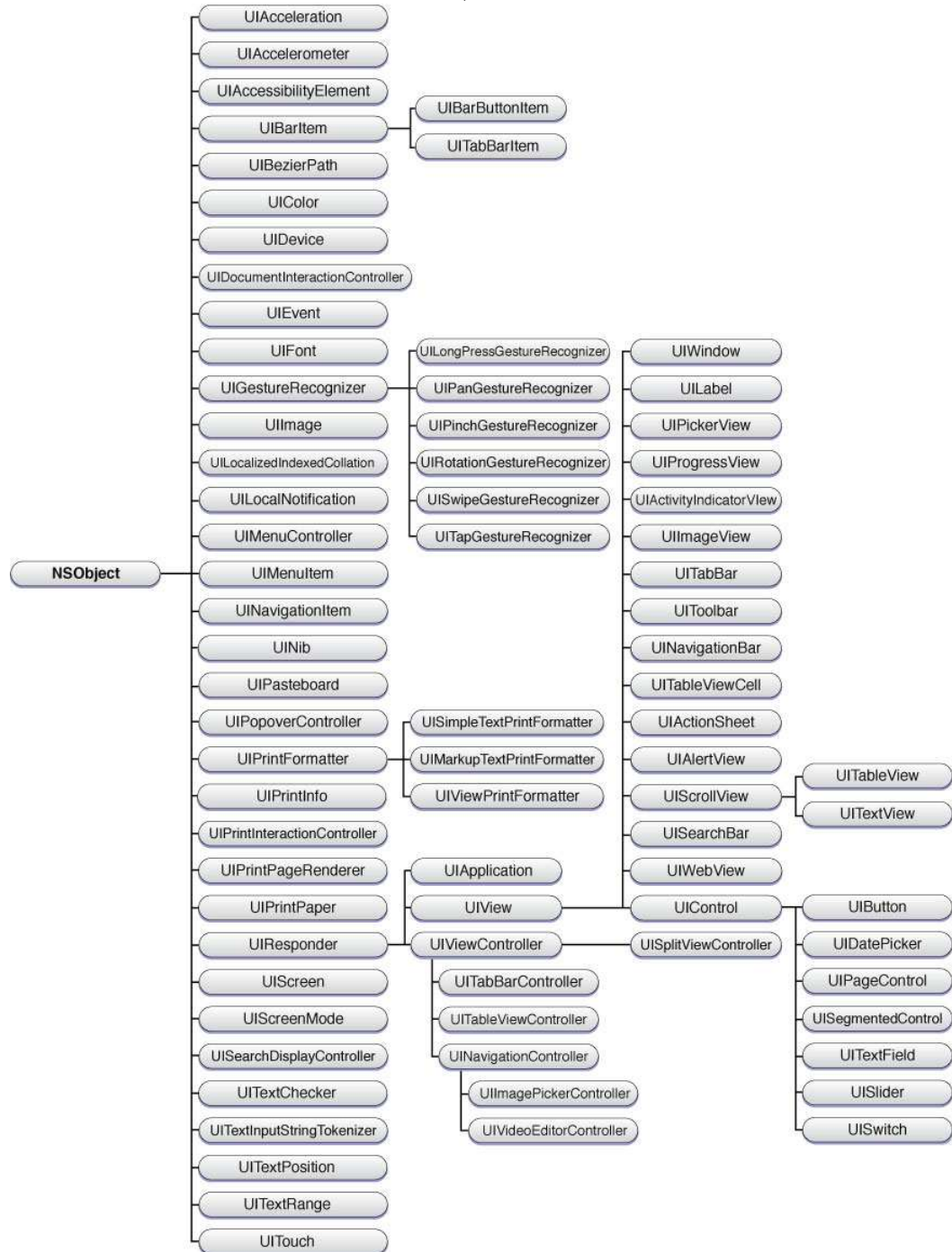
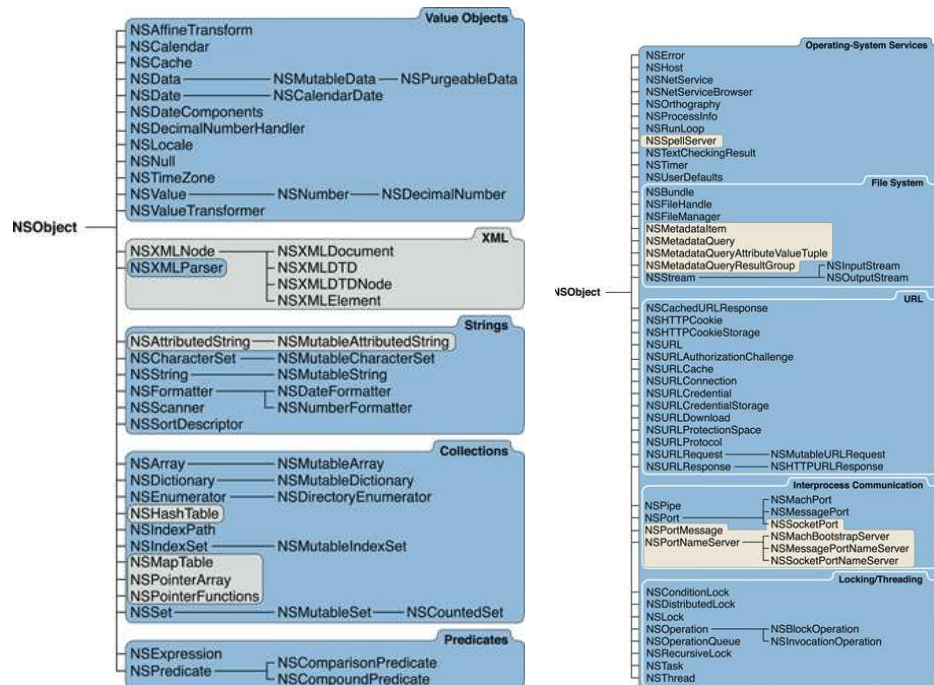
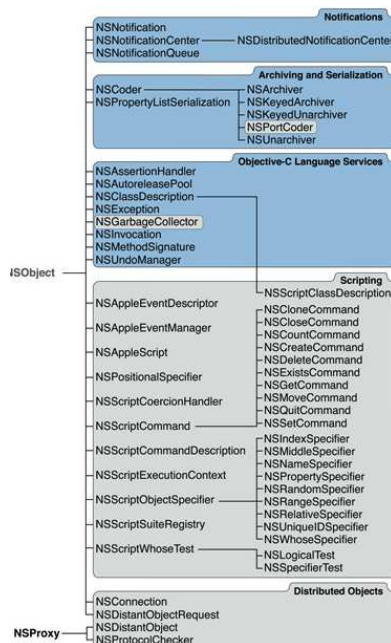


Figure 6.6: Classes in Foundation framework (Adapted from iOS developer library, 2010a)



(a) Classes

(b) Classes contd. . .



(c) Classes contd. . .

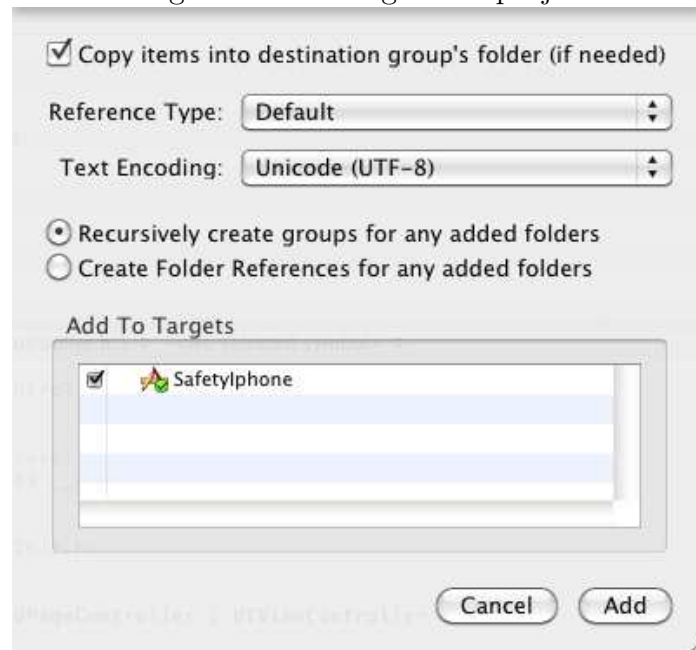
2. Navigate to Mac OSX <version> -- > Developer -- > Platforms -- > iPhoneOS.platform -- > Developer -- > SDKs -- > iPhoneOS<version>.sdk -- > usr -- > lib
3. Scroll down to find the file *libsqlite3.dylib* and add it to the project.

Now, *libsqlite3.dylib* has been added to the *Frameworks* folder in the *SafetyIphone* project. Next, we will add an icon which will represent the application when installed on the iOS device or iOS simulator. If this is not done, the installed application will be a default gray image, which of course won't attract any attention. Hence, we will create a portable network graphic (.png) icon and include it within the Resources folder of the project. It should be noticed although most format images are displayed well, Xcode automatically optimizes .png images while the application is built and run. This makes .png images to be the fastest and efficient image types in iOS based applications. We chose a 57 by 57 image named *icon1.png* as our application icon. In order to add it to our *SafetyIphone* project, follow these steps:

1. Right click on Resources folder in the project and choose Add -- > Existing Files.
2. Browse to the location of the PNG image you want to set for the application.
3. Click Add and a dialog shown in Figure 6.7 will pop up.

In this dialog box, select the checkbox *Copy items into destination group's folder (if needed)*. If not already selected, choose *Default* for the dropdown menu in front of *Reference Type:* label. Similarly, select *Unicode (UTF8)* for the *Text*

Figure 6.7: Adding file to project



*Encoding:* label. Then, select the checkbox next to *Recursively create groups for any added folders*. Hit add and the image will be added to the Resources folder. By adding the image to the Resources folder, the image has been built into the application bundle. Now, we need to specify that this image needs to be our application icon.

4. Single click on the *SafetyIphone-Info.plist* file under the *Resources* folder. This file is commonly known as a *property list* which contains information about our application, including the application icon and the company name. This property list is shown in Figure 6.8.
5. Within the property list, find a cell with *Icon file* in the left *Key* column. The cell in the *Value* column next to *Icon file* should be empty. Double click this cell and type the name of the PNG file, which in our case is *icon1.png*.



Figure 6.8: Property list

Key	Value
▼ Information Property List	(13 items)
Localization native development re	English
Bundle display name	\${PRODUCT_NAME}
Executable file	\${EXECUTABLE_NAME}
Icon file	icon1.png
Bundle identifier	com.trcunlv.\${PRODUCT_NAME:rfc1034identifier}
InfoDictionary version	6.0
Bundle name	\${PRODUCT_NAME}
Bundle OS Type code	APPL
Bundle creator OS Type code	????
Bundle version	1.0
Application requires iPhone enviro	<input checked="" type="checkbox"/>
Main nib file base name	MainWindow
Application supports iTunes file sh	<input checked="" type="checkbox"/>

6. Now, find a cell with *Bundle identifier* in the left *Key* column. The value next to this cell contains a unique identifier for our application. Double click the word *yourcompany* and edit it to your company name, which in our case is *TRC*.

Next, we will create an sqlite file named *safetydata.sqlite* on the terminal. Open Mac OS X terminal and browse to the Documents directory using the `cd` command. On the terminal, write '`sqlite safetydata.sqlite`' and press enter. On the Sqlite prompt, write the following command and replace `i` by 1 through 64 to generate 64 statements like the one in Listing 6.1. Enter the above 64 statements on the Sqlite prompt and hit enter. This will create 64 tables in *safetydata.sqlite* file. Write '`.quit`' on sqlite prompt and hit enter to come out of it, back to the terminal. Finally, import *safetydata.sqlite* to the Resources folder.

Listing 6.1: Creating *safetydata.sqlite* file on terminal in MAC OS X

```

1 CREATE TABLE sitei(entry_id INTEGER PRIMARY KEY, driversb VARCHAR(3),
2   driverage VARCHAR(5), driversex VARCHAR(6), driverrace VARCHAR(9),

```

```
3 passengersb VARCHAR(3), passengerage VARCHAR(5), passengersex VARCHAR(6),  
4 passengerrace VARCHAR(9), state VARCHAR(6), vehicle VARCHAR(19));
```

### 6.5.3 Step 3: Adding Custom View Controllers

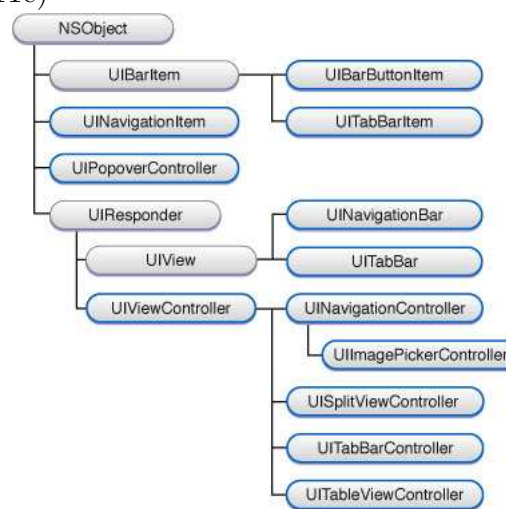
Before we actually add custom view controllers to our project, let's take a look at what they actually mean. In order to understand Custom view controllers, we need to go over the nice methods provided by the `UIViewController` class. This section provides the details of the `UIViewController` Class, Custom View Controllers, Navigation items and demonstrates how to add custom view controllers to our *SafetyIphone* project. Let's start with the `UIViewController` class.

#### 6.5.3.1 UIViewController Class Reference

When the MVC paradigm was described in Section 6.2, it was mentioned that controllers act as middle entities between the model and the view. These controllers manage how the view is presented and respond to the user events by querying the model. Similarly, `UIViewController` class in iOS SDK manages the presentation of views in iOS based applications. Every instance of `UIViewController` is used to manage a view hierarchy, which consists of a root view referenced by the *view* property of `UIViewController` class and a few subclasses which show the content. `UIViewController` class is contained in the `UIKit` framework, as shown in Figure 6.9.

View controllers, commonly known as the traditional controller objects in the MVC paradigm, are associated with only one view object, which are root view objects of a view hierarchy. In our *SafetyIphone* application, there are two screens or views

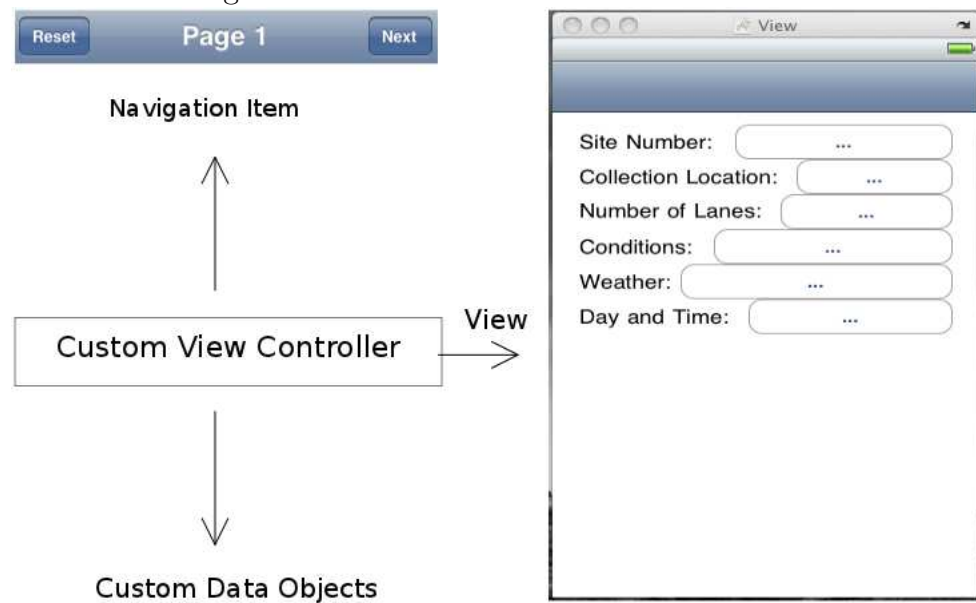
Figure 6.9: UIViewController class as a part of UIKit framework(Adapted from iOS Developer Library, 2011c)



(see Figure 6.1) that the user can see. These two screens are managed by distinct view controller objects. These two view controller objects manage the presentation of two different view objects, filling them with data and appropriate content, finally responding to the user interactions with the respective view. The view controllers are able to respond to the user interactions (also called as events) because their ancestor class is UIResponder. This is shown in Figure 6.9. In fact, the view controllers are sandwiched between the managed root view and its superview managed by a different view controller object. Hence, if a specific event dispatched due a certain user interaction is not handled by the associated view object of the view controller, the view controllers can handle the event before passing it to the superview.

For our application, the two view controllers which manage the view objects shown in Figure 6.1 are the subclasses of UIViewController class. These view controllers are called Custom View Controllers and are described in the next section. UIViewController class provides several methods, which will be described when the code snippets

Figure 6.10: Structure of FirstViewController



specific to the *SafetyIphone* project are provided later in this chapter.

### 6.5.3.2 Custom View Controllers

Whenever we need to present the views containing application specific content (PickerView with Round buttons in Figure 6.1), custom view controllers come into picture. Custom view controllers are created by subclassing `UIViewController` class, where we present the associated view and implement a few methods to manage event handling and memory management. In our application, we are using two custom view controllers namely *FirstViewController* and *SecondViewController* for managing content in Figure 6.1. Figure 6.10 shows some of the key objects owned and managed by the *FirstViewController*. Although the view object accessible through the *view* property of the `UIViewController` class is the only necessary component, other objects are required to support navigation.

In this section, a brief introduction to the way a custom view controller is imple-

mented will be shown, while the details will be provided in the next section where we add and modify our custom view controllers *FirstPageController* and *SecondPageController* to our *SafetyIphone* project. For the successful implementation of a custom view controller, the following steps need to be taken:

1. Creating and configuring the view to be loaded by the view controller.
2. Customizing the appearance of objects necessary for navigation between the views (navigation items to be specific).
3. Efficiently managing the memory and releasing when the view is deallocated.

#### 6.5.3.3 Adding two Custom View Controllers: *FirstPageController* and *SecondPageController*

Xcode provides the developers a template to subclass `UIViewController` class and hence add Custom View Controllers to the applications without any hassle. Lets start with *FirstPageController*. For adding *FirstPageController* to our *SafetyIphone* project, consider the following steps:

1. Right click on the *Classes* folder and select Add — > New File. This opens up a window (shown in Figure 6.11) which provides file templates for standard file types. This simplifies the process of creating the skeleton of a project. Single click on Cocoa Touch Class available under the iOS panel. This shows four standard file templates available for the developers. We will select a subclass of the *UIViewController* class, since the *FirstPageController* is a Custom View

Controller. In the middle right section of the Cocoa Touch Class, the developers are given three options to choose from:

(a) *Targeted for Ipad:*

Uncheck this checkbox since this *SafetyIphone* is an iPhone/iPod Touch project.

(b) *UITableViewController subclass:*

Uncheck this checkbox since we will not be require a table based layout.

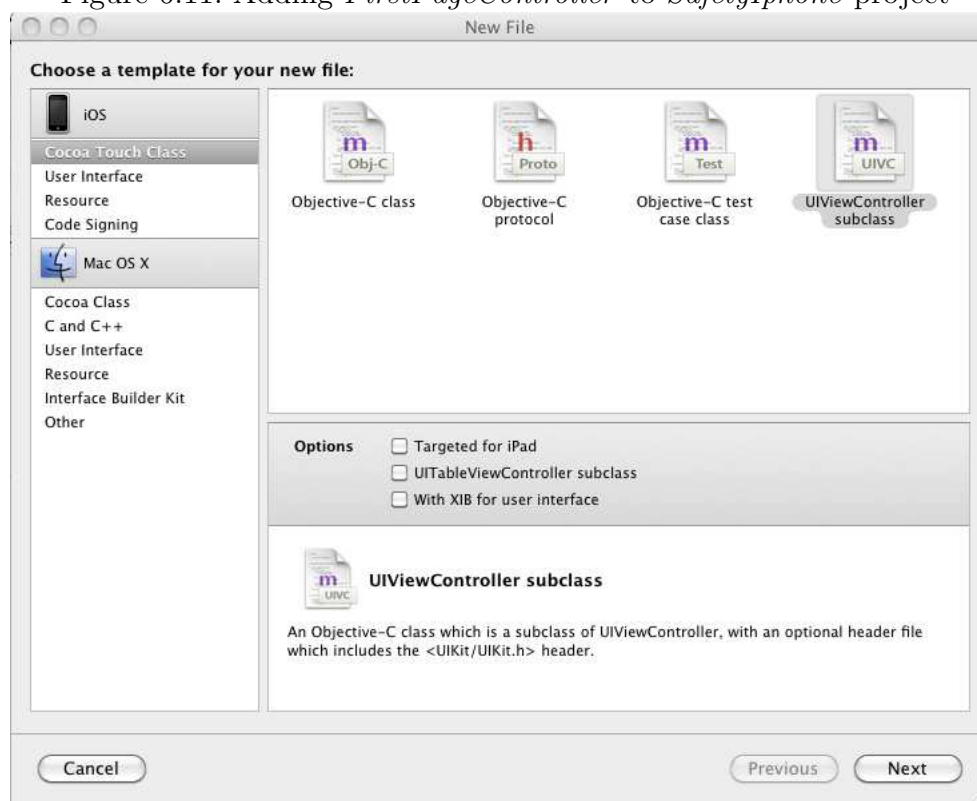
(c) *With XIB for user interface :*

If we select this checkbox, Xcode will create a nib file (a file with an extension of *.xib* and can be modified by the Interface Builder to change the view contents) corresponding to the controller class. However, we will not select this checkbox for providing a better comprehension of what goes under the shell.

2. Click Next and a window will appear which allows us to name the source file created in the above step with a *.m* extension. We will give it a name *First-PageController.m* and will also select the next checkbox which will allow us to create a corresponding header file with *.h* extension. This window is shown in Figure 6.12.

Repeat the same procedure for *SecondPageController* which will create and add *SecondPageController.h* and *SecondPageController.m* file to the *SafetyIphone* project.

Figure 6.11: Adding *FirstPageController* to *SafetyIphone* project

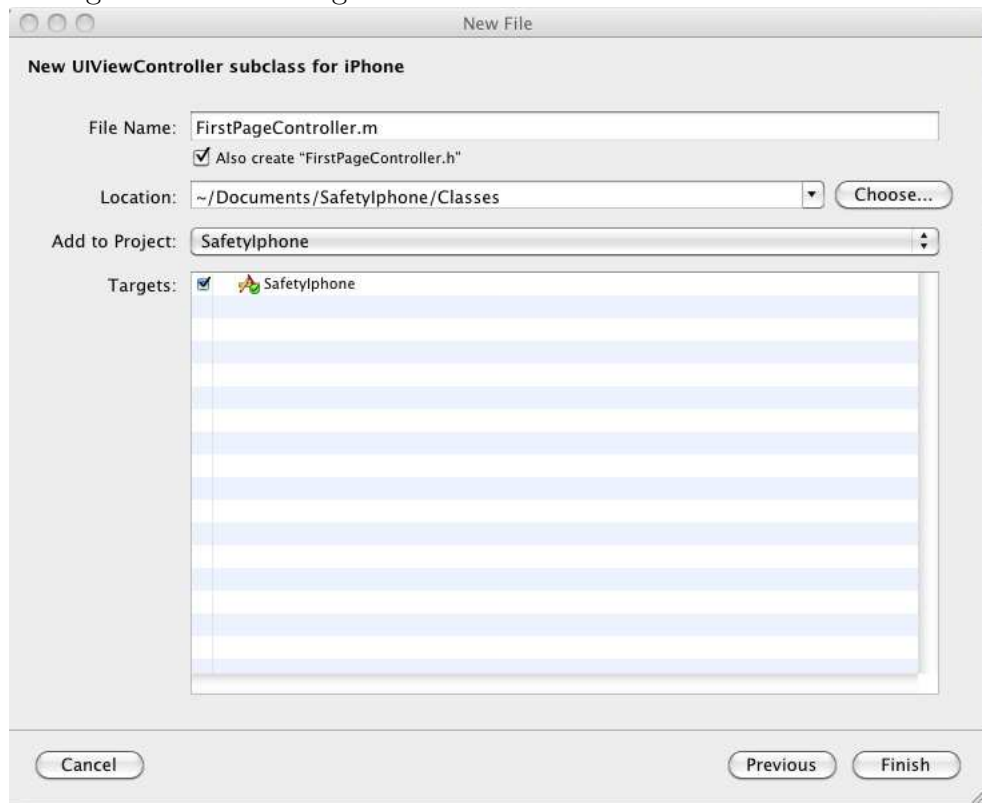


#### 6.5.3.4 Creating Nib files for *FirstPageController* and *SecondPageController*

Interface builder is a graphical development environment that comes along with Xcode. Interface Builder provides a library of view components like buttons, labels etc. along with controllers like ViewController, NavigationController etc. Using a simple drag and drop functionality, the developers can easily and quickly build views with content appealing to the public. The files which open in Interface Builder have *.xib* extension but are commonly known as *nib* files since Apple recently changed the extension of these files from *.nib* to *.xib*.

In fact, there are two methods to create views, one using the Interface Builder and the other programmatically. Since Interface Builder is an essential component of quick and easy iOS based application development, we will use it to create and modify

Figure 6.12: Creating *FirstViewController* source and header files



our views. Essentially, the *loadView* method from the UIViewController class needs to be overridden for adopting a programmatic approach towards view development. Further details are available online (iOS Developer Library, 2011c).

Now, let's create and add *FirstPageController.xib* file to our *SafetyIphone* project first. The approach will be exactly the same for adding *SecondPageController.xib*. Consider the following steps:

1. Right click on the *Resources* folder and select Add —> New File.
2. The window shown in Figure 6.11 will appear once again. This time, under the iOS heading, select the *User Interface* option to bring up a menu for standard *xib* file types. Select the *View XIB* template and mark the product option as



*iPhone*. The *View XIB* template will provide us with a Cocoa Touch view where we can drag controls from the Library onto the view and adjust the properties of the controls.

3. Click the Next button. We will give this file the name *FirstPageController.xib*, where the given name matches the name of the *FirstPageController* class except the extensions (*.h*, *.m* and *.xib*). Although the same name concept is not required, this is done for two reasons. Firstly, it is convenient to have a nib file with the same name as the view controller. Secondly, having the same name is important for the `nibName` property of the `initWithNibName:bundle:` method defined inside the `UIViewController` class. The details will be provided later in the chapter when we modify the contents of the *FirstPageController.m* and *SecondPageController.m* files.

Repeat the same process for *SecondPageController.xib* file. After this step, no more files or resources will be added to the *SafetyiPhone* project.

#### 6.5.4 Modifying the App delegate

In Layman's terms, delegates are those classes which implement certain tasks on the behalf of another object. When we created the *SafetyiPhone* project, Xcode automatically created two files *SafetyiPhoneAppDelegate.h* and *SafetyiPhoneAppDelegate.m* which implement *application delegate*. This *application delegate* lets us perform tasks on the behalf on *UIApplication* class. *UIApplication* class forms an important part of the UIKit framework and makes a call to specific methods, implemented by the application delegate if it exists. For example, when the application launches

and its nib file has been loaded in an inactive state, *UIApplication* class makes a call to the *application:didFinishLaunchingWithOptions:* method implemented by the application delegate. In this method, we can put our own custom code to initialize the application. After this method returns, *UIApplication* class makes another call to some other method implemented by the application delegate which puts the application in an active state. In the next two sections, we will write the code necessary for implementing the application delegate.

#### 6.5.4.1 Modifying *SafetyIphoneAppDelegate.h*

In the *Groups and Files* pane of the *SafetyIphone* project, single click on the *SafetyIphoneAppDelegate.h* file. This is the application delegate's header file. The code in this file is given in Listing 6.2:

Listing 6.2: *SafetyIphoneAppDelegate.h*

```
1  #import <UIKit/UIKit.h>
2  #import "FirstPageController.h"
3  @interface SafetyIphoneAppDelegate : NSObject <UIApplicationDelegate> {
4      UIWindow *window;
5      FirstPageController *first_page_controller;
6  }
7  @property (nonatomic, retain) IBOutlet UIWindow *window;
8  @end
```

#### Explanation of Listing 6.2:

Listing 6.2 shows the header file of the application delegate. The header files contain the declarations of structs, variables and the method prototypes. The stepwise description of Listing 6.2 is as follows:

1. Line 1 tells the compiler to look at the `UIKit.h` header file in the `UIKit` framework
2. In Line 2, we import the definitions in the class `FirstController.h`
3. Line 3 introduces `@interface` keyword which is an Objective C compiler directive. Before an object can be created from a specific class, the compiler needs to know about the data members(instance variables) of the object and the features (methods) it provides. `@interface` keyword provides such information to the compiler. `SafetyAppDelegate: NSObject` represents that this class is a subclass of the `NSObject` class available in the Foundation framework included in the project. The important part in Line 1 is the `<UIApplicationDelegate>` keyword. This keyword indicates that the class `SafetyAppDelegate` conforms to a *protocol* called `UIApplicationDelegate`. Holding down the option key and double clicking on the keyword `UIApplicationDelegate` in Xcode will up a window which provides basic information about this protocol. Essentially, this keyword notifies the compiler that the `UIApplicationDelegate` class will call the methods inside `SafetyAppDelegate` since it has agreed to implement the application delegate.
4. In Line 4, an instance `window` of the `UIWindow` class is created by Xcode. This `window` object was created when we chose to create a window based application. The window object will always be created, no matter what template is chosen to create the application on iOS. The `window` object can be seen in Figure 6.13, generated by double clicking on the `MainWindow.xib` file.

Figure 6.13: Window object in *MainWindow.xib*



5. In Line 5, we create an instance *first\_page\_controller* to the *FirstPageController* class, whose properties we will change when we modify *SafetyIphoneAppDelegate.m*
6. Note that the elements defined within curly braces of the *@interface* keyword are called instance variables of the class.
7. In Line 7, the keywords like *@property*, *nonatomic*, *retain* and *IBOutlet* have been introduced.
  - (a) *@property*:

Properties were added to Objective C to replace the traditional accessors (getters) and mutator (setters) methods required for getting or setting the values of the instance variables. The *@property* keyword in *SafetyIphoneAppDelegate.h* header file goes hand in hand with the *@synthesize* keyword defined in the *SafetyIphoneAppDelegate.m* implementation file. As soon as the *@synthesize* keyword is defined, the accessor and mutator

methods are created.

(b) *nonatomic*:

This is an optional attribute of the *@property* keyword. In Layman terms, *nonatomic* attribute is mentioned when we need to avoid creating additional overhead code which is created by default when the accessor and mutator methods are generated. If a property is declared atomic (default), its associated accessor will provide robust access to the property in a multi threaded environment. In other words, the values of the instance variables declared as atomic properties is instantly retrieved from the accessor or the mutator methods(after setting the value), no matter what other thread are doing at the present. Therefore, by declaring a property as nonatomic, we avoid that overhead code which is not required when a pointer to a user interface object (*window*) is declared.

(c) *retain*:

The *retain* attribute is a very important part of the memory management scheme in iOS based applications, which will be discussed later in this chapter. Essentially, this attribute will tell the compiler to send a *retain* message (which returns an id) to the *window* object. This will prevent the window object from getting flushed from the memory when it is still being used.

It is important to know that the *retain* attribute is not the only one that can specify the semantics of a set accessor. *assign* is the default attribute

which specifies that the setter uses a simple assignment other than sending a *retain* message. The *assign* attribute is used with low-level C datatypes like *int*, *float* etc or with scalar types like *NSInteger* and *CGRect*. Note that if the since datatypes like *int*, *float* etc. are not Objective C objects, they cannot be sent a *retain* message. Furthermore, *copy* attribute specifies that a copy of the object should be used for assignment, while the previous value is sent a *release* message. Note that *NSString* objects are always copied and hence *copy* attribute must be used while declaring properties for *NSString* objects.

(d) *IBOutlet*:

The IBOutlet keyword is defined in Listing 6.3.

Listing 6.3: Definition of IBOutlet

```
1 #ifndef IBOutlet
2 #define IBOutlet
3 #endif
```

Listing 6.3 shows that the IBOutlet doesn't do anything for the compiler but tells the Interface Builder that we will use the instance variable to connect to an object in the nib file. In our case, the *window* instance variable has been declared as an IBOutlet to give a hit to the Interface Builder that we will connect it to the *window* object in *MainWindow.xib* file. It is important to note that when we defined the window instance variable, we did not mention the IBOutlet keyword within the curly braces

of the *@interface* keyword. Although *IBOutlet UIWindow \* window* is also supported, Apple has been recently inclined towards declaring the *IBOutlet* keyword with *@property* keyword as shown in Listing 6.2.

#### 6.5.4.2 Modifying *SafetyIphoneAppDelegate.m*

Listing 6.2 showed the header file of the *SafetyIphoneAppDelegate* class. In this section, we will implement the same class. When Xcode created this source file, it automatically gave us some commented out delegate methods to work with. However, all these methods within the template were not used as they were not required for the *SafetyIphone* project. Since *SafetyIphoneAppDelegate* conforms to the *UIApplicationDelegate* protocol, all these methods can be found under its reference within the iOS developer library (2010b).

Listing 6.4 shows the implementation of this class. Once again, the pre-existing code has been shown in normal typeface while the bold font represents the code that was actually added to this source template.

Listing 6.4: *SafetyIphoneAppDelegate.m*

```
1  #import "SafetyIphoneAppDelegate.h"
2  #import "FirstPageController.h"
3  @implementation SafetyIphoneAppDelegate
4  @synthesize window;
5  #pragma mark -
6  #pragma mark Application lifecycle
7
8  - (BOOL) application:(UIApplication *)application didFinishLaunchingWithOptions:
9      (NSDictionary *)launchOptions {
10     first_page_controller = [[FirstPageController alloc]
```

```

11         initWithNibName:@"FirstPageController" bundle:nil];
12     UINavigationController *navController = [[ UINavigationController alloc]
13         initWithRootViewController:first_page_controller];
14     [window setRootViewController:navController];
15     [navController release];
16     [self.window makeKeyAndVisible];
17     return YES;
18 }
19
20 - (void)applicationWillTerminate:( UIApplication *)application {
21     [first_page_controller termsoon];
22 }
23
24 #pragma mark -
25 #pragma mark Memory management
26
27 - (void)dealloc {
28     [first_page_controller release];
29     [window release];
30     [super dealloc];
31 }
32 @end

```

### Explanation of Listing 6.4:

Listing 6.4 shows the source file of the application delegate, which implements the methods provided in the *UIApplicationDelegate* protocol. The stepwise description of Listing 6.4 is as follows:

1. Lines 1 and 2 tell the compiler to look at the *UIKit.h* and *FirstPageController.h* header files for the definitions of methods and data members.



2. Line 3 introduces *@implementation* keyword which is an Objective C compiler directive. *@implementation* tells the compiler that the actual code that will make an object work is about to start. *@implementation* is followed by the name our class *SafetyAppDelegate*. Usually, the methods implemented in *@implementation* are declared in *@interface*, however this is not a requirement. The methods implemented in *@implementation* but not mentioned in *@interface* can be considered as private methods used just in the implementation of the class.
3. Listing 6.2 contained a property declaration of the *window* outlet that is automatically connected to the *window* object by the Interface Builder. *@synthesize* keyword on Line 4 tells the compiler to create the accessor and mutator methods the *window* instance variable.
4. *#pragma mark* introduced in Line 5 and 6 is a directive to the IDE and tells the Xcode editor to put a break in the popup menu of methods at the top of the editor pane. This directive particularly comes very handy if the code in a file is too long and we need to jump to a particular code snippet. *#pragma mark* - puts a horizontal line spacer in the code while the *#pragma mark <labelname>* adds the given *<labelname>* in bold to the popup menu of methods.
5. - (*BOOL*) *application:didFinishLaunchingWithOptions:* method is implemented in Lines 8-18. Most of the methods created by developers start with a leading minus (-) sign and are known as instance methods. However, some methods exist which are also declared with a leading plus (+) sign. These methods are

called class methods or factory methods. This method belongs to the class object (which does not mean the instance of an object), which is associated with the following features: represents a class and is created when Objective C runtime builds the class, contains pointers to superclass, class name and to the list of the methods in the class. An example of an instance method is shown on Line 8 of Listing 6.4. Similarly, an example of a class method is *stringWithFormat:* which creates an object based upon the given argument.

**application:didFinishLaunchingWithOptions: method (iOS Developer Library, 2010c):**

The *application:didFinishLaunchingWithOptions:* method in the *UIApplicationDelegate* protocol is called after the application has launched and has finished loading, but is still in inactive state. It has two parameters: the delegating application object (*application*) and a dictionary (*launchOptions*) containing the reasons why the application was launched. If the URL resource is successfully handled by the application, a bool *YES* will be returned otherwise *NO* will be returned. This method is called after the application has launched and has finished loading, but is still in inactive state.

**initWithNibName:bundle: method (iOS Developer Library, 2011a):**

The *initWithNibName:bundle:* method in the *UIViewController* class returns an initialized view controller with the nib file provided in the given bundle. It has two parameters: associated view controller nib file (*nibName*) and the bundle (*nibBundle*) where the nib file will be searched. The nib file given in the *nibName* property is only loaded when the associated view controller's view is

accessed for the first time. There are two important things to note about the parameters. Firstly, if *nibName* parameter is set to be *nil*, then either a nib file with the same name as the view controller should exist in the application bundle or the view should be created by overriding the *loadView* method in a subclass of the *UIViewController* class. Secondly, the nib files is first searched in the given bundle's project directory, and then in the *Resources* folder. However, if a *nil* argument is assigned to the *nibBundle* property, the nib file is searched in the main bundle.

***initWithRootViewController:* method (iOS Developer Library, 2009b):**

The *initWithRootViewController:* method in the *UINavigationController* class is a factory method that pushes a root view controller onto the navigation stack and returns an initialized navigation controller. The root view controller is taken as a parameter and is set to be at the bottom of the navigation stack.

***makeKeyAndVisible* method (iOS Developer Library, 2011b):**

The *makeKeyAndVisible* method in the *UIWindow* class is a factory method which makes the receiver a key window and displays it in the front of other windows, if any. This method does not return anything and hence, the return type is declared as void.

**Using *application:didFinishLaunchingWithOptions:* method:**

Firstly, we create memory for the *FirstPageController* object by calling the *alloc* method of the *NSObject* class. *alloc* method initializes the *isa* instance variable of the newly allocated *FirstPageController* object and makes it point

to the *FirstPageController* class. The rest of the instance variables defined in the *@interface* section of the *FirstPageController* class are set to 0. However, it is important that the newly allocated *FirstPageController* object be specifically initialized before it is safely used in the application. Since *FirstPageController* inherits the instance methods of *UIViewController* class, the initialization of this *FirstPageController* object is done by the *initWithNibName:bundle:* method described above. Hence, we initialize the newly allocated *FirstPageController* object by calling *initWithNibName:bundle:* method and sending the *FirstPageController.xib* file in the project's *Resources* folder and nil as parameters. *initWithNibName:bundle:* method will search for the provided nib file in the main bundle and will return an initialized view controller object pointed by *first\_page\_controller*.

On Line 12, we similarly allocate memory to a *UINavigationController* object and then initialize it using the *initWithRootViewController:* instance method defined in the *UINavigationController* class. *first\_page\_controller*, pointing to the initialized view controller object of *FirstPageController* class is sent as a root view controller argument to the *initWithRootViewController:* method and an initialized navigation controller is returned from the method. *navController* points to this navigation controller. *navController* is then set as root view controller of the *window* instance variable connected to the *window* object in *MainWindow.xib*.

Next, Line 13 sends a *release* message to the *navController* object owned by the *SafetyIphoneAppDelegate* class. In order to briefly introduce memory manage-

ment, lets quickly go through how the *navController* object is flushed from the memory when the *release* message is sent. When the *alloc* method was called on the *UINavigationController* object, the application was asked to set the *retain count* of the *navController* object to 1. Since the *navController* object is not being used elsewhere in the application, we sent a *release* message which reduced it's *retain count* to 0. Whenever the retain count of any object becomes 0, the application is notified that the object is no longer needed and can be destroyed, hence returning its memory back to the system for use. Hence, this destroyed the *navController* object and its memory was returned to the system. Line 14 of Listing 6.4 makes the current *window* object as the main window and displays it on the screen. This is done by calling the *makeKeyAndVisible* method of the *UIWindow* class. Finally a bool *YES* is returned to the application, indicating that the URL resource was successfully handled.

## 6. Details regarding *applicationWillTerminate:* method: (iOS Developer Library, 2010c)

The *applicationWillTerminate:* method in the *UIApplicationDelegate* class is called when the user quits the application that does not support background execution. It takes the delegating application object as a parameter and informs the application that it is about to be terminated and removed from the memory. This method is useful for the final clean up tasks before an application is terminated. In our application, this method calls a method in *SecondPageController* class to close an open connection to the *SQLite* database.

7. Lines 27-31 present the *dealloc* method from the *NSObject* class. Since every class is a subclass of the root *NSObject* class, every class must implement its version of *dealloc* method. In this method, we send a release message to the *window* and *first\_page\_controller* instance variables owned by the *SafetyAppDelegate* class and later call the superclass version of *dealloc* by sending a message to *super*.

### 6.5.5 Modifying FirstPageController

In the above section, the code added a navigation controller to the window object whose root view controller was initialized to be a *FirstPageController* object with an associated *FirstPageController.xib* file. In this section, we will first create the view for the *FirstPageController* in the Interface Builder and will later provide explanations to the *FirstPageController* header and the source files.

#### 6.5.5.1 Modifying *FirstPageController* in Interface Builder

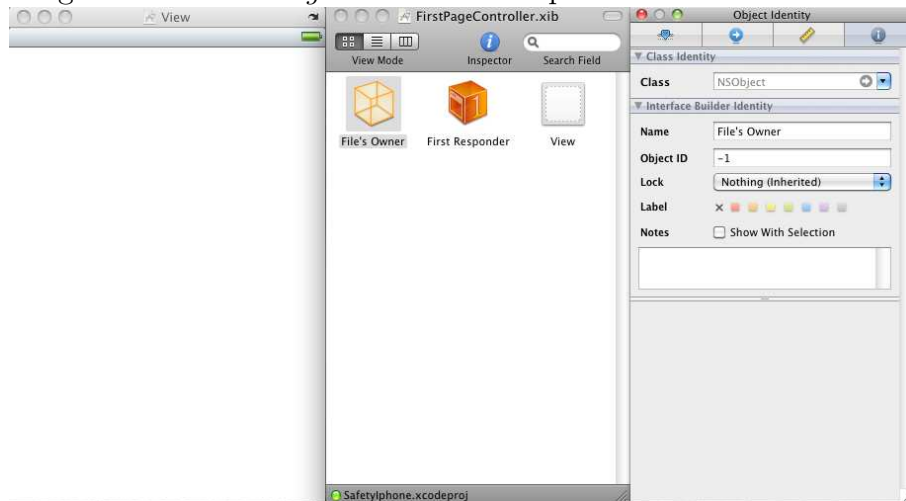
In our project window's *Groups and Files* list, look for *FirstPageController.xib* and double click on the file to open it in Interface Builder. The window opened is shown in Figure 6.14.

When we created *FirstPageController* class in Figure 6.11, we had unchecked the option *With XIB for user interface*. Now, we need to associate *FirstPageController.xib* file with the class *FirstPageController*. To do so, two steps are required from our side:

1. Changing the owner class:

In Figure 6.14, we can see a *File's Owner* icon. This is always the first icon in

Figure 6.14: *FirstPageController.xib* opened in Interface Builder

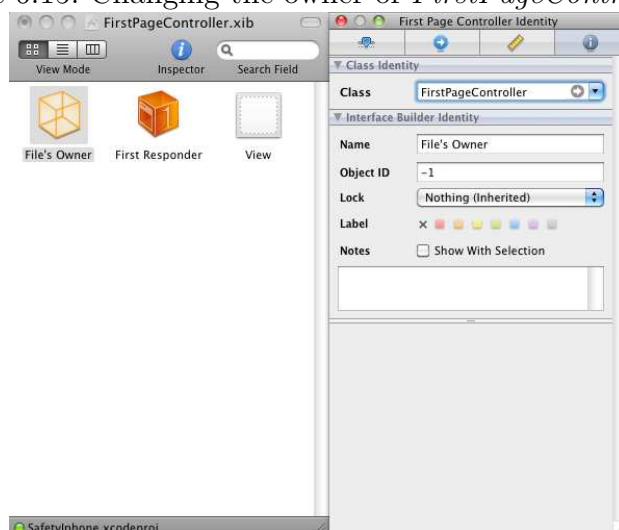


any nib file and represents the object responsible for loading the nib file from the disk. The object loading this nib file is it's owner. Therefore, we need to change the *Class* attribute of the *File's owner* in *FirstPageController.xib* file. Single Click on *File's Owner* icon and bring forward the *Identity Inspector* from the *Tools* menu of the Interface Builder. In Figure 6.14, we can see that the root class *NSObject* is the owner of *FirstPageController.xib*. Instead of *NSObject*, select *FirstPageController* from the drop down menu. After this little change, *FirstPageController* class will own *FirstPageController.xib* file. This is shown in Figure 6.15.

## 2. Connecting the *view* outlet:

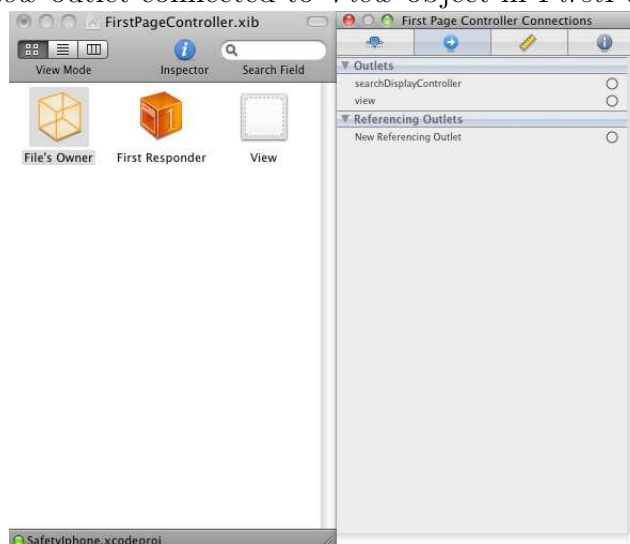
Hold down the control key on the keyboard and drag from the *File's Owner* icon to the *View* object. Now, release the mouse and a menu will be displayed. In this menu, select the *view* outlet. This technique represents connecting an outlet from an owner class to an object. This allows the *FirstPageController* to

Figure 6.15: Changing the owner of *FirstPageController.xib*



manage the *View* object in the associated nib file through its *view* property. To make sure that the *view* outlet has been appropriately connected to the *View* object, single click on the *File's owner* icon in *FirstPageController.xib* and select the *Connections Inspector* from the *Tools* menu of the *Interface Builder*. This will show a window similar to the one shown in Figure 6.16.

Figure 6.16: *view* outlet connected to *View* object in *FirstPageController.xib*

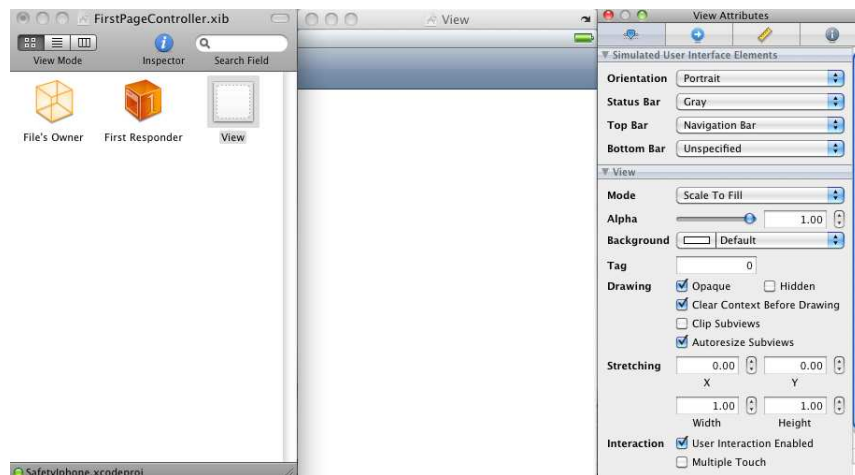


After the above two steps, the stage is set to change the *View* of the *FirstPage*-



*Controller.xib* as desired. On the first page of our application, we need to allow the users to select from a few options before the safety belt usage data can be collected at a particular site. However, we also need to allow navigation between two screens: first which allows the users to select information about a specific site and the second which actually lets the users collect the safety belt usage data. We start modifying the *View* object by adding a *Navigation Bar* at the top of it. This is done by selecting the *Navigation Bar* option from the drop down menu next to *Top Bar* in the *Attributes Inspector* obtained from the *Tools* menu in *Interface Builder*. This process is shown in Figure 6.17.

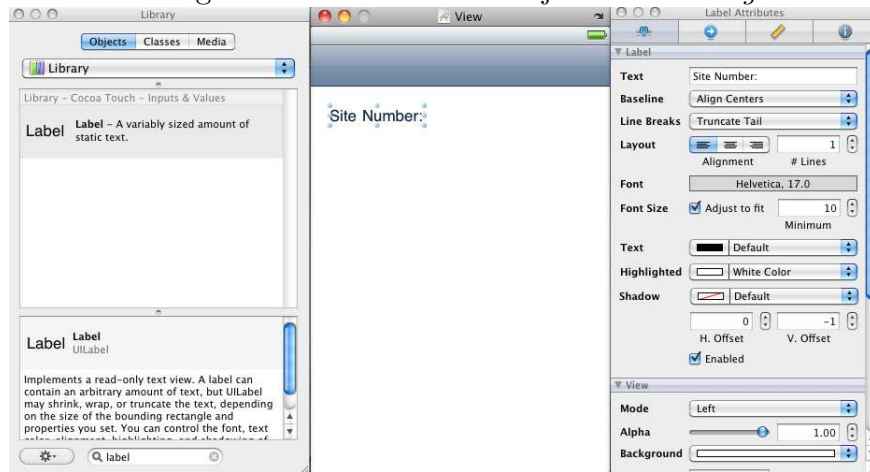
Figure 6.17: Attaching a Navigation Bar to the *View* object in *FirstPageController.xib*



Next, bring forward the *Library* from *Tools* menu in the *Interface Builder*. Search for a label in the library and drag it onto the screen of the *View* object. Double tap the dragged *label* object and change its title according to your preference. In our case, we change it to *Site Number*:. Although the other attributes were not changed for the labels in *SafetyIphone* project, all the available attributes can be found by bringing forth *Attributes Inspector* from the *Tools* menu. The process of searching a

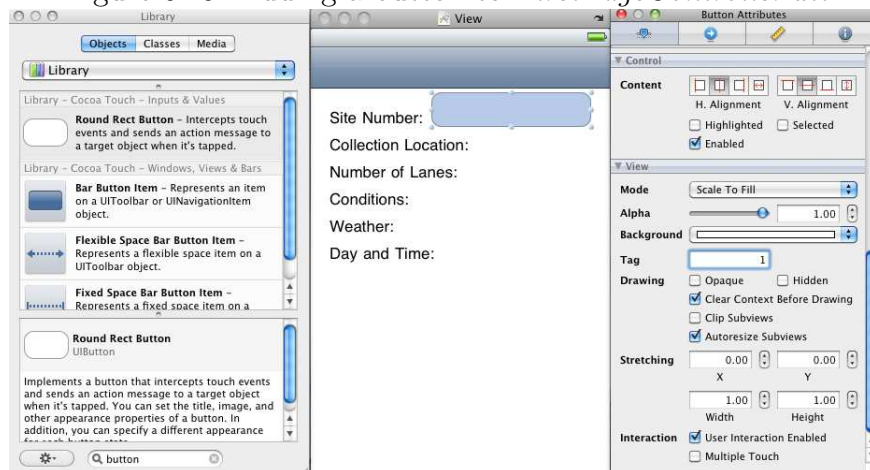
label in the library, dragging it onto the *View* object and changing its title is shown in Figure 6.18.

Figure 6.18: Adding a Label to the *View* object in *FirstPageController.xib*



Next, add five more labels to the *View* and change their titles. In our application, these titles have been set for those five labels: *Collection of Location*, *Number of Lanes*, *Conditions*, *Weather* and *Day and Time*. Next, look up a button in the library and drag in onto the *View* as shown in Figure 6.19.

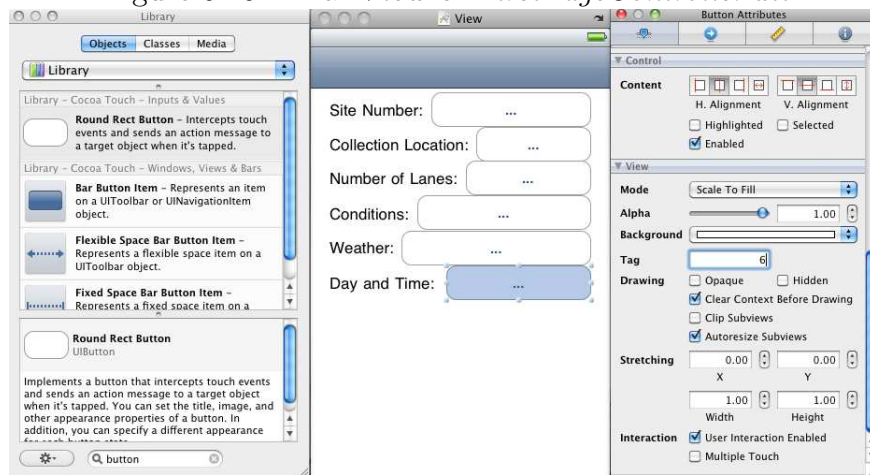
Figure 6.19: Adding a button to *FirstPageController.xib*



There is a very important property called *tag* associated with all the subclass of *UIView* class, including views and controls. The *tag* property represents a numeric

value set as a unique identifier to an object, which will never be changed by the system and will remain the same throughout the application execution. In Figure 6.19, we set the button's *tag* attribute to 1. Similarly, the *tag* attribute is also set for next 5 buttons we drag onto the *View*. The buttons are given 'tags' 1 through 6, with the last button added given a *tag* value of 6. The *View* obtained by setting the navigation bar to the top, adding labels and buttons and changing their respective attributes is shown in Figure 6.20.

Figure 6.20: Final *View* of *FirstPageController.xib*



No more controls will be added to the *View* in *FirstPageController.xib*, although connections will be made to the *outlets* and the *action methods* (defined later) after modifying the *FirstPageController.h* and *FirstPageController.m* files.

#### 6.5.5.2 Modifying *FirstPageController.h*

In the *Groups and Files* pane of the *SafetyIphone* project, single click on the *FirstPageController.h* file. This is the header file for the *FirstPageController* class. The code in this file is given in Listing 6.5.

### Listing 6.5: FirstPageController.h

```
1  #import <UIKit/UIKit.h>
2  @class SecondPageController;
3  @interface FirstPageController : UIViewController <UIPickerViewDelegate,
4      UIPickerViewDataSource, UIAlertViewDelegate>
5  {
6      SecondPageController *second_page_controller;
7      int which_button;
8      int current_1;
9      int current_2;
10     int current_3;
11     int current_4;
12     int current_5;
13     int current_row;
14     int i;
15     int j;
16     bool ispickerup;
17     bool isdateup;
18     bool firsttime_1;
19     bool firsttime_2;
20     bool firsttime_3;
21     bool firsttime_4;
22     bool firsttime_5;
23     bool SN_answered;
24     bool CL_answered;
25     bool NL_answered;
26     bool C_answered;
27     bool W_answered;
28     bool TD_answered;
29     UIPickerView *myPickerView;
30     UIDatePicker *myDatePicker;
31     NSMutableArray *arrayNo;
32     NSMutableArray *collection_location;
```

```

33  NSMutableArray *number_of_lanes;
34  NSMutableArray *conditions;
35  NSMutableArray *weather;
36  NSMutableArray *time_and_day;
37  UIButton *SN_button;
38  UIButton *CL_button;
39  UIButton *NL_button;
40  UIButton *C_button;
41  UIButton *W_button;
42  UIButton *TD_button;
43  }
44  @property (nonatomic, retain) IBOutlet UIButton *SN_button;
45  @property (nonatomic, retain) IBOutlet UIButton *CL_button;
46  @property (nonatomic, retain) IBOutlet UIButton *NL_button;
47  @property (nonatomic, retain) IBOutlet UIButton *C_button;
48  @property (nonatomic, retain) IBOutlet UIButton *W_button;
49  @property (nonatomic, retain) IBOutlet UIButton *TD_button;
50  - (IBAction)choose_timeday:(id)sender;
51  - (IBAction)choose_everything:(id)sender;
52  - (void)termsoon;
53  @end

```

### Explanation of Listing 6.5:

Listing 6.5 shows the header file of the *FirstPageController*. The stepwise description of this header file is as follows:

1. Similar to Listing 6.2, the definitions in *UIKit.h* are imported on Line 1.
2. *@class* keyword on Line 2 tells the compiler that *SecondPageController* is a class and will be referred only via a pointer *second\_page\_controller* on Line 6 of the code.

3. In our application, we have a pickerview control which uses a spinning wheel to show or more user defined values. We also use a datepicker control that uses the same spinning wheel mechanism and lets the users select dates and times. Lines 3-4 suggest that the class *FirstPageController*, a subclass of *UIViewController* conforms to three protocols: *UIPickerViewDelegate*, *UIPickerViewDataSource* and *UIAlertViewDelegate*. In short, this class implements methods declared in all these three protocols as follows:

(a) Conforming to *UIPickerViewDelegate*:

The class *FirstPageController* implements a few required methods of this protocol to return the width, height, the title of each row and the content for the rows in every component of the picker view. According to the definition in the iOS developer library (2009c), a component is a wheel with a series of objects represented by rows in the vertical manner. Consider the alarm application on an iPhone. When we edit an alarm, we are provided with a picker view which has three separate wheels or components to set hour, minute and the period format (am/pm). Similarly, *FirstPageController* will populate the picker view with data and optionally implement the methods which respond to interaction with the rows in the picker view.

(b) Conforming to *UIPickerViewDataSource*:

The class *FirstPageController* implements both the required methods of this protocol to provide the picker view with the number of rows and components in the picker view.

(c) Conforming to *UIAlertViewDelegate*:

By conforming to the *UIAlertViewDelegate* protocol, *FirstPageController* class declares that it will be a delegate of a *UIAlertView* object. In other words, *FirstPageController* will implement methods when the user interacts with the buttons in an alert view displayed on the iOS screen.

Note that no delegates or data sources are required for implementing date pickers.

4. Lines 7-28 declared a few low level C datatypes that will be used in the *FirstPageController* class. Note that these instance methods won't be declared as properties since these are not Objective-C objects, and hence cannot be sent any messages.
5. Lines 29-30 declare two pointers to the picker view and the date picker controls added programatically in *FirstPageController* class implementation.
6. Lines 31-36 declare six mutable arrays needed to populate the picker view and the date picker corresponding to the user clicks on 6 rounded rectangle buttons shown in Figure 6.20.
7. Lines 37-49 declare six instance variables as outlets to the six rounded rectangle buttons shown in Figure 6.20. These outlets will be used by the *FirstPageController* class to communicate with the buttons at runtime.
8. Lines 50-51 define *action methods*, which are special methods which tell the *Interface Builder* that those methods are actions and can be triggered by controls.

The action methods typically take the id of the sender control as an argument and returns nothing, thus being similar to the return type *void*. When the user taps on all the buttons except the button next to the *Day and Time* label in Figure 6.20, the *id* of that button will be sent to the *choose\_everything* action method which handles the event. Similarly, *choose\_timeday* action method will be called when the user interacts with the button next to the *Day and Time* label. Lastly, Line 52 defines an instance method *termsoon* which will be implemented in *FirstPageController.m*

### 6.5.5.3 Modifying *FirstPageController.m*

In this section, we will implement the *FirstPageController* class. Since the code in this file is quite long, code snippets of the important methods have been provided in every listing. The functionality of the code in the snippet will be explained but some unnecessary details will be skipped while writing the listing itself.

Listing 6.6 shows the customization of the *initWithNibName:bundle:* method in *FirstPageController* class.

Listing 6.6: Customizing *initWithNibName:bundle:* method in *FirstPageController* class

```

1  #import "FirstPageController.h"
2  #import "SecondPageController.h"
3  @implementation FirstPageController
4  @synthesize SN_button, CL_button, NL_button, C_button, W_button, TD_button;
5
6  - (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil {
7      self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];

```



```

8  if (self) {
9      second_page_controller = [[SecondPageController alloc]
10     initWithNibName:@"SecondPageController" bundle:nil];
11     myPickerView = [[UIPickerView alloc]
12     initWithFrame:CGRectMake(0, 200, 320, 200)];
13     myPickerView.delegate = self;
14     myPickerView.showsSelectionIndicator = YES;
15     myDatePicker = [[UIDatePicker alloc]
16     initWithFrame:CGRectMake(0, 200, 320, 200)];
17     myDatePicker.datePickerMode = UIDatepickerModeDateAndTime;
18     myDatePicker.hidden = NO;
19     [myDatePicker addTarget:self action:@selector(changeDateInLabel:)
20     forControlEvents:UIControlEventValueChanged];
21     [[self navigationItem] setTitle:@"Page 1"];
22     UIBarButtonItem *reset_one = [[UIBarButtonItem alloc]
23     initWithTitle:@"Reset" style:UIBarButtonItemStylePlain
24     target:self action:@selector(reset_button_pushed:)];
25     [[self navigationItem] setLeftBarButtonItem:reset_one animated:YES];
26     [reset_one release];
27     UIBarButtonItem *done_one = [[UIBarButtonItem alloc] initWithTitle:
28     @"Next" style:UIBarButtonItemStylePlain
29     target:self action:@selector(right_button_pushed:)];
30     [[self navigationItem] setRightBarButtonItem:done_one animated:YES];
31     [done_one release];
32 }
33 return self;
34 }
35 [...]
36 @end

```

### Explanation of Listing 6.6:

Listing 6.6 shows a couple of new methods that have been used to create view

objects. This section first provides the definitions and explanations of those methods and then, elucidates how the customized *initWithNibName:bundle* method works in the above Listing 6.6. Listing 6.7 provides the definitions of the methods in Listing 6.6.

Listing 6.7: Definitions of methods in Listing 6.6

```

1  - (id)initWithNibName:(NSString *)nibName
2      bundle:(NSBundle *)nibBundle /*UIViewController class*/
3  + (id)alloc /*NSObject class*/
4  - (id)initWithFrame:(CGRect)aRect /*UIView class*/
5  CGRect CGRectMake (CGFloat x,CGFloat y,CGFloat width,CGFloat height);
6      /*CGGeometry class*/
7  - (void)addTarget:(id)target action:(SEL)action
8      forControlEvents:(UIControlEvents)controlEvents /*UIControl Class*/
9  - (id)initWithTitle:(NSString *)title style:(UIBarButtonItemStyle)style
10     target:(id)target action:(SEL)action /*UIBarButtonItem Class*/
11 - (void)setLeftBarButtonItem:(UIBarButtonItem *)item
12     animated:(BOOL)animated /*UINavigationController Class*/
13 - (void)setRightBarButtonItem:(UIBarButtonItem *)item
14     animated:(BOOL)animated /*UINavigationController Class*/

```

The explanation of the methods listed above are as follows:

1. *initWithNibName:bundle:* method

As previously mentioned in Section 6.5.4.2, this method searches for the given nib file (*nibName*) in the specified bundle (*nibBundle*) of the application and returns a newly initialized view controller.

2. *alloc* method

This class method returns a new instance of the receiving class, thereby setting the *isa* instance variable of the new instance to a data structure which describes the receiver class.

3. *initWithFrame:* method

This method initializes and returns a view object according to a frame rectangle specified by *aRect* property which is added with respect to the coordinates in the superview.

4. *addTarget:action:forControlEvents:* method

When a user interacts with controls on a user interface, an event is dispatched by an internal table. This method attaches an action message to a specified target and adds it to the dispatch table when an event specified by *controlEvents* is triggered.

5. *initWithTitle:style:target:action:* method

This method creates and returns a stylish navigation item (showed in Figure 6.10) with a particular title and sends an action message (*action*) to a particular target object(*target*) when this item is tapped.

6. *setLeftBarButtonItem:animated:* method

This method sets a custom bar item to the left of the navigation bar (showed in Figure 6.17) and moves to the next view with/without a transition animation based on the bool *animated* parameter.

7. *setRightBarButtonItem:animated:* method

This method sets a custom bar item to the right of the navigation bar and moves to the next view with/without a transition animation based on the bool *animated* parameter.

After explaining the methods used in Listing 6.6, it's time to customize the *initWithNibName:bundle:* method. First, we synthesize the outlets connected to the button controls on the user interface of the *FirstPageController* view. Then, we call the *initWithNibName:bundle:* method of the super class *UIViewController* to perform the internal initializations and return a view controller. If the returned view controller object is not *nil*, we allocate and initialize *SecondPageController* based view controller object by searching for the *SecondPageController.xib* file in the main bundle. Next, we create and initialize a 320 by 200 rectangular picker view and attach it to the *FirstPageController View* with its origin specified at (0, 200). In Listing 6.5, we had declared that *FirstPageController* conforms to *UIPickerViewDelegate* protocol. By setting the *delegate* property to self, we specify that the *FirstPageController* view controller object will serve as the delegate of the picker view. We also make sure that an overlay is displayed on any row of the picker view when it is selected by setting the *showsSelectionIndicator* property. Similarly, we also initialize a rectangular date picker allowed to select both date and time with a single overlay add it to the same origin (0,200) as the picker view. The picker view will stay hidden when the date picker is active and vice versa. It has already been mentioned that date pickers do not need any data sources or delegates which are required by the picker view and are implemented by *FirstPageController*. Therefore, via *addTarget:action:forControlEvents:*

method, we specify that when the value of the date picker is changed, *changeDateInLabel:* method is called. Further, we set the title of the *FirstPageController* view by calling the setting the title of the *navigationItem*. Note that we could also have done that by *[self setTitle]*. However, we set the title of the navigation button to show that in case a navigation controller is used and a new view is pushed onto the stack, the title of the previous view will be the text of our back button on the new view. However, in our case, this does not matter that much since we add a custom button as a bar item in *SecondPageController* implementation. Next, Listing 6.6 also shows that we add a *Reset* button to the left of the navigation bar. This reset button when tapped, will call *reset\_button\_pushed* method that will clear the selected values on the *FirstPageController View*. Lastly, we also add a *Next* button to the right of the navigation bar which calls *right\_button\_pushed* method when it is tapped. This will take the user to the next view which allows the users to collect the safety belt usage data.

Next, *viewDidLoad* method has been customized in Listing 6.8.

Listing 6.8: Customizing viewDidLoad method

```

1  [...]
2  - (void)viewDidLoad {
3      [super viewDidLoad];
4      ispickerup = NO;
5      isdateup = NO;
6      /*Other variables initialized*/
7      collection_location = [[NSMutableArray alloc] init];
8      [collection_location addObject:@"Freeway"];
9      /*Addition of 5 similar NSString objects to collection_location array*/

```

```

10     number_of_lanes = [[NSMutableArray alloc] init];
11     [number_of_lanes addObject:@"1"];
12     /*Addition of 7 similar NSString objects to number_of_lanes array*/
13     conditions = [[NSMutableArray alloc] init];
14     [conditions addObject:@" Available"];
15     /*Addition of 5 similar NSString objects to conditions array*/
16     weather = [[NSMutableArray alloc] init];
17     [weather addObject:@" Sunny"];
18     /*Addition of 5 similar NSString objects to weather array*/
19     arrayNo = [[NSMutableArray alloc] init];
20     [arrayNo addObject:@"1. I15 at Valley of Fire"];
21     /*Addition of 63 similar NSString objects to arrayNo array*/
22     SN_answered = NO;
23     CL_answered = NO;
24     NL_answered = NO;
25     C_answered = NO;
26     W_answered = NO;
27     TD_answered = NO;
28 }
29 [...]

```

### Explanation of Listing 6.8:

In Listing 6.8, we have customized the *viewDidLoad* method provided by the *UIViewController* class. This method does not take any arguments, neither it returns anything back to the caller. After the *FirstPageController* finishes loading its associated view into the memory, this method is called to perform additional data initialization on the view loaded into the memory from *FirstPageController.xib* file. First, we call the *viewDidLoad* method of the super class *UIViewController* to perform the internal initializations if any. In this code snippet, we create 5 mutable arrays to

hold *NSString* objects which feed the picker view with site information: collection location, number of lanes, conditions(availability), weather and the site name. We also create and initialize a few variables *current\_i* ( $i = 1 \dots 5$ ) to keep track of what values have been currently chosen in the respective arrays. Last but not the least, we keep track of the information if all the buttons displayed in Figure 6.20 have been selected or not. By doing so, we make sure if all the information for a particular site has been chosen. If not, the user is not allowed to advance to the next view where he/she can collect the safety belt usage data.

In Listing 6.9, *viewDidLoad* and *dealloc* methods from *UIViewController* class have been customized.

Listing 6.9: Customizing viewDidLoad and dealloc methods

```
1  [...]
2  - (void)viewDidLoad {
3      SN_button = nil;
4      CL_button = nil;
5      NL_button = nil;
6      C_button = nil;
7      W_button = nil;
8      TD_button = nil;
9      [super viewDidLoad];
10 }
11
12 - (void)dealloc {
13     [SN_button release];
14     [CL_button release];
15     [NL_button release];
16     [C_button release];
17     [W_button release];
18     [TD_button release];
```

```
19     [super dealloc];  
20 }  
21 [...]
```

### Explanation of Listing 6.8:

In Listing 6.8, we have provided the details of how *viewDidLoad* and *dealloc* methods have been customized in our *FirstPageController* class. *viewDidLoad* and *dealloc* methods are found in *UIViewController* and *NSObject* class respectively. *viewDidLoad* method is called during low memory conditions when the controller needs to reclaim the memory of its view and the associated view objects. In order to reclaim the memory of an object, the view controllers need to give up the ownership by setting its value to be *nil*. When the *FirstViewController* class calls *viewDidLoad* method, it gives up the ownership of the outlets, which stored the references of the round rectangle buttons (objects) used to collect site specific information as shown in Figure 6.20. Then, the *viewDidLoad* method is called for the super class to perform the remaining memory clean up if any.

Since *FirstPageController* class had instance variables, it is important that a *dealloc* method is implemented within *FirstPageController* class which releases the instance variables and then call super's implementation to free the memory occupied by *FirstPageController* object. It should be noted that *dealloc* method is never called from an object directly. Instead, it is sent a *release* or an *autorelease* message and the owner class takes care of the rest. In the next few listings, we first present the action methods called when the user interacts with the controls and later present the



customized picker view delegate and data source methods need to populate the picker view with the relevant information.

Action methods were previously explained in Listing 6.5. Since action methods form an integral part of any iOS application and this is the first time we are coming across the implementation of an action method, we will provide a generic approach to how the actions are specified. Before specifying the actions, it is convenient that at least the action methods have been mentioned in the header file of the class (see Listing 6.5) so that it is easier to connect them to the controls through the Interface Builder. We will take the example of the first button next to the label *Site Name:* in Figure 6.20 and will specify an action method for that control. Right click on the button and bring up the *Connections Inspector* from the *Tools* menu. Under the *Events* heading, find the event *Touch Up Inside*. This event will be triggered when the user taps a control, but only if the last place touched before lifting the finger was inside the control territory. Click on the little circle next to *Touch Up inside* and drag the gray line obtained with the mouse button pressed from the circle to the *File's Owner* icon. When the mouse button is released on *File's Owner* icon, a pop menu showing the *action methods* available will be displayed. For all the buttons except the button next to *Day and Time* label shown in Figure 6.20, choose *choose\_everything* action method from the popup menu. This will create a connection between the action method *choose\_everything* and the button controls. Repeat the same procedure while connecting *choose\_timeday* action method implemented within *FirstPageController.m* to the button next to *Day and Time* label.

Listing 6.10 provides the two custom action methods connected to the six button

controls discussed above.

Listing 6.10: Action methods in *FirstPageController* class

```
1  [...]
2  - (IBAction)choose_everything:(id)sender
3  {
4      if (isdateup == YES){
5          [myDatePicker removeFromSuperview];
6          isdateup = NO;
7      }
8      switch ([sender tag]) {
9          case 1:
10             which_button = 1;
11             if (firsttime_1 == YES) {
12                 [SN_button setTitle:@"%@" , [arrayNo objectAtIndex:0]]
13                 forState:UIControlStateNormal];
14                 second_page_controller.site_num = 1;
15                 NSString *temp = ("%@" , [arrayNo objectAtIndex:0]);
16                 second_page_controller.site_description = [temp substringFromIndex:3];
17                 second_page_controller.county = @"Clark";
18                 second_page_controller.area = @"Rural";
19                 second_page_controller.volume = @"High";
20                 firsttime_1 = NO;
21             }
22             SN_answered = YES;
23             break;
24          case 2:
25             which_button = 2;
26             if (firsttime_2 == YES) {
27                 [CL_button setTitle:@"%@" , [collection_location
28                 objectAtIndex:0]]
29                 forState:UIControlStateNormal];
30                 second_page_controller.where_collection = [collection_location
```

```

31         objectAtIndex:0];
32         firsttime_2 = NO;
33     }
34     CL_answered = YES;
35     break;
36 case 3:
37     which_button = 3;
38     if (firsttime_3 == YES) {
39         [NL_button setTitle:@"%@", [number_of_lanes objectAtIndex:0]]
40         forState:UIControlStateNormal];
41         second_page_controller.lanes = [number_of_lanes objectAtIndex:0];
42         firsttime_3 = NO;
43     }
44     NL_answered = YES;
45     break;
46 case 4:
47     which_button = 4;
48     if (firsttime_4 == YES) {
49         [C_button setTitle:@"%@", [conditions objectAtIndex:0]]
50         forState:UIControlStateNormal];
51         second_page_controller.conditions = [conditions objectAtIndex:0];
52         firsttime_4 = NO;
53     }
54     C_answered = YES;
55     break;
56 case 5:
57     which_button = 5;
58     if (firsttime_5 == YES) {
59         [W_button setTitle:@"%@", [weather objectAtIndex:0]]
60         forState:UIControlStateNormal];
61         second_page_controller.weather = [weather objectAtIndex:0];
62         firsttime_5 = NO;
63     }

```

```

64         W_answered = YES;
65         break;
66     default:
67         NSLog(@"I don't know which button it is");
68         break;
69     }
70     [myPickerView reloadData];
71     switch (which_button) {
72     case 1:
73         [myPickerView selectRow:current_1 inComponent:0 animated:NO];
74         break;
75     case 2:
76         [myPickerView selectRow:current_2 inComponent:0 animated:NO];
77         break;
78     case 3:
79         [myPickerView selectRow:current_3 inComponent:0 animated:NO];
80         break;
81     case 4:
82         [myPickerView selectRow:current_4 inComponent:0 animated:NO];
83         break;
84     case 5:
85         [myPickerView selectRow:current_5 inComponent:0 animated:NO];
86         break;
87     default: initialize
88         NSLog(@"Which option should I chose in pickerView?");
89         break;
90     }
91     [self.view addSubview:myPickerView];
92     ispickerup = YES;
93 }
94
95 - (IBAction)choose_timeday:(id)sender
96 {

```

```

97     if (ispickerup == YES){
98         [myPickerView removeFromSuperview];
99         ispickerup = NO;
100    }
101    which_button = 6;
102    myDatePicker.date = [NSDate date];
103    myDatePicker.maximumDate = [NSDate date];
104    [self.view addSubview:myDatePicker];
105    isdateup = YES;
106    NSDateFormatter *df = [[NSDateFormatter alloc] init];
107    df.dateStyle = NSDateFormatterShortStyle;
108    df.timeStyle = NSDateFormatterShortStyle;
109    [TD_button setTitle:(@"%@ %@", [NSString stringWithFormat:@"%@",
110        [df stringFromDate:myDatePicker.date]]) forState:UIControlStateNormal];
111    second_page_controller.date_ = (@%@", [NSString stringWithFormat:@"%%",
112        [df stringFromDate:myDatePicker.date]]);
113    [df release];
114    TD_answered = YES;
115 }
116 [...]

```

### Explanation of Listing 6.10:

Listing 6.10 provides two action methods triggered when the user taps one of the six buttons kept to store the site information. First, let's discuss the method *choose\_everything*. As previously mentioned, this method is called when the user needs to enter site name, collection location, number of lanes, availability conditions and weather by tapping on any one of these five buttons. In Figure 6.19, we had set the *tag* attribute of all the six buttons on the screen from top to bottom in an ascending order, starting from 1. This *tag* property helps us uniquely identify which button

control triggered the action *choose\_everything*. Within *choose\_everything* method, we first remove the date picker from the *superview*, since tapping first five buttons should bring up only the picker view and has nothing to do with the date picker. This is done by calling the *removeFromSuperview* method (found within *UIView* class) on the date picker component, which unlinks it from its superview and its window, making it unable to respond to user events. Then, from Lines 8-89, we use *switch-case* statements to find out which button was tapped using the sender control's *tag* property. Once this is decided which button was tapped, we assign the respective tag values to a variable *which\_button*. As will be seen later in this chapter, *which\_button* helps us globally keep track of which button was tapped and will also be used while implementing picker view delegate methods. If the chosen button control has been tapped for the first time (*firsttime\_i* variables), we set its title using the object at the zeroth index inside the respective array using *setTitle:forState:* method from the *UIButton* class. For example: If the button control next to *Site Name:* label is chosen, *which\_button* is set to 1, which is infact it's *tag* value. Then, we set its title using the *arrayNo* array containing the site names and the *setTitle:forState:* method. As its obvious from the method's name, *setTitle:forState:* method sets the title of a *UIButton* object which can be in the following states:

1. *UIControlStateNormal:*

The default state of the control, when it is neither selected nor highlighted.

2. *UIControlStateHighlighted :*

The highlighted state of a control, when there is a touch up event on the control

or when a touch enters and exits.

3. *UIControlStateDisabled:*

The disabled state of a control.

4. *UIControlStateSelected:*

The selected state of a control, which does not affect the appearance or behaviour of certain controls but might affect the appearance of other subclasses like `UISegmentedControl`.

We then store the vital information like the collection location, number of lanes etc in the instance variables of the *SecondPageController* class. This information is passed to the *SecondPageController* class since all the database related work is done there. Finally, for every button control visited, we store the bool value *YES* to keep track of the information if there exist button controls for which a value hasn't been selected yet. If this is the case, the user is not allowed to advance to the next page. Notice the use of *substringFromIndex:* method from the *NSString* class under *Case 1:* of switch case statements in *choose\_everything* method. The use of this method was required since the *NSString* objects representing the site names stored in *arrayNo* array are of the type: *32. Gowan at Clayton.* and we needed only *Gowan at Clayton* to initialize the *site\_description* instance variable from the *SecondPageController* class. Thus, the method *substringFromIndex:* returns a string from the mentioned integer index till the end of the input string.

Now, notice the use of *reloadAllComponents* method at Line 70 in Listing 6.10. This method tells the picker view to query its delegate *FirstPageController* for the

new data. The delegate responds to this call by calling the required methods shown in Listing 6.11 and 6.12. Furthermore, sometimes, a user might click on a button control (call it X) and select an option from the provided items in the picker view. Next, he clicks on another button control (call it Y) which populates the picker view with new data corresponding to that button control. However, the user might realize that he has made a mistake or wants to change his decision from the previously selected item associated with button control X to a new value. In such a situation, we don't want the user to start selecting from the top index 0 of the picker view. We want to save the current index for every button control (whether it be X or Y) so that the user can start his selection from around the neighbourhood of the previously selected value. Lines 71-90 implement such a scenario using the *selectRow:InComponent:animated:* method. This method will select the previously selected row in a picker view for the currently tapped button control with an animation, depending upon the value of *animated*. After setting up the stage of the picker view, we add it to the current view of *FirstPageController* by calling the *addSubview:* method in *UIView* class and save the information that the picker view is up and running.

Listing 6.10 also provides the implementation of another action method, *choose\_timeday*. This action method is connected to the button control next to the *Day and Time:* label. When this button control is tapped, the picker is removed from its superview to give way to the date picker control, positioned at the same coordinates on the *FirstPageController* view. *choose\_timeday* method further sets the current date and the maximum date displayed by the date picker to be the current date and time. This is doing by setting the *date* and *maximumDate* properties of the date picker by



calling the *date* class method from the *NSDate* class. The code then adds the date picker to the view by calling the *addSubview:* method explained before. In order to display the chosen date as the title of the button control next to *Day and Time:* label, it is essential that its short representation is chosen. For such a purpose, *NSDateFormatter* class provides a constant *NSDateFormatterShortStyle* which converts a date into 'mm/dd/yy' format and time into 'hh:mm' format. This class also provide *stringFromDate:* instance method to convert the formatted date object to a string. Using the above functionality and the factory method *stringWithFormat:*, we set the title of the button control as shown on Lines 109-110 of Listing 6.10.

Till this point, we have customized the methods inherited from *UIViewController* class and presented custom instance methods written to provide the navigation and initial information collection capabilities we look from *FirstPageContoller* class. Listing 6.11 presents the implementation of the two required data source methods from *UIPickerViewDataSource* protocol.

Listing 6.11: Picker View Data Source Methods

```

1  [...]
2  #pragma mark -
3  #pragma mark Picker Data Source Methods
4  - (NSInteger)numberOfComponentsInPickerView:( UIPickerView *)pickerView
5  {
6      return 1;
7  }
8
9  - (NSInteger)pickerView:( UIPickerView *)pickerView
10     numberOfRowsInComponent:( NSInteger )component
11  {
12      switch (which_button)
13      {

```

```

14         case 1:
15             return [arrayNo count];
16             break;
17         case 2:
18             return [collection_location count];
19             break;
20         case 3:
21             return [number_of_lanes count];
22             break;
23         case 4:
24             return [conditions count];
25             break;
26         case 5:
27             return [weather count];
28             break;
29         default:
30             return 0;
31             break;
32     }
33 }
34 [...]

```

### Explanation of Listing 6.11:

As explained earlier, a picker view needs to know how many components and rows in every component there are to display its data. Since *FirstPageController* conformed to the *UIPickerViewDataSource* protocol, it promised that it will provide the picker view with the above functionality. Listing 6.11 implements this functionality. In our application, we use the picker view to display five things: site name, the collection location (intersection, freeway etc.), number of lanes at the site, the

conditions during the which safety belt usage data is being collected (availability or non availability due to congestion, accident etc) and the weather conditions. For this purpose, we maintain five separate arrays, where every array is a single 1-D array storing NSString objects. Hence, only one component (wheel) with an overlay is required to present and select the user choice. Therefore, we just return 1 component when the picker view asks *FirstPageController* class via its *numberOfComponentsInPickerView:* method. The picker view also needs to know how many rows it needs to display for that 1 component. Since all the maintained arrays present information that is mutually exclusive, we let the tapped button in Figure 6.20 decide via a switch case which array should the class approach for the relevant information. When the appropriate array is decided, we just let the picker view know the number of items in the array to be displayed within a single component. This is done by returning the *count* of the chosen array back to the picker view. Although *FirstPageController* class provides this vital information about the number of components and rows to the picker view, still the picker view needs to know what items to display and how to respond when some row is chosen. For this purpose, the delegate *FirstPageController* of the picker view needs to implement the methods in *UIPickerViewDelegate* protocol which is presented in the next Listing 6.12.

#### Listing 6.12: Picker View Delegate Methods

```
1  [...]
2  #pragma mark Picker Delegate Methods
3
4  - (CGFloat)pickerView:(UIPickerView *)pickerView
```

```

5   widthForComponent:(NSInteger)component
6   {
7       return 300;
8   }
9   - (CGFloat)pickerView:(UIPickerView *)pickerView
10  heightForComponent:(NSInteger)component
11  {
12      return 50;
13  }
14  - (NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row
15  forComponent:(NSInteger)component
16  {(NSInteger)row
17      switch (which_button)
18      {
19          case 1:
20              return [arrayNo objectAtIndex:row];
21              break;
22          case 2:
23              return [collection_location objectAtIndex:row];
24              break;
25          case 3:
26              return [number_of_lanes objectAtIndex:row];
27              break;
28          case 4:
29              return [conditions objectAtIndex:row];
30              break;
31          case 5:
32              return [weather objectAtIndex:row];
33              break;
34          default:
35              return 0;
36              break;
37      }

```

```

38 }
39 - (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row
40 inComponent:(NSInteger)component
41 {
42     current_row = row;
43     switch (which_button)
44     {
45         case 1:
46             [SN_button setTitle:@"%@", [arrayNo objectAtIndex:row]]
47             forState:UIControlStateNormal];
48             current_1 = row;
49             second_page_controller.site_num = row + 1;
50             NSString *temp = @"%@", [arrayNo objectAtIndex:row]];
51             if (row<9) {
52                 second_page_controller.site_description = [temp substringFromIndex:3];
53             }
54             else{
55                 second_page_controller.site_description = [temp substringFromIndex:4];
56             }
57             if (second_page_controller.site_num > 32)
58                 second_page_controller.county = @"Washoe";(UIPickerView *)pickerView
59             else
60                 second_page_controller.county = @"Clark";
61             if ((second_page_controller.site_num >= 1 && second_page_controller.site_num
62 <= 5) || (second_page_controller.site_num >= 33 &&
63 second_page_controller.site_num <= 39))
64                 second_page_controller.area = @"Rural";
65             else
66                 second_page_controller.area = @"Urban";
67             if ((second_page_controller.site_num == 5) ||
68 (second_page_controller.site_num >= 28 &&
69 second_page_controller.site_num <= 32) ||
70 (second_page_controller.site_num >= 38 &&

```

```

71         second_page_controller.site_num <= 39) ||
72         (second_page_controller.site_num >= 63 &&
73         second_page_controller.site_num <= 64))
74     {
75         second_page_controller.volume = @"Low";
76     }
77     else if ((second_page_controller.site_num == 4) ||
78     (second_page_controller.site_num>= 18 &&
79     second_page_controller.site_num <= 27) ||
80     (second_page_controller.site_num>= 35 && second_page_controller.site_num
81     <= 37) || (second_page_controller.site_num>= 40 &&
82     second_page_controller.site_num<= 46))
83         second_page_controller.volume = @"Medium";
84     else
85         second_page_controller.volume = @"High";
86     break;
87     case 2:
88         [CL_button setTitle:@"%@",[collection_location objectAtIndex:row]]
89         forState:UIControlStateNormal];
90         current_2 = row;
91         second_page_controller.where_collection = [collection_location
92         objectAtIndex:row];
93         break;
94     case 3:
95         [NL_button setTitle:@"%@",[number_of_lanes objectAtIndex:row]]
96         forState:UIControlStateNormal];
97         current_3 = row;
98         second_page_controller.lanes = [number_of_lanes objectAtIndex:row];
99         break;
100     case 4:
101         [C_button setTitle:@"%@", [conditions objectAtIndex:row]]
102         forState:UIControlStateNormal];
103         current_4 = row;

```

```

102         second_page_controller.conditions = [conditions objectAtIndex:row];
103         break;
104     case 5:
105         [W_button setTitle:@"%@", [weather objectAtIndex:row]]
106             forState:UIControlStateNormal];
107         current_5 = row;
108         second_page_controller.weather = [weather objectAtIndex:row];
109         break;
110     default:
111         NSLog(@"Unknown option");
112         break;
113     }
114 }
115 [...]

```

*UIPickerViewDelegate* protocol requires *FirstPageController* class to implement *pickerView:widthForComponent:*, *pickerview:rowHeightForComponent:* and

*pickerView:titleForRow:forComponent:* methods to provide the picker view with the width of its components, the row height of every component and the title of every row in each component respectively. Since the picker view used in *FirstPageController* view has only one component, we specify the width of the component (300 float points) to be slightly lesser than the rectangular frame width (320 float points) provided in Listing 6.6. To provide a clear view of the rows in a single component of the picker view, we customize the height of the row to be 50 float points. The picker view makes a call to the *pickerView:widthForComponent:* and *pickerview:rowHeightForComponent:* methods when it needs the component width and the row height to set the layout for displaying the data. However, in order to

actually display the data, it makes a call to *pickerView:titleForRow:* method to provide it with either a string or a view object that could be set as the title of each row in the picker view. In Listing 6.10, a call to *reloadAllComponents* method was made which made the picker view query its delegate for the new data. When *FirstPageController*, the delegate of the picker view is queried for the data, it obtains the number of components and the rows in each component from *numberOfComponentsInPickerView:* and *pickerView:numberOfRowsInComponent:* methods, then obtains the component width and row height by calling *pickerView:widthForComponent:* and *pickerView:rowHeightForComponent:* methods and at last, calls the

*pickerView:titleForRow:forComponent:* method to fill up the picker view from the appropriate array.

The picker view delegates can also implement optional methods like

*pickerView:didSelectRow:inComponent:* to specify if anything should happen when a user interacts with a row within a component in the picker view. The implementation of this method is shown in Listing 6.12. Although the implementation seems to be long, but it is basically doing only two tasks. Firstly, it sets the title of the button control to be the text of the current row selected in the picker view. Secondly, it stores the site information to pass onto the *SecondPageController* class. Lets quickly go through the code for method *pickerView:didSelectRow:inComponent:* in Listing 6.12 via an example. Consider that the first button control corresponding to the *Site Name:* has been chosen. In this case, *which\_button* is set to 1. Hence, first we set the title of *SN\_button* corresponding to the object in *arrayNO* array at the selected row number of the picker view. Then, we store the value of the *site\_description* instance



variable by partially slicing the `NSString` object obtained from *arrayNo* array. Next, based upon the site number and the information available on sites, we decide if the chosen site falls in Washoe or the Clark county. This is saved in *county* instance variable of the *SecondPageController* class. Similarly, the area and the volumen information is saved in *area* and *volume* instance variables respectively. The rest of the cases also perform similar tasks. Note that in this method, we also save the current row selected to get back to the correct row in the picker view once the user moves away from the button control, as was explained at the end of Listing 6.10.

*FirstPageController* class also conformed to the *UIAlertViewDelegate* protocol in its header file. This was required since we needed a customized alert view component in our application which would behave in accordance with the user input. Listing 6.13 provides an example of an alert view component and a method *alertView:clickedButtonAtIndex:* implemented to handle the click of a user on the buttons displayed on an alert view.

Listing 6.13: Presenting an alert and handling it

```

1  [...]
2  /*Calling an Alert view*/
3  if ([second_page_controller does_site_exist]){
4      UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Data already exists"
5          message:@"Data for this site already exists. Delete the previous table and
6          create a new one?" delegate:self cancelButtonTitle:@"NO" otherButtonTitles:
7          @"YES", nil];
8      [alert show];
9      return;
10 }
11
12 /*Handling an Alert view*/

```

```

13 - (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:
14     (NSInteger)buttonIndex
15 {
16     if (buttonIndex == 1) {
17         /*do something*/
18     }
19     else
20         return;
21 }
22 [...]

```

### Explanation of Listing 6.13:

Listing 6.13 provides a part of the code written for *right\_button\_pushed:* method provided in Listing 6.14. In this code snippet, an alert has been called which appears as a blue rectangle in the middle of the screen. It forces us to respond before the application can be continued or exited, based on the user input. In Listing 6.13, we allocate and initialize an alert with a customized title and message displayed on the screen. By passing self as the delegate parameter, we ensure that the *alertView:clickedButtonAtIndex:* method implemented by the delegate of the alert view i.e. *FirstPageController* is called. All alert views have a cancel button, though it can be given any title as can be seen in Listing 6.13. We then create a *Yes* button by assigning the same to *otherButtonTitles* parameter. Note that *otherButtonTitles* can be used to create as many buttons as one wants on an alert view. Lastly, a *show* method is called to show it in the current view.

In our last code snippet of *FirstPageController* class, we provide the method

*right\_button\_pushed*: which is called once the user taps on the *Next* button on the top navigation bar. Since several parts in this method are structurally similar, the code repetitions will be skipped. Listing 6.14 provides the implementation of this method,

Listing 6.14: Implementation of *right\_button\_pushed*: method

```
1  [...]
2  - (void)right_button_pushed:(id)sender
3  {
4      if (SN_answered==NO)
5          /*An alert*/
6      if (CL_answered==NO)
7          /*Another alert*/
8      if (NL_answered==NO)
9          /*Another alert*/
10     if (C_answered==NO)
11         /*Another alert*/
12     if (W_answered==NO) {
13         /*Another alert*/
14     if (TD_answered==NO) {
15         /*Another alert*/
16     if ([second_page_controller does_site_exist]){
17         UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Data already exists"
18             message:@"Data for this site already exists. Delete the previous table and
19             create a new one?" delegate:self cancelButtonTitle:@"NO"
20             otherButtonTitles:@"YES", nil];
21         [alert show];
22         return;
23     }
24     [second_page_controller enter_entry];
25     [[self navigationController] pushViewController:second_page_controller
26         animated:YES];
```

```

27 }
28 - (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:
29     (NSInteger)buttonIndex
30 {
31     if (buttonIndex == 1) {
32         [second_page_controller erase_entry];
33         [second_page_controller enter_entry];
34         [[self navigationController] pushViewController:second_page_controller
35             animated:YES];
36     }
37     else
38         return;
39 }
40 [...]

```

#### Explanation of Listing 6.14:

In Listing 6.14, we provide the implementation of a method *right.button.pushed:* which is called once the *Next* button on the navigation bar is tapped. Although this seems like an action method which should have been connected through the Interface Builder, we provided this connection programmatically by specifying the target and the action in Listing 6.6. In Listing 6.14, we first check if all the button controls have been tapped to store the site information. If this is not true, we implement an alert view with a *nil* delegate, representing that it does not call the *alertView:clickedButtonAtIndex:* method but simply presents an alert to the user. Next, we call a method *does\_site\_exist* implemented in *SecondPageController* to see if the selected site already exists. If a positive answer is returned from this method, we present an alert view with *Yes* and *No* buttons. If the user selects to overwrite

the existing site with the new data by pressing *Yes* button, we execute another method *enter\_entry* from *SecondPageController* class and push the *SecondPageController* onto the navigation stack, which becomes the top view controller and the display is updated with the new view managed by the new controller. At last, we enter into the next page with an animated transition !

### 6.5.6 Modifying SecondPageController

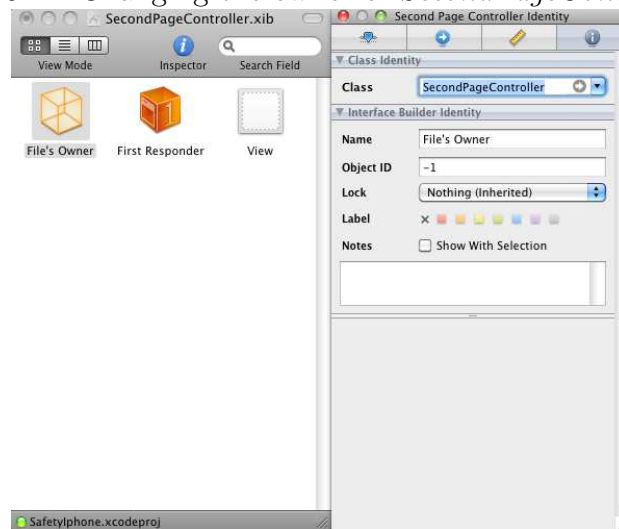
In the above section, we covered the details of how *FirstPageController.xib* was modified and the *FirstPageController* class was implemented. We covered the declaration and implementation of *FirstPageController* in a great detail. In this section, we will skip the details already covered and will focus on presenting the new code specific to *SecondPageController* class.

#### 6.5.6.1 Modifying *SecondPageController* in Interface Builder

In the *Groups and Files* pane, look for *SecondPageController.xib* and double click to open it in Interface Builder. The window opened here would be exactly same to Figure 6.14 except its title. Similar to what we did while modifying *FirstPageController.xib*, we also need to update the configuration of *SecondPageController.xib*, Firstly, we will make the owner of this nib file to be *SecondPageController*. With the *Identity Inspector* opened, single click on the *File's Owner* icon within *SecondPageController.xib* and replace the default *NSObject* class with *SecondPageController*. This makes *SecondPageController* class to be the owner of *SecondPageController.xib* and is shown in Figure 6.21.

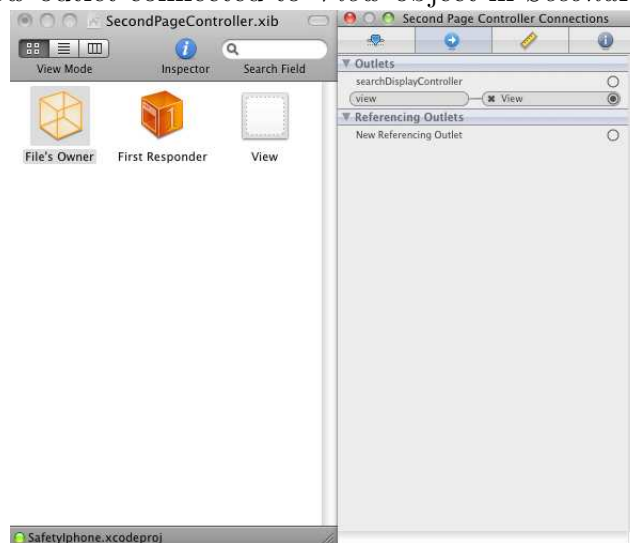
Secondly, we will control drag from the *File's Owner* icon and release the mouse

Figure 6.21: Changing the owner of *SecondPageController.xib*



on the *View* object within *SecondPageController.xib*. From the menu displayed, select the *view* outlet. This outlet connection will allow the *SecondPageController* to manage the *View* object through it's *view* property. The connection between *view* outlet and the *View* object within *SecondPageController.xib* can be seen by bringing up the *Connections Inspector* and single clicking on the *File's Owner* icon. This is shown in Figure 6.22.

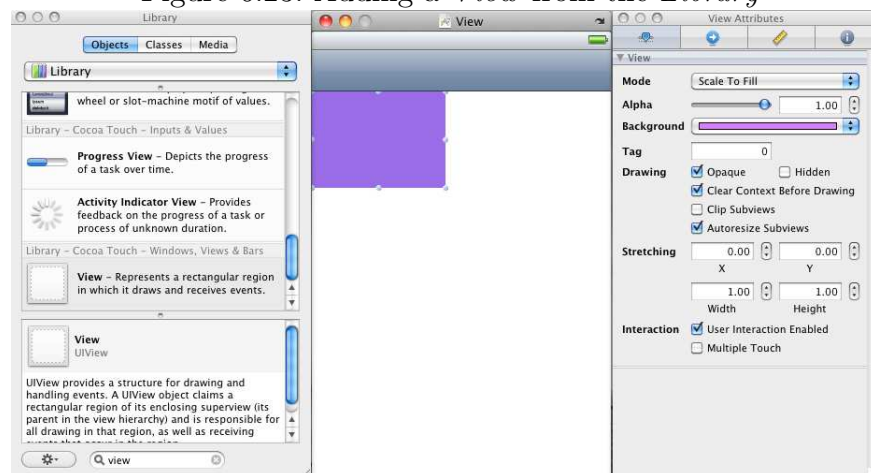
Figure 6.22: *view* outlet connected to *View* object in *SecondPageController.xib*



Next, we will make changes to the *SecondPageController* view. The presented view should allow the user to collect the safety belt usage data for both front seat occupants, their characteristics, the vehicle type and the state of registration of their vehicle. Also, it should allow the user to save the current input and present a similar view for the next vehicle. Similarly, it should allow the user to reset the current screen or pull up the last entry made for performing any entry update. Apart from this functionality, it should also allow the user to navigate back to the first page once the data has been collected for one site. Keeping all this functionality in mind, let's design the graphical user interface of our second page.

First, we add a *Navigation Bar* at the top of the view similar to Figure 6.17. Then, we drag a *View* from the *Library* and place it on our screen as shown in Figure 6.23. We also change its background color, since its default color is white and it's difficult to distinguish between the screen and *View* background.

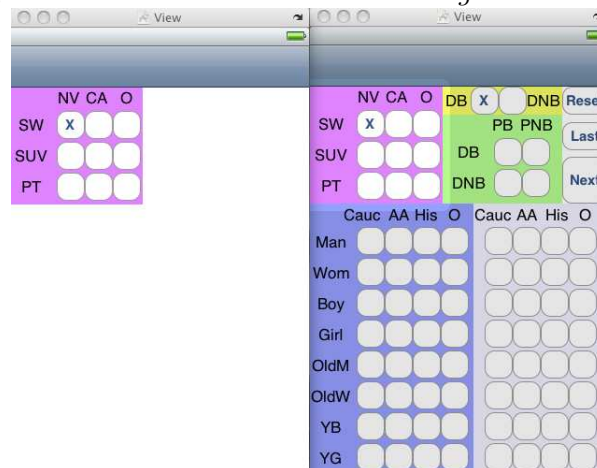
Figure 6.23: Adding a *View* from the *Library*



Next, we drag a round rectangular button from the Library onto the added *View* in Figure 6.23 and set its title to be 'X'. Then, we copy that round rectangular button

object and paste it two more times horizontally next to the original button control. Now, we copy the first row of three button controls and paste the row twice vertically. This saves us time from dragging the button from the Library multiple times and adjusting the size of the buttons every time. Now, set the title of all the buttons except the first original button to be empty (i.e. leave the title of the first original button to be 'X'). Note that if we had first copied the original button, formed the matrix format of 3 by 3 buttons and had later adjusted the title of the first original button to be 'X', the size of the first original button control would have automatically increased due to space limitations. However, the user is advised to follow either approach based on the space limitations on his iOS screen. Furthermore, also drag a few labels and adjust their titles as shown in the left image of Figure 6.24. Repeat this procedure multiple times to create a graphical user interface similar to the right image of Figure 6.24. Note that the backgrounds of all the *View(s)* have been customized and the user is free to select a color accordingly.

Figure 6.24: Final *View* of *SecondPageController.xib*

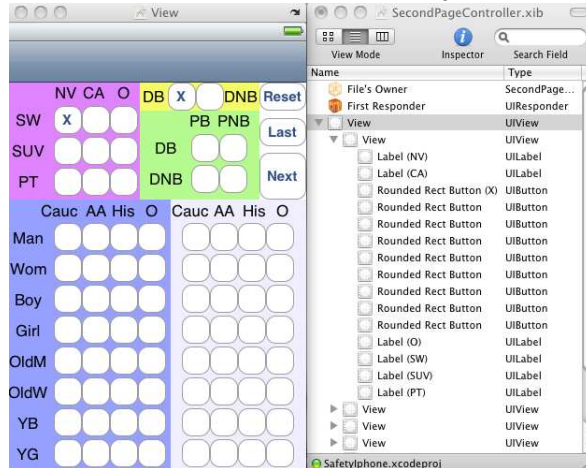


Now, go back to the *SecondPageController.xib* file opened in the *Interface Builder*.



Click the button *View Mode* on the window and the hierarchy of the objects attached to the *View* object of *SecondPageController.xib* can be seen. This is shown in Figure 6.25.

Figure 6.25: Hierarchical structure of objects in View Mode



It is evident from Figure 6.25 that we have added five *View(s)* from the Library to the *View* object of *SecondPageController.xib*. Now, count the five *View(s)* from 1 to 5 in the order of their color: Purple (top left), Yellow (top middle), Green (below Yellow), Dark Blue (bottom left) and Light Blue (bottom right). In other words, assign number 4 to the Dark Blue *View* on our screen. Now, consider a single *View*. For example: Consider the Purple *View*. Assign values 1 to 3 to the columns *NV*, *CA*, *O* respectively. Similarly, assign values 1 to 3 to the rows *SW*, *SUV*, *PT* respectively. In other words, the button marked 'X' within the 3 by 3 matrix of buttons can be said to be at a position (1,1) within that 3 by 3 matrix and hence assign a value of 11 to that button. Concatenate the number assigned to this Purple View (1) in front of the (row)(column) format value and we will get a three digit value for that button marked with 'X' in the Purple View. This number is 111 and is a unique

identifier for this button, and hence will be set as its *tag*. Similarly, the tag value of the button marked 'X' in the Yellow View is 211. Hence, the set of tags assigned to buttons in Purple View are within the set 111, 112, 113, 121, 122, 123, 131, 132, 133. Thus, the minimum tag value possible is 111 and the maximum value possible is 584. The tagging of the button controls throughout the five colored views in such manner makes it easier for us to identify which button was tapped among all the five *View(s)*.

#### 6.5.6.2 Modifying *SecondPageController.h*

In the *Groups and Files* pane of the *SafetyIphone* project, single click on the *SecondPageController.h* file. The code in this file is given in Listing 6.15.

Listing 6.15: SecondPageController.h

```

1  #import <UIKit/UIKit.h>
2  #import <sqlite3.h>
3  @interface SecondPageController : UIViewController<UIAlertViewDelegate> {
4      sqlite3 *database;
5      sqlite3_stmt *statement;
6      UIButton *default_box1;
7      UIButton *default_box2;
8      UIButton *default_box3;
9      UIButton *previous_box1;
10     UIButton *previous_box2;
11     UIButton *previous_box3;
12     UIButton *previous_box4;
13     UIButton *previous_box5;
14     UIButton *saved_previous_box1;
15     UIButton *saved_previous_box2;
16     UIButton *saved_previous_box3;
17     UIButton *saved_previous_box4;
18     UIButton *saved_previous_box5;

```

```

19     bool just_driver_flag;
20     bool saved_just_driver_flag;
21     bool resetter;
22     bool never_been_visited;
23     int site_num;
24     int box_row;
25     int box_column;
26     int site_counter;
27     int primaryKey;
28     BOOL does_exist;
29     BOOL no_entries;
30     NSString *site_description;
31     NSString *date_;
32     NSString *where_collection;
33     NSString *lanes;
34     NSString *conditions;
35     NSString *weather;
36     NSString *drivers_seatbelt;
37     NSString *drivers_age;
38     NSString *drivers_sex;
39     NSString *drivers_race;
40     NSString *passengers_seatbelt;
41     NSString *passengers_age;
42     NSString *passengers_sex;
43     NSString *passengers_race;
44     NSString *state;
45     NSString *vehicle;
46     NSString *area;
47     NSString *volume;
48     NSString *county;
49     NSString *saved_drivers_seatbelt;
50     NSString *saved_drivers_age;
51     NSString *saved_drivers_sex;

```

```

52     NSString *saved_drivers_race;
53     NSString *saved_passengers_seatbelt;
54     NSString *saved_passengers_age;
55     NSString *saved_passengers_sex;
56     NSString *saved_passengers_race;
57     NSString *saved_state;
58     NSString *saved_vehicle;
59     NSString *query;
60 }
61 @property (nonatomic, copy) NSString *vehicle;
62 @property (nonatomic, copy) NSString *state;
63 @property (nonatomic, copy) NSString *drivers_seatbelt;
64 @property (nonatomic, copy) NSString *drivers_age;
65 @property (nonatomic, copy) NSString *drivers_sex;
66 @property (nonatomic, copy) NSString *drivers_race;
67 @property (nonatomic, copy) NSString *passengers_seatbelt;
68 @property (nonatomic, copy) NSString *passengers_age;
69 @property (nonatomic, copy) NSString *passengers_sex;
70 @property (nonatomic, copy) NSString *passengers_race;
71 @property (nonatomic) int box_column;
72 @property (nonatomic) int box_row;
73 @property (nonatomic) int site_num;
74 @property (nonatomic) bool just_driver_flag;
75 @property (nonatomic) BOOL no_entries;
76 @property (nonatomic) bool never_been_visited;
77 @property (nonatomic, copy) NSString *site_description;
78 @property (nonatomic, copy) NSString *date_;
79 @property (nonatomic, copy) NSString *where_collection;
80 @property (nonatomic, copy) NSString *lanes;
81 @property (nonatomic, copy) NSString *conditions;
82 @property (nonatomic, copy) NSString *weather;
83 @property (nonatomic, copy) NSString *area;
84 @property (nonatomic, copy) NSString *volume;

```

```

85  @property (nonatomic, copy) NSString *county;
86  @property(nonatomic, retain) IBOutlet UIButton *default_box1;
87  @property(nonatomic, retain) IBOutlet UIButton *default_box2;
88  @property(nonatomic, retain) IBOutlet UIButton *default_box3;
89  @property(nonatomic, retain) IBOutlet UIButton *previous_box1;
90  @property(nonatomic, retain) IBOutlet UIButton *previous_box2;
91  @property(nonatomic, retain) IBOutlet UIButton *previous_box3;
92  @property(nonatomic, retain) IBOutlet UIButton *previous_box4;
93  @property(nonatomic, retain) IBOutlet UIButton *previous_box5;
94  @property(nonatomic, retain) IBOutlet UIButton *saved_previous_box1;
95  @property(nonatomic, retain) IBOutlet UIButton *saved_previous_box2;
96  @property(nonatomic, retain) IBOutlet UIButton *saved_previous_box3;
97  @property(nonatomic, retain) IBOutlet UIButton *saved_previous_box4;
98  @property(nonatomic, retain) IBOutlet UIButton *saved_previous_box5;
99
100 - (IBAction)button_pressed:(id)sender;
101 - (IBAction)reset_button:(id)sender;
102 - (IBAction)next_button;
103 - (IBAction)last_button;
104 - (IBAction)go_back;
105 - (void)appwillterminate;
106 - (BOOL)does_site_exist;
107 - (void)erase_entry;
108 - (void)enter_entry;
109 @end

```

### Explanation of Listing 6.15:

Listing 6.15 shows the header file of the *SecondPageController* class. First of all, since Sqlite statements will be used in *SecondPageController.m*, *Sqlite.h* has been imported into the class. Next, *SecondPageController* class has been defined to be a

subclass of *UIViewController* class, similar to what was done for *FirstPageController* class. Since *SecondPageController* class uses an alert view in its implementation, the class also conforms to the *UIAlertViewDelegate* protocol. In this header file, a lot of instance variables have been defined. Hence, it is useful to quickly go over the most important variables:

1. *default\_box(i)*, ( $i = 1 \dots 3$ ) buttons:

These variables will be used as fixed outlets to the buttons with *tag(s)* 111, 211 and 311 on the view of *SecondPageController* class respectively. These outlets are defined as IBOutlets on Lines 86-88 in Listing 6.15 and are connected to their respective buttons by control dragging to them from the *File's Owner* icon in *SecondPageController.xib*

2. *previous\_box(i)*, ( $i = 1 \dots 5$ ) buttons:

The variable *previous\_box(i)* is set to clear the previously selected button in *View(i)*, and is afterwards set to the current tapped button in the  $i_{th}$  View.

3. *saved\_previous\_box(i)*, ( $i = 1 \dots 5$ ):

The variables *saved\_previous\_box(i)* save the last tapped buttons for the  $i_{th}$  View for implementing *last\_button* action method in *SecondPageController.m*. This action method is called when the user taps on the *Last* button on the graphical user interface of the *SecondPageController*. Note that variables in Line 49-58 also aid in the same process.

4. *Resetter*:

This variable is set to be true only when the user taps *Last* button in Figure 6.25. Instead of inserting the data into the database, it will update the last database entry.

#### 5. Variables from Line 30-48:

These variables store the data to be inserted into the database.

In the header file, the NSString objects are sent the *copy* message when they are declared as properties. Similarly, the objects declared as IBOutlet are sent *retain* message. However, the low level C datatypes like int, boolean etc. are not sent any messages since they are not Objective C objects. Notice that all the properties here are declared nonatomic as was discussed previously in the explanation of Listing 6.2. Last but not the least, the header file provides the definition of action methods and a few instance methods being used in *SecondPageController.m*. Now, before we dive into the implementation of *SecondPageController* class, let's look at the essentials of *SQLite3* database.

#### 6.5.6.3 Working with SQLite3 database

SQLite3 is an integral part of almost every database oriented application in iOS devices. In Section 6.5.2, we saw how to add *libsqlite3.dylib* library to our project. Also, we imported *sqlite3.h* in *SecondPageController* header file. Now, in order to actually use SQLite database, we first need to create an object of the type *sqlite3*. This was done on Line 4 in Listing 6.15. In Listing 6.16, the code snippet for searching and opening the *safetydata.sqlite* file is provided.

### Listing 6.16: Searching and opening a SQLite database

```
1  NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory ,
2      NSUserDomainMask, YES);
3  NSString *path = [paths objectAtIndex:0];
4  NSString *fullpath = [path stringByAppendingPathComponent:@"safetydata.sqlite"];
5  NSFileManager *fm = [NSFileManager defaultManager];
6  BOOL exists = [fm fileExistsAtPath:fullpath];
7  if (exists)
8      NSLog(@"%@ exists - just opening", fullpath);
9  else{
10     NSLog(@"%@ does not exist - copying and opening", fullpath);
11     NSString *pathForStartingDB = [[NSBundle mainBundle]
12         pathForResource:@"safetydata" ofType:@"sqlite"];
13     NSLog(@"My file: %@", pathForStartingDB);
14     BOOL success = [fm copyItemAtPath:pathForStartingDB toPath:fullpath
15         error:NULL];
16     if (!success)
17         NSLog(@"database copy failed");
18 }
19 const char *cFullPath = [fullpath cStringUsingEncoding:NSUTF8StringEncoding];
20 if (sqlite3_open(cFullPath, &database) != SQLITE_OK)
21     NSLog(@"unable to open database at %@", fullpath);
```

### Explanation of Listing 6.16:

Listing 6.16 provides the implementation of the code required for searching *safetydata.sqlite* in our *Documents* folder. The function *NSSearchPathForDirectoriesInDomains* from the *Foundation* class creates a list of path strings for the *NSDocumentDirectory*, a datatype defined in *NSUtilities.h* which stands for the Document directory. This directory is searched in the user's home directory, specified by *NSUserDomain-*



*Mask* again defined in *NSUtilities.h* , where all the personal items of the users are installed ( $\sim$ ). Since the last parameter in *NSSearchPathForDirectoriesInDomains* function, lets call it *expandTilde*, is set to be *YES*, the full path value will be returned instead of a  $\sim$  character by inherently calling *stringByExpandingTildeInPath* method in *NSString* class. Line 4 in Listing 6.16 helps to find the full path of the *safetydata.sqlite* file. By calling a file manager, we check if the file with a path obtained in Line 4 exists or not. If the file *safetydata.sql* does not exist in the Document directory, then it is searched in the main bundle using an object of the type *NSFileManager*. If the file is found in the main bundle, it is copied to the user directory using *copyItemAtPath:toPath:error:* method from *NSFileManager* class. Note that in this Listing, we have come across *NSLog* method multiple times. *NSLog* method is used to print user defined information at the Xcode console and is useful for debugging errors at the compile time.

Next, we open *safetydata.sqlite* database using the *sqlite3\_open()* C function. Before delving into the pool of *sqlite3* functions, lets have a look at a few result codes returned by various *SQLite* functions. Listing 6.17 provides the result codes as well as their explanations within the comments.

Listing 6.17: Result codes returned by *SQLite* functions (Mark et al., 2011)

```

1  #define SQLITE_OK 0 /* Successful result */
2  #define SQLITE_ERROR 1 /* SQL error or missing database */
3  #define SQLITE_INTERNAL 2 /* Internal logic error in SQLite */
4  #define SQLITE_PERM 3 /* Access permission denied */
5  #define SQLITE_ABORT 4 /* Callback routine requested an abort */
6  #define SQLITE_BUSY 5 /* The database file is locked */

```

```

7  #define SQLITE_LOCKED 6 /* A table in the database is locked */
8  #define SQLITE_NOMEM 7 /* A malloc() failed */
9  #define SQLITE_READONLY 8 /* Attempt to write a readonly database */
10 #define SQLITE_INTERRUPT 9 /* Operation terminated by sqlite3_interrupt()*/
11 #define SQLITE_IOERR 10 /* Some kind of disk I/O error occurred */
12 #define SQLITE_CORRUPT 11 /* The database disk image is malformed */
13 #define SQLITE_NOTFOUND 12 /* NOT USED. Table or record not found */
14 #define SQLITE_FULL 13 /* Insertion failed because database is full */
15 #define SQLITE_CANTOPEN 14 /* Unable to open the database file */
16 #define SQLITE_PROTOCOL 15 /* NOT USED. Database lock protocol error */
17 #define SQLITE_EMPTY 16 /* Database is empty */
18 #define SQLITE_SCHEMA 17 /* The database schema changed */
19 #define SQLITE_TOOBIG 18 /* String or BLOB exceeds size limit */
20 #define SQLITE_CONSTRAINT 19 /* Abort due to constraint violation */
21 #define SQLITE_MISMATCH 20 /* Data type mismatch */
22 #define SQLITE_MISUSE 21 /* Library used incorrectly */
23 #define SQLITE_NOLFS 22 /* Uses OS features not supported on host */
24 #define SQLITE_AUTH 23 /* Authorization denied */
25 #define SQLITE_FORMAT 24 /* Auxiliary database format error */
26 #define SQLITE_RANGE 25 /* 2nd parameter to sqlite3_bind out of range */
27 #define SQLITE_NOTADB 26 /* File opened that is not a database file */
28 #define SQLITE_ROW 100 /* sqlite3_step() has another row ready */
29 #define SQLITE_DONE 101 /* sqlite3_step() has finished executing */

```

Listing 6.18 provides the definitions of SQLite functions that will be used in the implementation of *SecondPageController.m*.

#### Listing 6.18: SQLite function definitions

```

1  int sqlite3_open(const char *filename, sqlite3 **ppDb);
2  int sqlite3_prepare_v2(sqlite3 *db, const char *zSql, int nByte,
3      sqlite3_stmt **ppStmt, const char **pzTail);

```

```

4  int  sqlite3_bind_text(sqlite3_stmt*, int, const char*, int n, void(*)(void*));
5  int  sqlite3_step(sqlite3_stmt*);
6  const unsigned char *sqlite3_column_text(sqlite3_stmt*, int iCol);
7  double  sqlite3_column_double(sqlite3_stmt*, int iCol);
8  int  sqlite3_column_int(sqlite3_stmt*, int iCol);
9  int  sqlite3_reset(sqlite3_stmt *pStmt);
10 int  sqlite3_finalize(sqlite3_stmt *pStmt);
11 int  sqlite3_close(sqlite3 *);

```

### Explanation of Listing 6.18:

The explanation of functions defined in Listing 6.18 is as follows:

1. *sqlite3\_open()*:

This C function opens up an SQLite database. The first parameter *filename* has to be a C string. Therefore, the NSString object *fullpath* in Listing 6.16 was converted to a C string on Line 19 using *cStringUsingEncoding:* method from the *NSString* class. The second parameter *ppDb* in Listing 6.18 contains a handle to the *sqlite* object, which in our case is *database* from Listing 6.15. This function returns an integer 0, represented by *SQLITE\_OK* shown in Listing 6.17.

2. *sqlite3\_prepare\_v2()*:

This function is used to compile SQLite query into a byte-code program, necessary for executing it later. The first parameter *db* represents an open database connection while the second parameter *zSql* contains the SQL statement to be compiled. The third parameter *nByte* decides how many bytes should be read

from the *zSql* statement. Only if it is negative can the *zSql* statement can be read until the first zero terminator. In our project, we will use -1 as its value to read the whole SQL statement. The parameter *\*ppStmt* points to the compiled statement which can be executed using *sqlite3\_step()* function. Lastly, the parameter *\*pzTail*, if NULL, points to the uncompiled statement in *zSql*. On successful execution of this method, it returns 0 (SQLITE\_OK) otherwise an error code is returned.

### 3. *sqlite3\_bind\_text()*:

This function is used to bind values to the prepared statements. The first parameter to this function is the prepared statement received from *sqlite3\_prepare\_v2()* function. The second parameter represents the index of the SQL parameter to be set, where the left most SQL parameter is represented by an index 1. The third parameter to this function represents the value to be bound to the parameter. The fourth parameter, representing the number of bytes in the value, is usually taken to be -1. The last argument is a destructor which is used to destroy the string after SQLite is done with it. If this parameter is set to be SQLITE\_TRANSIENT, SQLite will make a private copy of the data before the *sqlite3\_bind\_text()* returns.

### 4. *sqlite3\_step()*:

This function is called single or multiple times to evaluate a prepared SQL statement received from *sqlite3\_prepare\_v2()*. If 100 is returned (SQLITE\_ROW) by this function, it means that another row is ready to be retrieved. On the

other hand, a return value of 101 (SQLITE\_DONE) represents that the function has finished executing. Note that after 101 has been returned, it is essential that the statement should be reset before it could be used again.

5. *sqlite3\_column\_text()*, *sqlite3\_column\_double()* or *sqlite3\_column\_int()*:

These functions are commonly called as column access functions. In other words, these functions return a single column value of the current row in the result. In any case, the first parameter points to the prepared statement being evaluated while the second parameter represents the column index for which information is required. number of columns in the result set can be retrieved using *sqlite3\_column\_count()* function. As is obvious from the function names, *sqlite3\_column\_text()* returns a zero terminated string, *sqlite3\_column\_double()* returns a double value and *sqlite3\_column\_int()* returns an integer value.

6. *sqlite3\_reset()*:

This function is called to reset a prepared statement to its initial state so that it can be executed again. It returns 0 (SQLITE\_OK) if *sqlite3\_step()* hasn't been called yet or the most recent call to *sqlite3\_step()* returned 100 (SQLITE\_ROW) or 101 (SQLITE\_DONE).

7. *sqlite3\_finalize()*:

This function is an important function implemented to avoid resource leaks. It is used to delete a prepared statement and returns 0 (SQLITE\_OK) if there was no error.

## 8. *sqlite3\_close()*:

As is obvious from this function's name, it closes the connection to an open SQLite database. It returns 0 (SQLITE\_OK) if there was no error while deleting the *sqlite3* object and closing the connection.

### 6.5.6.4 Modifying *SecondPageController.m*

In this section, we provide the details of the implementation of the *SecondPageController* class. Listing 6.19 shows the customization of the *initWithNibName:bundle:* method in *SecondPageController* class.

Listing 6.19: Customizing *initWithNibName:bundle:* method in *SecondPageController* class

```
1  #import "SecondPageController.h"
2  @implementation SecondPageController
3
4  @synthesize box_column, box_row, state, vehicle;
5
6  @synthesize drivers_seatbelt, drivers_age, drivers_sex, drivers_race;
7
8  @synthesize passengers_seatbelt, passengers_age, passengers_sex, passengers_race;
9
10 @synthesize site_num, site_description, date_, where_collection, lanes,
11     conditions;
12
13 @synthesize just_driver_flag, no_entries, never_been_visited, weather;
14
15 @synthesize area, volume, county;
16
17 @synthesize default_box1, default_box2, default_box3;
18
19 @synthesize previous_box1, previous_box2, previous_box3, previous_box4,
20     previous_box5;
21
22 @synthesize saved_previous_box1, saved_previous_box2, saved_previous_box3,
23     saved_previous_box4, saved_previous_box5;
24
25
26 - (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil {
27     self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
28     if (self) {
```

```

19 UIBarButtonItem *go_back = [[UIBarButtonItem alloc] initWithTitle:
20     @"Site Selection" style:UIBarButtonItemStylePlain target:self
21     action:@selector(go_back)];
22 [[self navigationItem] setLeftBarButtonItem:go_back animated:YES];
23 [go_back release];
24 NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, here
25     NSUserDomainMask, YES);
26 NSString *path = [paths objectAtIndex:0];
27 NSString *fullpath = [path stringByAppendingPathComponent:@"safetydata.sqlite"];
28 NSFileManager *fm = [NSFileManager defaultManager];
29 BOOL exists = [fm fileExistsAtPath:fullpath];
30 if (exists)
31     NSLog(@"%@ exists - just opening", fullpath);
32 else
33 {
34     NSLog(@"%@ does not exist - copying and opening", fullpath);
35     NSString *pathForStartingDB = [[NSBundle mainBundle] pathForResource:
36     '    @"safetydata" ofType:@"sqlite"];
37     NSLog(@"My file: %@", pathForStartingDB);
38     BOOL success = [fm copyItemAtPath:pathForStartingDB toPath:fullpath
39     error:NULL];
40     if (!success)
41         NSLog(@"database copy failed");
42 }
43 const char *cFullPath = [fullpath cStringUsingEncoding:NSUTF8StringEncoding];
44 if (sqlite3_open(cFullPath, &database) != SQLITE_OK)
45     NSLog(@"unable to open database at %@", fullpath);
46 }
47 return self;
48 }
49 [...]
50 @endhere

```

### Explanation of Listing 6.19:

In Listing 6.19, *SecondPageController.h* header file is first imported and then the properties defined in the header file are implemented. Inside the *initWithNibName:bundle:* method, a *UIBarButtonItem* object is created to work as a navigation button back to the first page. Then, we search and open up the sqlite3 database file named *safetydata.sqlite* as explained in Listing 6.16.

Listing 6.20 shows a part of the customized implementation of other methods inherited from the *UIViewController* class.

Listing 6.20: Customizing other methods inherited from *UIViewController* class

```
1  - (void) viewDidLoad {
2      drivers_seatbelt = @"Yes";
3      state = @"NV";
4      vehicle = @"Sedan/Station Wagon";
5      drivers_age = @"";
6      /*Initialization of other variables*/
7      [super viewDidLoad];
8  }
9
10 - (void) viewWillAppear:(BOOL) animated
11 {
12     site_counter = 1;
13     [[self navigationItem] setTitle:@" Vehicle Count: 1"];
14     no_entries = YES;
15     never_been_visited = NO;
16     just_driver_flag = YES;
17     [previous_box1 setTitle:@" " forState: UIControlStateNormal];
18     [previous_box2 setTitle:@" " forState: UIControlStateNormal];
19     [previous_box3 setTitle:@" " forState: UIControlStateNormal];
20     [previous_box4 setTitle:@" " forState: UIControlStateNormal];
```



```

21     [previous_box5 setTitle: @" " forState: UIControlStateNormal];
22     previous_box1 = default_box1;
23     previous_box2 = default_box2;
24     [previous_box1 setTitle: @"X" forState: UIControlStateNormal];
25     [previous_box2 setTitle: @"X" forState: UIControlStateNormal];
26 }
27
28 - (void)viewDidLoad {
29     default_box1 = nil;
30     /*Making other outlets nil*/
31     [super viewDidLoad];
32 }
33
34 - (void)dealloc {
35     [default_box1 release];
36     /*Releasing other resources*/
37     [super dealloc];
38 }

```

### Explanation of Listing 6.20:

In Listing 6.20, code snippets from *viewDidLoad*, *viewWillAppear:*, *viewDidLoad* and *dealloc* methods have been provided. In *viewDidLoad* method, additional data initializations are done. Since our data collection template implements default static selection, the variables corresponding to the default selections are initialized as shown in *viewDidLoad* method while rest of the variables are initialized as empty strings. Inside *viewWillAppear:* method, the visual properties of the controls on the screen are changed. For example: the titles of the navigationitem and the button controls are set to provide a user interface with static defaults, ready to collect the safety belt

usage data. Lastly, similar to *FirstPageController* class, *viewDidLoad* and *dealloc* methods are implemented to manage the resources for reuse by the system.

Listing 6.21 shows the implementation of the action method triggered by the button controls when a user interacts with them to select a choice from within the presented five *Views*.

Listing 6.21: Implementation of button\_pressed: action method triggered by five *Views*

```
1  - (IBAction)button_pressed:(id)sender
2  {
3      UIButton *senderButton = (UIButton *)sender;
4      int boxnumber = (int)(senderButton.tag /100);
5      self.box_row = (senderButton.tag /10) % 10;
6      self.box_column = senderButton.tag % 10;
7      switch (boxnumber) {
8          case 1:
9              [previous_box1 setTitle: @" " forState: UIControlStateNormal];
10             [senderButton setTitle: @"X" forState: UIControlStateNormal];
11             previous_box1 = senderButton;
12             switch (self.box_row) {
13                 case 1:
14                     self.vehicle = @"Sedan/Station Wagon";
15                     break;
16                     /*Other cases to find which vehicle was tapped*/
17             }
18             switch (self.box_column) {
19                 case 1:
20                     self.state = @"NV";
21                     break;
22                     /*Other cases to find the state of registration of the vehicle*/
23             }
```

```

24         break; // end of Purple View
25     case 2:
26         just_driver_flag = YES;
27         passengers_seatbelt = @" ";
28         passengers_age = @" ";
29         passengers_sex = @" ";
30         passengers_race = @" ";
31         [previous_box3 setTitle: @" " forState: UIControlStateNormal];
32         [previous_box5 setTitle: @" " forState: UIControlStateNormal];
33         [previous_box2 setTitle: @" " forState: UIControlStateNormal];
34         [senderButton setTitle: @"X" forState: UIControlStateNormal];
35         previous_box2 = senderButton;
36         switch (self.box_column){
37             case 1:
38                 self.drivers_seatbelt = @" Yes";
39                 break;
40                 /*Other cases to find driver's safety belt status*/
41         }
42         break; // end of Yellow View
43     case 3:
44         just_driver_flag = NO;
45         [previous_box2 setTitle: @" " forState: UIControlStateNormal];
46         [previous_box3 setTitle: @" " forState: UIControlStateNormal];
47         [senderButton setTitle: @"X" forState: UIControlStateNormal];
48         previous_box3 = senderButton;
49         switch (self.box_row) {
50             case 1:
51                 self.drivers_seatbelt = @" Yes";
52                 break;
53                 /*Other cases to find driver's safety belt status*/
54         }
55         switch (self.box_column) {
56             case 1:

```

```

57         self.passengers_seatbelt = @"Yes";
58         break;
59         /*Other cases to find passenger's safety belt status*/
60     }
61     break; // end of Green View
62 case 4:
63     [previous_box4 setTitle:@" " forState: UIControlStateNormal];
64     [senderButton setTitle:@"X" forState: UIControlStateNormal];
65     previous_box4 = senderButton;
66     switch (self.box_row) {
67         case 1:
68             self.drivers_age = @"20-60";
69             self.drivers_sex = @"Male";
70             break;
71             /*Other cases to find age and gender of the driver*/
72     }
73     switch (self.box_column) {
74         case 1:
75             self.drivers_race = @"Caucasian";
76             break;
77             /*Other cases to ethnicity of the driver*/
78     }
79     break; // end of Dark Blue View
80 case 5:
81     [previous_box5 setTitle:@" " forState: UIControlStateNormal];
82     [senderButton setTitle:@"X" forState: UIControlStateNormal];
83     previous_box5 = senderButton;
84     just_driver_flag = NO;
85     [previous_box2 setTitle:@" " forState: UIControlStateNormal];
86     previous_box3 = default_box3;
87     [previous_box3 setTitle:@"X" forState: UIControlStateNormal];
88     self.drivers_seatbelt = @"Yes";
89     self.passengers_seatbelt = @"Yes";

```

```

90         switch (self.box_row) {
91             case 1:
92                 self.passengers_age = @"20-60";
93                 self.passengers_sex = @"Male";
94                 /*Other cases to find age and gender of the passenger*/
95                 break;
96         }
97         switch (self.box_column) {
98             case 1:
99                 self.passengers_race = @"Caucasian";
100                break;
101                /*Other cases to find ethnicity of the passenger*/
102            }
103            break; // end of Light Blue View
104        default:
105            break;
106    }
107 }

```

### Explanation of Listing 6.21:

*button\_pressed*: method is triggered when the user interacts with button controls inside Purple, Yellow, Green, Dark Blue or Light Blue Views showed in Figure 6.24. In Section 6.5.6.1, a technique was discussed to allot unique tags to the button controls within these five Views. When this action method is called, the id of the sender button control is sent which allows us to find out which button was touched based upon its unique *tag* attribute. We assign the hundreds, tens and the ones position of the three digit *tag* to *boxnumber*, *self.box\_row* and *self.box\_column* variables respectively. Using the *boxnumber* variable, we can find out which colored View triggered the action

method. For example: if the Purple View triggered the method, the previously marked button control is cleared and an 'X' is marked on the button which was tapped inside the Purple View. Then, using the *box\_row* and *box\_column* variables, the vehicle type and the state of registration of the vehicle are found. Similarly, the safety belt status, age, gender and ethnicity of the driver and the front seat passenger (if present) can be found by touching a control from an appropriate colored View.

After the appropriate selection has been made from the five colored Views, the data needs to be entered into the SQLite database by triggering a *Next* button showed in Figure 6.24. When this button control is touched, an action method named *next\_button* is called which inserts the made selections into the database. The implementation of this action method is shown in Listing 6.22.

Listing 6.22: Implementation of *next\_button* action method triggered by *Next* button

```

1  - (IBAction)next_button
2  {
3      if (previous_box4.currentTitle != @"X")
4          /*Implement an alert to ask to fill the complete information for driver*/
5          /*Other alerts checking if driver and passenger(if present) information
6          has been provided */
7          saved_drivers_seatbelt = drivers_seatbelt;
8          saved_drivers_age = drivers_age;
9          /*Save information for last_button method*/
10         saved_previous_box1 = previous_box1;
11         /*Save states of other marked/unmarked buttons in five colored Views*/
12         NSString *two;
13         if (!resetter) {
14             site_counter = site_counter++;

```

```

15     two= [NSString stringWithFormat:@"insert into site%d (driversb, driverage,
16         driversex, driverrace, passengersb, passengerage, passengersex,
17         passengerrace, state, vehicle) Values(?, ?, ?, ?, ?, ?, ?, ?, ?, ?)",
18         site_num];
19 }
20 else
21     two= [NSString stringWithFormat:@"update site%d set driversb=?, driverage=?,
22         driversex=?, driverrace=?, passengersb=?, passengerage=?, passengersex=?,
23         passengerrace=?, state=?, vehicle=? where entry_id = ?", site_num];
24 NSString *count = [NSString stringWithFormat:@"Vehicle Count: %i ",
25     site_counter];
26 [[self navigationItem] setTitle:count];
27 const char *sql = [two cStringUsingEncoding:NSUTF8StringEncoding];
28 if(sqlite3_prepare_v2(database, sql, -1, &statement, NULL) != SQLITE_OK)
29     NSLog(@"Error while creating first insertion statement. '%s'",
30         sqlite3_errmsg(database));
31 sqlite3_bind_text(statement, 1, [self.drivers_seatbelt UTF8String], -1,
32     SQLITE_TRANSIENT);
33 sqlite3_bind_text(statement, 2, [self.drivers_age UTF8String], -1,
34     SQLITE_TRANSIENT);
35 sqlite3_bind_text(statement, 3, [self.drivers_sex UTF8String], -1,
36     SQLITE_TRANSIENT);
37 sqlite3_bind_text(statement, 4, [self.drivers_race UTF8String], -1,
38     SQLITE_TRANSIENT);
39 sqlite3_bind_text(statement, 5, [self.passengers_seatbelt UTF8String], -1,
40     SQLITE_TRANSIENT);
41 sqlite3_bind_text(statement, 6, [self.passengers_age UTF8String], -1,
42     SQLITE_TRANSIENT);
43 sqlite3_bind_text(statement, 7, [self.passengers_sex UTF8String], -1,
44     SQLITE_TRANSIENT);
45 sqlite3_bind_text(statement, 8, [self.passengers_race UTF8String], -1,
46     SQLITE_TRANSIENT);
47 sqlite3_bind_text(statement, 9, [self.state UTF8String], -1,

```

```

48     SQLITE_TRANSIENT);
49     sqlite3_bind_text(statement, 10, [self.vehicle UTF8String], -1,
50     SQLITE_TRANSIENT);
51     if (resetter)
52         sqlite3_bind_int(statement, 11, (site_counter - 1));
53     if (SQLITE_DONE != sqlite3_step(statement)){
54         if (resetter)
55             NSLog(@"Error while updating data. '%s'", sqlite3_errmsg(database));
56         else
57             NSLog(@"Error while inserting. '%s'", sqlite3_errmsg(database));
58     }
59     sqlite3_reset(statement);
60     if (resetter)
61         resetter = NO;
62     [self reset_button:drivers_age];
63     no_entries = NO;
64 }

```

### Explanation of Listing 6.22:

When the *Next* button on Figure 6.24 is touched, the *next\_button* action method implemented in *SecondPageController* class is triggered. This method first checks if complete information about driver and passenger (if present), has been provided. The required information includes: Safety belt status, age, gender and ethnicity of the driver, vehicle type and the state of registration of the vehicle. If the information provided is not complete, the user is notified through an alert view and the faulty entry is not inserted into the database. However, if all the checking is criteria is met, the current variables and the states of the marked/unmarked button controls within the five colored Views are saved. Next, through the *resetter* vari-



able, it is checked if the user has requested to update the last faulty entry or to insert a new record into the database. In both the cases, an appropriate SQL query is created and the *NSString* variable carrying the query is converted to a C string as explained in Listing 6.18. Then, the SQL query is compiled into a byte-code program by calling *sqlite3\_prepare\_v2()* function and if there is any error, an error message is printed through *NSAssert1* defined in *Foundation* class. However, if there was no error (i.e. if *SQLITE\_OK* was returned), then the selected values (safety belt status, age etc.) are bound to the SQL parameters using the *sqlite3\_bind\_text()* and *sqlite3\_bind\_int()* functions. Then, the query is executed using the *sqlite3\_step()* function and if *SQLITE\_DONE* is returned, the statement is reset to be used again. Finally, the page is reset for the next vehicle by calling the *reset\_button:* method implemented in this class. Note that all the SQLite functions used here have been explained in detail under Listing 6.18.

In Listing 6.23, short code snippets from the implementation of *reset\_button:* and *last\_button* methods triggered by tapping *Reset* and *Last* buttons has been presented.

Listing 6.23: Implementation of *reset\_button:* and *last\_button* action methods

```

1  - (IBAction)reset_button:(id)sender
2  {
3      drivers_seatbelt = @"Yes";
4      state = @"NV";
5      vehicle = @"Sedan/Station Wagon";
6      /*Re-initialize other variables*/
7      [previous_box1 setTitle: @"" forState: UIControlStateNormal];
8      /*Clear other currently marked button controls*/
9      previous_box1 = default_box1;

```

```

10     previous_box2 = default_box2;
11     [previous_box1 setTitle: @"X" forState: UIControlStateNormal];
12     [previous_box2 setTitle: @"X" forState: UIControlStateNormal];
13 }
14 - (IBAction)last_button
15 {
16     if (no_entries)
17         return;
18     [previous_box1 setTitle: @" " forState: UIControlStateNormal];
19     /*Clear other marked buttons*/
20     previous_box1 = saved_previous_box1;
21     /*Set the other pointers to the address of saved_previous_box(i) controls*/
22     [previous_box1 setTitle: @"X" forState: UIControlStateNormal];
23     [previous_box4 setTitle: @"X" forState: UIControlStateNormal];
24     if (saved_just_driver_flag)
25         [previous_box2 setTitle: @"X" forState: UIControlStateNormal];
26     else {
27         [previous_box3 setTitle: @"X" forState: UIControlStateNormal];
28         [previous_box5 setTitle: @"X" forState: UIControlStateNormal];
29     }
30     drivers_seatbelt = saved_drivers_seatbelt;
31     /*Initialize other variables from saved variables*/
32     resetter = YES;
33 }

```

### Explanation of Listing 6.23:

When the *reset\_button:* method is triggered by *Reset* button control in Figure 6.24, all the variables are re-initialized to reflect the same state the variables were in when the this page was first entered. Except a few static default buttons marked 'X' and the corresponding variables initialized appropriately, all other variables are initialized

as empty NSString objects. Next, if the user taps on the *Last* button, *last\_button* action method is triggered. This method sets the *resetter* variable to *YES*, which is later used in *next\_button* method. In this method, the currently marked button controls are cleared and the previously saved marked button controls are re-marked. Also, the saved variables are re-loaded into the current variables and the stage is prepared for updating the last entered record into the database.

In the Listing 6.14, *does\_site\_exist*, *enter\_entry* and *erase\_entry* methods were introduced and it was stated that their implementation will be explained later. These methods check if the data for a particular site already exists and requires user feedback either to cancel the transaction or delete the previous data and create a new entry in its place. Listing 6.24, the last listing of *SecondPageController* class, presents the implementation of these three methods.

Listing 6.24: Implementation of *does\_site\_exist*, *enter\_entry* and *erase\_entry* methods

```

1  - (BOOL) does_site_exist
2  {
3      does_exist = NO;
4      char *cQuery = "select primary_key from siteinfo where SiteNum = ? and
5          DateKiValue LIKE ?";
6      if (sqlite3_prepare_v2(database, cQuery, -1, &statement, NULL) != SQLITE_OK)
7          NSLog(@"Error while selecting primary key in function does site exist.
8              '%s'", sqlite3_errmsg(database));
9      sqlite3_bind_int(statement, 1, self.site_num);
10     sqlite3_bind_text(statement, 2, [[[self.date_ substringToIndex:8]
11         stringByAppendingString:@"%"] UTF8String], -1, SQLITE_TRANSIENT);
12     while (sqlite3_step(statement) == SQLITE_ROW) {

```

```

13     primaryKey = sqlite3_column_int(statement, 0);
14     does_exist = YES;
15 }
16 sqlite3_reset(statement);
17 if (does_exist == YES)
18     return YES;
19 return NO;
20 }
21
22 - (void)erase_entry
23 {
24     const char *sql = "delete from siteinfo where primary_key = ?";
25     if(sqlite3_prepare_v2(database, sql, -1, &statement, NULL) != SQLITE_OK)
26         NSLog(@"Error while deleting from siteinfo. '%s'",
27               sqlite3_errmsg(database));
28     sqlite3_bind_int(statement, 1, primaryKey);
29     if (SQLITE_DONE != sqlite3_step(statement))
30         NSLog(@"Error while executing deletion from siteinfo. '%s'",
31               sqlite3_errmsg(database));
32     sqlite3_reset(statement);
33     NSString *two= [NSString stringWithFormat:@"delete from site%d", site_num];
34     const char *sql2 = [two cStringUsingEncoding:NSUTF8StringEncoding];
35     if(sqlite3_prepare_v2(database, sql2, -1, &statement, NULL) != SQLITE_OK)
36         NSLog(@"Inside erase_entry: Error while deleting from site. '%s'",
37               sqlite3_errmsg(database));
38     if (SQLITE_DONE != sqlite3_step(statement))
39         NSLog(@"Error while executing deletion from site. '%s'",
40               sqlite3_errmsg(database));
41     sqlite3_reset(statement);
42 }
43
44 - (void)enter_entry
45 {

```

```

46     const char *sql = "insert into siteinfo (SiteNum, SiteName, WhereCollected,
47         Lanes, Conditions, Weather, DateKiValue, Area, Volume, County) Values(?, ?,
48         ?, ?, ?, ?, ?, ?, ?)";
49     if(sqlite3_prepare_v2(database, sql, -1, &statement, NULL) != SQLITE_OK)
50         NSLog(@"Error while inserting new entry into siteinfo the first time.
51             '%s'", sqlite3_errmsg(database));
52     sqlite3_bind_int(statement, 1, self.site_num);
53     sqlite3_bind_text(statement, 2, [self.site_description UTF8String], -1,
54         SQLITE_TRANSIENT);
55     sqlite3_bind_text(statement, 3, [self.where_collection UTF8String], -1,
56         SQLITE_TRANSIENT);
57     sqlite3_bind_text(statement, 4, [self.lanes UTF8String], -1,
58         SQLITE_TRANSIENT);
59     sqlite3_bind_text(statement, 5, [self.conditions UTF8String], -1,
60         SQLITE_TRANSIENT);
61     sqlite3_bind_text(statement, 6, [self.weather UTF8String], -1,
62         SQLITE_TRANSIENT);
63     sqlite3_bind_text(statement, 7, [self.date_ UTF8String], -1,
64         SQLITE_TRANSIENT);
65     sqlite3_bind_text(statement, 8, [self.area UTF8String], -1,
66         SQLITE_TRANSIENT);
67     sqlite3_bind_text(statement, 9, [self.volume UTF8String], -1,
68         SQLITE_TRANSIENT);
69     sqlite3_bind_text(statement, 10, [self.county UTF8String], -1,
70         SQLITE_TRANSIENT);
71     if (SQLITE_DONE != sqlite3_step(statement))
72         NSLog(@"Error while inserting. '%s'", sqlite3_errmsg(database));
73     sqlite3_reset(statement);
74 }

```

### Explanation of Listing 6.24:

The creation and execution of the SQL statements in all the three methods pro-

vided in Listing 6.24 are simple and the details of these statements can be found under Listing 6.18.

When the user selects a site on Page 1 shown in Figure 6.20, the *does\_site\_exist* method is called which returns *YES* if the data exists for the selected site on the same date. This method checks for the primary key in a table named *siteinfo* and if it matches the site number for the same date, *YES* is returned otherwise *NO* is sent back as a response to the *FirstPageController* method *right\_button\_pushed:*.

If *YES* is returned back from a call to the method *does\_site\_exist*, an alert is displayed to the user with options to either stay on the same page or delete the existing entry and start fresh data collection on that site. In case the user selects to delete the existing entry, *erase\_entry* method is called which has a void return type. In this method, two SQL queries are created to first delete the existing entry (with site information) from *siteinfo* table and then drop the table containing its data.

After the site information has been deleted from *siteinfo* and the data for the site has been dropped, *right\_button\_pushed:* method from the *FirstPageController* class makes a call to the *enter\_entry* method from the *SecondPageController* class. This method, similar to *erase\_entry* method, also has a void return type. In this method, the site information for the selected site is entered into *siteinfo* table and the control is returned back to the *right\_button\_pushed* method. Back in this method, the view controller for the second page is pushed onto the navigation controller and the associated view is displayed to the user with the database open and ready to be filled with user observation entries. With this, we finish the chapter on iPhone application development for Daytime Safety Belt Usage Studies.

## CHAPTER 7

### GETTING FAMILIAR WITH ADOBE FLEX, PHP AND MYSQL

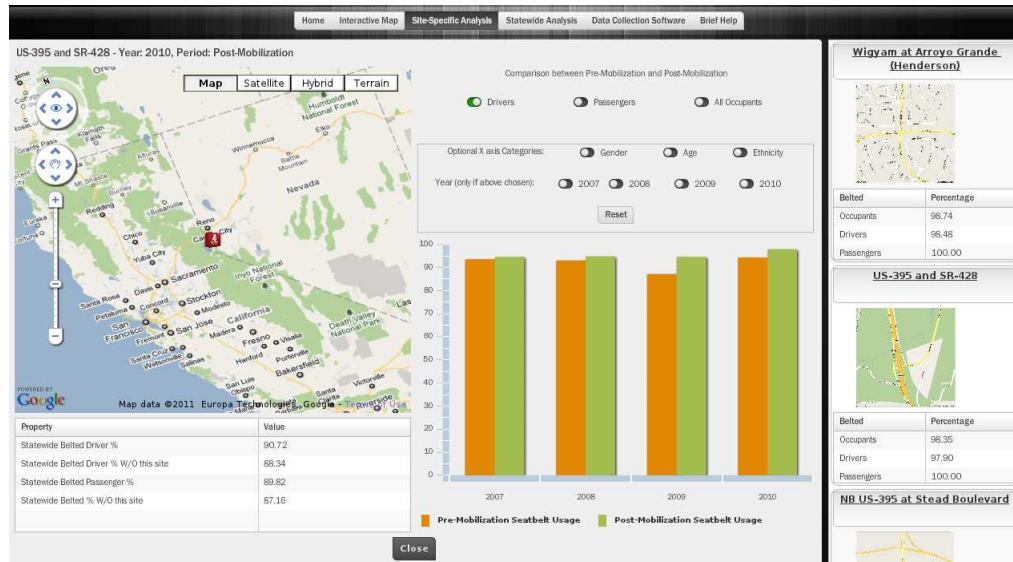
#### Abstract

In Transportation studies, data collection is the backbone of almost every project. The research done, based upon the data collection, can be published in the form of reports. However, all these efforts need to be duplicated in case anyone else wants to take one step ahead in the project. To save time and increase throughput, it is essential that all the data collected be available online and comparative graphical results be available on the fly. With this view in mind, a free of cost data visualization software was developed for the Daytime Safety Belt studies. This software was developed using Adobe Flex, PHP and MySQL. Although building this software application was not difficult, however, new developers can easily get lost in the complexities of these three platforms. Thus, it is necessary to introduce a few fundamental concepts that were used while building the software application. This chapter presents the installation details of the required softwares, provides some code snippets as an introduction to the PhpMyadmin graphical user interface for database queries and a brief summary of the Adobe Flex components used in the application. Figure 7.1 shows a screenshot of the final application.

#### 7.1 Adobe Flex: What and Why ?

Nowadays, there is a boom of rich internet applications on the web. Every website that hosts interesting, enticing content has loads of Javascript and DHTML coding behind it to make the content interactive and dynamic. However, going through such

Figure 7.1: Screenshot of the final application



(a) Detailed Analysis of a site



(b) Statewide Graphical Analysis



a development process can be tiring and time consuming.

Adobe Flex takes us to the next level of web application development. With the growing ubiquity of Adobe Flash player, it has become comparatively to develop rich and interactive applications across different platforms. The Adobe Flex SDK allows us to use the components within the extensive resource library and incorporate animations and transitions across states. In Figure 7.1, it can be observed that a simple design has been wrapped into delicious animations and transitions. This is where Adobe Flex is very powerful, and once experienced with playing with the program logic, the ease of use is simply phenomenal.

Furthermore, With the recent growth of applications on mobile devices, Adobe Flex 4.5 is taking huge strides for developing charming applications on Android and iOS devices. With huge resources of online help content on Adobe Flex SDK and active online forums, the development of rich applications through Adobe Flex platform is increasing at a quick pace.

## 7.2 Installation Details

In this section, the installation details of Adobe Flash Builder, WAMP on Windows and Lamp on Linux have been provided.

### 7.2.1 Installation of Adobe Flash Builder

This section provides a summary of the installation instructions of Adobe Flash Builder 4, which has been used in developing the daytime safety belt application that will be shown in the next chapter. It is worth mentioning that quite recently, Adobe Flash Builder 4.5 (codenamed Burrito) has been released which includes the Flex 4.5

SDK (codenamed Hero). Adobe Flash builder 4.5 includes the support for building Flex and Actionscript applications on Google Android, BlackBerry Tablet OS and Apple iOS platforms. The new features in the Burrito release have been well covered by Shorten (2011). Although there are some new features in Adobe Flash Builder 4.5, the installation of Adobe Flash Builder 4 and 4.5 is exactly the same. Thus, we will restrict ourselves to provide installation guidelines to Adobe Flash Builder 4, which has been used in this thesis.

A 60 day trial version of Adobe Flash Builder 4 can be downloaded from the internet (Adobe, 2011f). On this website, two versions are available: Standalone version for Windows/Mac and Eclipse Plug-in version (which is installed on a bundled copy of Eclipse or an existing installation)for Windows/Mac.

#### 7.2.1.1 Installing Flash Builder 4 with stand-alone configuration

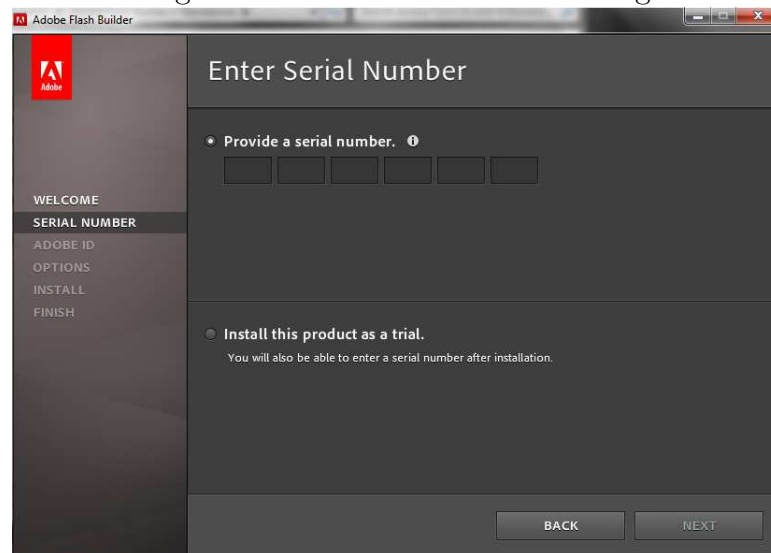
The following components are bundled with the stand-alone version of Adobe Flash Builder 4:

1. Java Runtime Environment (JRE) for Windows
2. Eclipse workbench
3. Flex SDK (3.5 and 4.1)
4. Flash Builder plugin
5. Debug version of Flash Player 10

### **Running the stand-alone installer of Adobe Flash Builder 4**

When the installer is started, a window comes up which asks the user to enter the serial or continue using the trial version. This window is shown in Figure 7.2. A new copy of Adobe Flash Builder 4.5 Premium (currently on Adobe website) costs about US \$699. However, a standard edition of Adobe Flash Builder 4 is available free of cost to university students and faculty members. In fact, faculty or staff members can request upto 30 serial numbers for mutiple machines in the lab environment. For obtaining a free copy of the standard edition, one can register at their website (Adobe Developer Connection, 2011g).

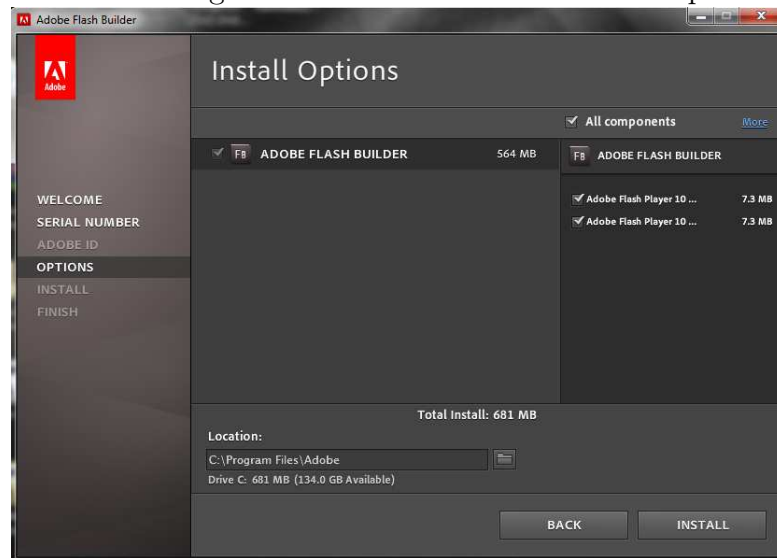
Figure 7.2: Installing Adobe Flash Builder 4: Entering the serial number



After entering the serial number, a new screen will show up which will prompt for the installation folder. Enter the folder name, press next and review the Install Options screen as shown in Figure 7.3.

This concludes the installation process of the stand-alone version of Adobe Flash Builder 4 on Windows/Mac.

Figure 7.3: Installing Adobe Flash Builder 4: Install Options Screen



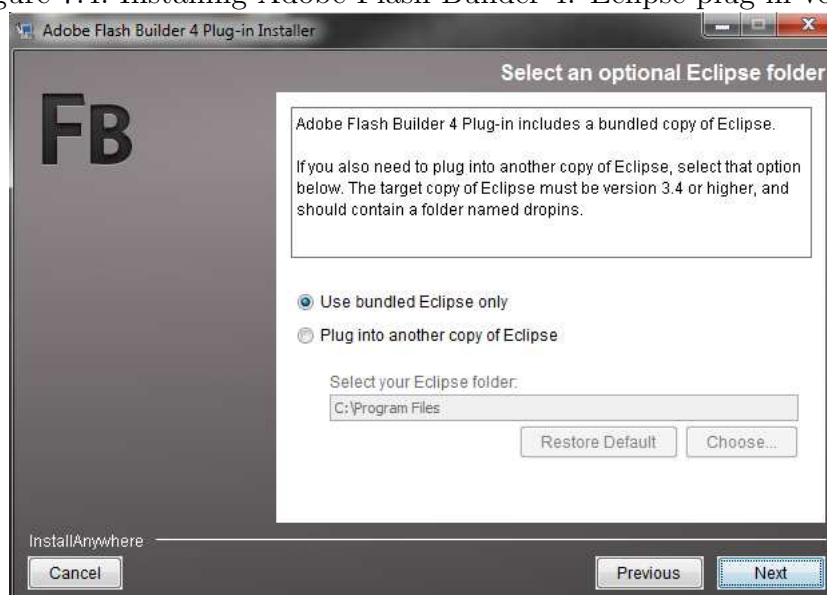
#### 7.2.1.2 Installing the Eclipse plug-in version of Adobe Flash Builder 4

When the Eclipse plug-in installer is double clicked, a prompt will appear which will ask permission to install *installAnywhere* on the computer. Hit yes and wait for the welcome screen to show up. After hitting Next on the welcome screen, the installer will prompt the user to either chose to install the Adobe Flash Builder plugin onto an existing copy of Eclipse or just use the bundled Eclipse which came with the Adobe Flash Builder plug-in installer. This is shown in Figure 7.4.

#### 7.2.2 What is PHP and MySQL

PHP, originally meaning *Personal Home Page* and being called *PHP:Hypertext Preprocessor* , is a scripting language designed for web development to produce dynamic web content. The PHP code is executed on a web server and the result is returned back to the client which is visible through a web browser. With PHP, we have the freedom to choose an operating system: Linux, Unix Variants (HP-UX, Solaris, OpenBSD), Windows, Mac OS X etc. Moreover, PHP has support for most

Figure 7.4: Installing Adobe Flash Builder 4: Eclipse plug-in version



of the web servers including Apache, IIS, and many others. It can not only output HTML, but can also return images, PDF files, XML and even Flash movies. The most significant feature in PHP is that it supports a wide range of databases like MySQL.

MySQL is an open source relational database management system that runs on a web server, providing multi-user access to a number of databases. It is written in C and C++, with its SQL parser being written in yacc. It works on several operating systems namely Linux, Mac OS X, Windows, NetBSD, openSolaris, FreeBSD etc.

PHP and MYSQL go hand in hand for creating fast, and more importantly, free and very easy to use Flex based applications.

### 7.2.3 Installation of WAMP on Windows and LAMP on Linux

Most of the Flex based applications are web and database oriented, hence, it is important that web servers be installed on the local machines. A lot of web servers

are available online: Apache (HTTP), Apache (TomCat), Microsoft's IIS, Nginx, lighttpd, Jigsaw, Klone, Abyss, Oracle, X5 (Xitami), Zeus etc. Since three components are essential to make a Flex based application: PHP, MySQL and a web server, hence the packages which come bundled with all three of them cannot be ignored. In fact, their one click installation with default settings provide quite an attractive option for a general user.

WAMP server is an all-in-one package which allows a general user to download and install *Apache*, *PHP 5.3.5*, *MySQL 5.5.8* and *PhpMyAdmin 3.3.9* with a single click. PhpMyAdmin is a graphical user interface written in PHP through which database queries can be made to the MySQL server. The whole WAMP package is available online (WampServer, 2011). The installation of WAMP is very straight forward and hence will be ignored.

In this section, we will also demonstrate how to install LAMP server on Ubuntu (Linux), which infact stands for Linux, Apache, MySQL and PhpMyAdmin.

1. Open up the Terminal (Applications – > Accessories – > Terminal).
2. Installing lamp-server through terminal:

Type *sudo apt-get install lamp-server^*. This will automatically install Apache, MySQL and PhpMyAdmin on the computer.

3. Manually installing Apache, MySQL and PHP through terminal:

This is an easy, yet a length process. A very good tutorial targetting manual installation of LAMP on Ubuntu is available online (Cargoship, 2007).

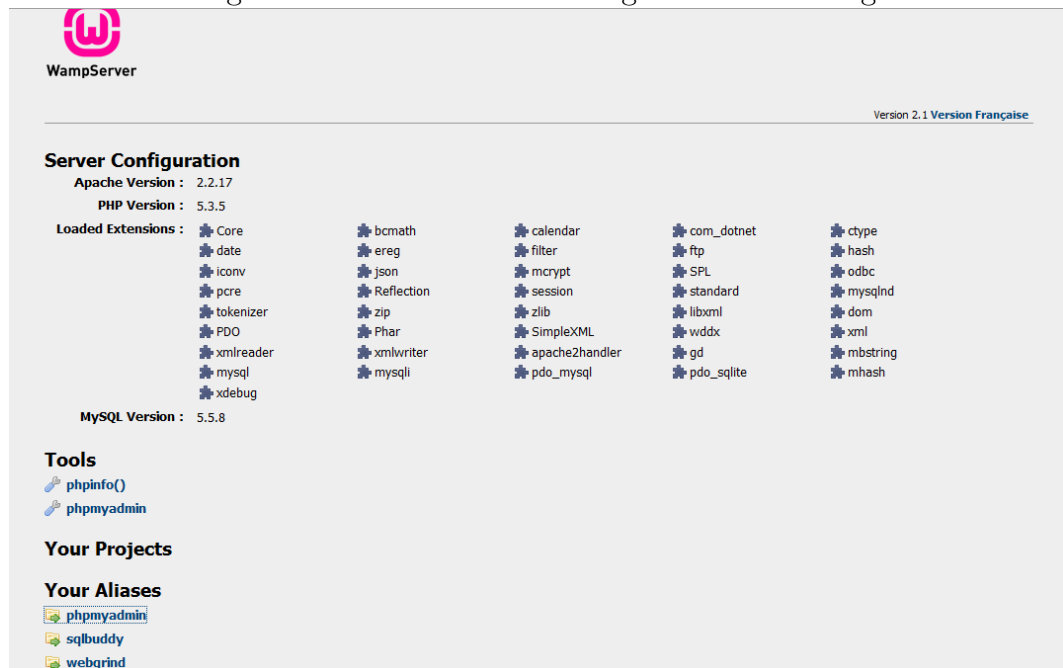
## 7.3 Short tutorial on PhpMyAdmin and MySQL

As mentioned earlier, PhpMyadmin is a very simple Graphical User Interface for making SQL queries to the database. A complete list of SQL commands is available online (w3schools, 2011b). In this section, the basic SQL commands will be demonstrated to give a quick start to the reader. Please follow the following steps to create a database, create a table and finally insert values into the table:

### 1. *Step 1:*

Make sure you have installed WAMP on your computer and have allowed access to it through the firewall. Launch a web browser and go to localhost in the address bar. If something like Figure 7.5 shows up, this means the WAMP server is up and running otherwise repeat Step 1.

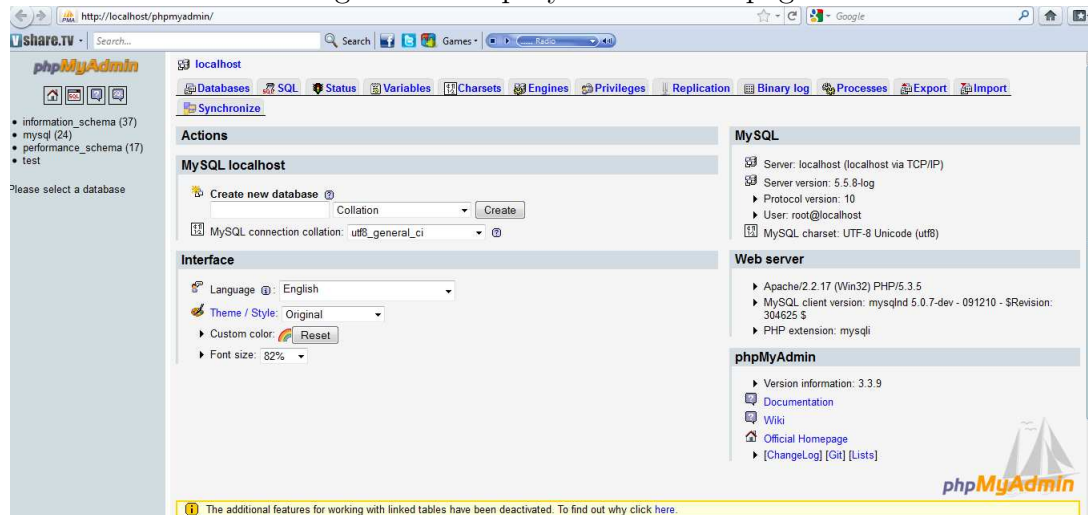
Figure 7.5: WAMP server configuration HomePage



### 2. *Step 2:*

Look closely at the selected link called *phpmyadmin* at the bottom left page of Figure 7.5. Click there and the homepage of PhpMyAdmin will show up, similar to Figure 7.6. This is the Graphical User Interface of PhyMyAdmin, where all the MySQL queries will be executed.

Figure 7.6: PhpMyAdmin Homepage



### 3. Step 3:

On the PhpMyadmin homepage, create a database named *puneet\_test* and thereafter, create a table named *puneet\_test\_table*. You should see an image similar to Figure 7.7.

### 4. Step 4:

After the table has been created, lets insert a few values into the table *puneet\_test\_table* by clicking on the *insert* tab at the top of the page. An image similar to Figure 7.8 will show up.

### 5. Step 5:

Figure 7.9 shows the image of the final table created in MySQL,



Figure 7.7: Creating a table in MySQL

The screenshot shows the phpMyAdmin interface for creating a new table named 'puneet\_test\_table1' in the 'puneet\_test' database. The interface is divided into several sections:

- Field Section:** A table with columns for Field, Type, Length/Values, Default, Collation, Attributes, Null, Index, AUTO\_INCREMENT, and Comments. The fields being created are:
 

Field	Type	Length/Values	Default	Collation	Attributes	Null	Index	AUTO_INCREMENT	Comments
index_key	INT	30	None			<input type="checkbox"/>	PRIMARY	<input checked="" type="checkbox"/>	
Name	VARCHAR	40	None			<input type="checkbox"/>		<input type="checkbox"/>	
College	VARCHAR	40	None			<input type="checkbox"/>		<input type="checkbox"/>	
- Table comments:** A text area for adding comments to the table.
- Storage Engine:** A dropdown menu currently set to 'InnoDB'.
- Collation:** A dropdown menu for selecting the collation.
- PARTITION definition:** A text area for defining table partitions.

At the bottom right, there are buttons for 'Save', 'Or Add 1 field(s)', and 'Go'.

Figure 7.8: Inserting values in the table

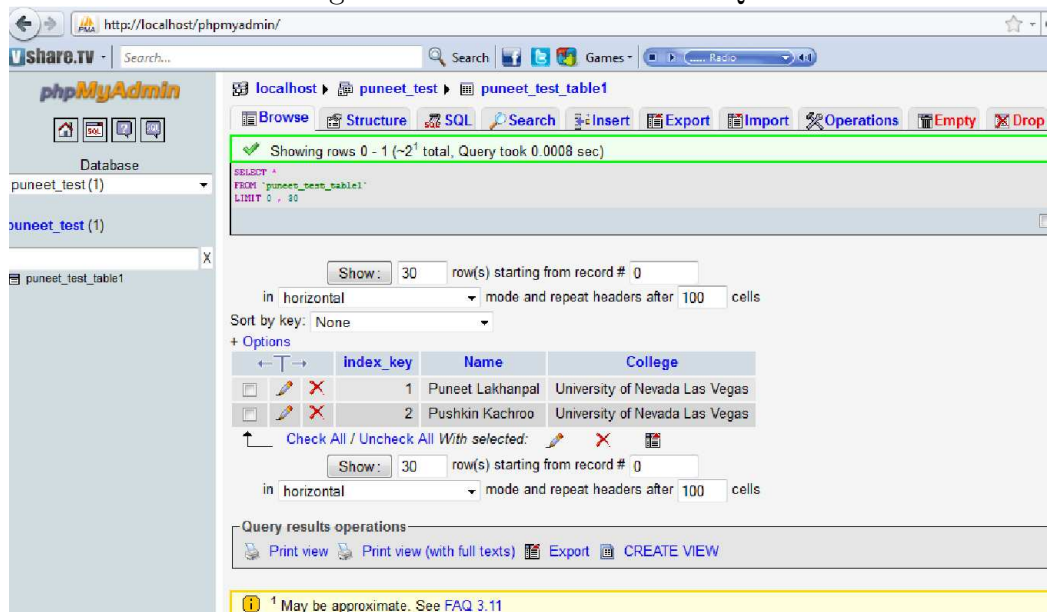
The screenshot shows the phpMyAdmin interface for inserting data into the 'puneet\_test\_table1' table. The interface includes a top navigation bar with tabs for 'Browse', 'Structure', 'SQL', 'Search', 'Insert', 'Export', 'Import', 'Operations', 'Empty', and 'Drop'. The 'Insert' tab is active.

The main area contains two tables for data entry:

Field	Type	Function	Null	Value
index_key	int(11)			
Name	varchar(30)			Puneet Lakhanpal
College	varchar(40)			University of Nevada Las Vegas

Below the first table is a 'Go' button. There is also an 'Ignore' checkbox and a second identical table for inserting another row. At the bottom, there are options to 'Insert as new row' and 'Go back to previous page', along with 'Go' and 'Reset' buttons. A status bar at the bottom indicates 'Restart insertion with 2 rows' and provides a tip: 'Use TAB key to move from value to value, or CTRL+arrows to move anywhere'.

Figure 7.9: Final table in MYSQL



#### 6. Step 6:

Now, let's create an SQL query to show how easy it is to write SQL statements through PhpMyAdmin. Here, we introduce a wildcard '%' to match any missing letters after the given pattern 'Puneet'.

Listing 7.1: Custom SQL Query, introducing the % wildcard

```
1 select * from 'puneet_test_table' where Name LIKE 'Puneet%'
```

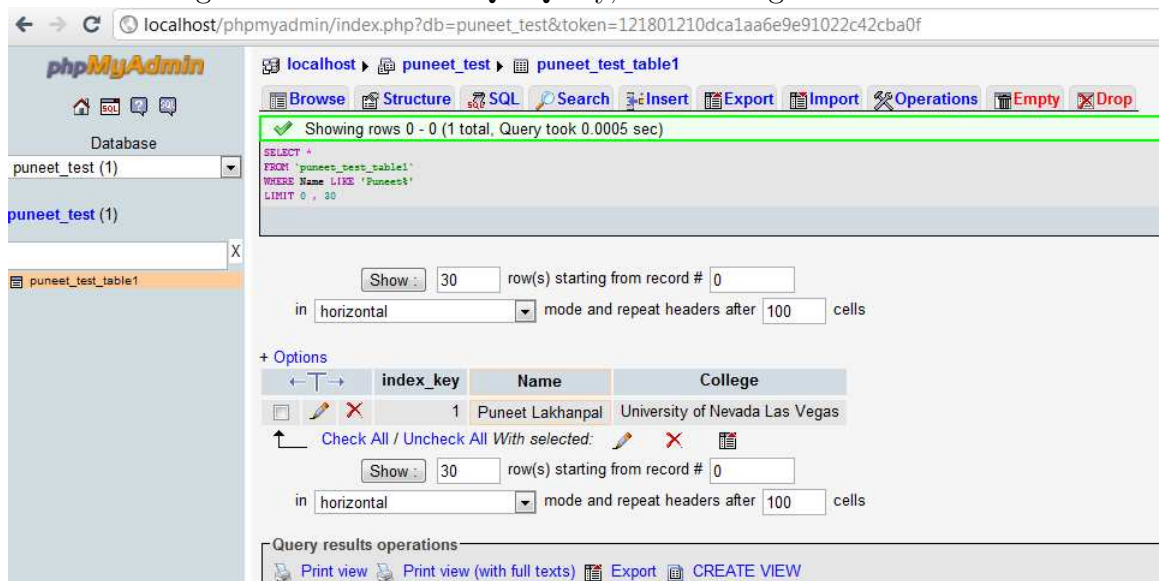
The result set is shown in Figure 7.10.

## 7.4 Fundamentals of Flex 3

### 7.4.1 A review of the components used

A wide variety of Flex components were used in the Interactive Software application. In this section, a brief overview of a few Flex components is given. The code listings demonstrating how these components can be used in Flex applications are

Figure 7.10: Custom SQL Query, introducing the % wildcard



available as Adobe Flex 3 Component Explorer (Adobe, 2011a).

### 1. *Panel:*

The widely used Flex component, Panel, consists of a title bar, border, and a content area for its children. Following are the properties of the Panel that have been used in the application:

- *Layout:*

Three kinds of layouts are possible for the panel: Vertical, absolute and horizontal. If the layout of the panel is mentioned to be vertical, all its child components will be aligned in a vertical manner, originating from the top left corner denoted by (0,0). Horizontal layout of the panel will position its child components from the left to the right of the container. Both the vertical and horizontal layouts provide an automatic positioning of all the child components. However, mentioning an absolute layout of

the panel requires the exact x and y coordinates of the child components on the panel screen.

- *Title:*

As the name suggests, the title property of the panel provides the caption of the panel.

## 2. *ViewStack:*

In Flex, navigator containers organize the user interface into a group of related views so that these views can be navigated sequentially. Viewstack is such a navigator container which maintains the children views in a stack i.e. on the top of one another. At a time, only one view is available and the views can be switched within each other by using the *selectedIndex* property of the view. The index of the ViewStack starts from 0, and hence when an application starts, the zeroth index will be displayed by default. The switching mechanism is not in-built within the ViewStack, hence other controls such as *Linkbar*, *ToggleButtonBar* etc. have to be used to provide this mechanism for the ViewStack.

## 3. *ToggleButtonBar:*

The ToggleButtonBar control provides a horizontal or a vertical arrangement of the the children components, by selecting only one component at a particular time. The ToggleButtonBar switches to some other child component when that component is selected. The ToggleButtonBar component is ideally used in conjunction with the ViewStack component to switch between states between its children components. For such a task, only the *id* property of the viewstack

needs to be assigned to the *dataProvider* property of the ToggleButtonBar component.

#### 4. *ApplicationControlBar*:

The ApplicationControlBar container organizes several components within the Application container, thereby providing global navigation commands. Typically, it is set up at the top of the application container.

#### 5. *VBox and HBox Containers*:

As their name suggests, VBox and HBox are layout containers which align the children components in a predefined vertical or horizontal directions respectively. These layout containers are specifically useful to make the application independent of the sizing changes with respect to the browser or the operating system. In other words, instead of defining the components at specific x and y coordinates within the Flex application, it is sometimes useful to include them within the HBox or VBox layout containers to maintain the spacing within the components. The Seat-belt interactive software described in next chapter widely uses these layout containers to avoid any unexpected changes within the relative sizes of the components.

### 7.4.2 Metadata tags

Metadata tags are inserted into MXML and ActionScript files to provide information to the Adobe Flex compiler. These can be considered as parameters to the compiler, which themselves don't get compiled into the executable code, but only provide information on how to control certain parts of code during compilation. When we specify this metadata tag, the Flex compiler inserts the required code for the specified function while compiling the application. Adobe help resource center website

(Adobe, 2011d) provides a complete list of Metadata tags that can be used in Flex applications. A few of them are: *[Bindable]*, *[Embed]* and *[Event]*. There are two ways of inserting Metadata tags in Flex applications as shown in Listing 7.2:

Listing 7.2: Inserting Metadata tags

```
1 //MXML(Flex 3):
2 <mx:Metadata>
3     [Event(name="enableChange", type="flash.events.Event")]
4 </mx:Metadata>
5
6 //ActionScript(Flex 3):
7 [Event(name="enableChange", type="flash.events.Event")]
8 public class abc extends Event{
9     public function abc():void{
10    }
11 }
```

The way a Metadata tag is inserted into the flex application actually depends upon two factors:

1. As can be seen in Listing 7.2, MXML files can contain the Metadata tags (within `[]`) either inside `<mx:Script>` block or inside `<mx:Metadata>` block. On the other hand, the Metadata tags can be represented only by square brackets `[]` outside the class definition so that it could be bound to the entire class.
2. Inside the MXML files, an important difference between inserting Metadata tags within `<mx:Metadata>` and `<mx:Script>` tags is that the text within the `<mx:Metadata>` tag is inserted before the generated class declaration, but the text within `<mx:Script>` tag is inserted in the body of the generated class

declaration. Hence, the Metadata tags like *[Event]* and *[Effect]* must be written inside `<mx:Metadata>` tag, but the *[Bindable]* and *[Embed]* Metadata tags must be contained within `<mx:Script>` tag.

### 7.4.3 Working with Events

Flex applications are driven by events. In other words, events inform the developers that something has happened within the Flex application. The events can be generated through user interaction (like mouse gestures), or can simply be system events (like completion of page loading). Table 7.1 shows a few common events in Flex.

Table 7.1: A few common Flex events

Event	Constant	Job
change	Event.CHANGE	Occurs when a selection changes in a navigation component
click	MouseEvent.CLICK	Occurs when mouse is pressed down and released on a component
mouseDown	MouseEvent.MOUSE_DOWN	Occurs when mouse button is pressed down
mouseUp	MouseEvent.MOUSE_UP	Occurs when mouse button is released
<i>continued on next page</i>		

<i>continued from previous page</i>		
Event	Constant	Job
mouseOver	MouseEvent.MOUSE_OVER	Occurs when mouse pointer moves over component
mouseOut	MouseEvent.MOUSE_OUT	Occurs when mouse pointer moves out of a component area
rollOver	MouseEvent.ROLL_OVER	Occurs when mouse pointer moves over component
rollOut	MouseEvent.ROLL_OUT	Occurs when mouse pointer moves out of a component
initialize	FlexEvent.INITIALIZE	Occurs when Flex component or application is instantiated
resize	Event.RESIZE	Occurs when component is resized

There is a very subtle difference between *mouseOver*, *rollOver*, *mouseOut* and *rollOut* events. This difference becomes evident when a style is applied to a parent container containing a few children through these events. Consider two Flex panels, each containing one button as their children. Lets say, on the first panel, *rollOver* and *rollOut* events are used to change the style of the parent panel. Simiarly, *mouseOver* and *mouseOut* events are used to change the style of the second panel. In case of the first panel, *rollOut* event will only be fired when the mouse pointer moves out of the panel area and will not take into account the child button of the panel. However,



in the second panel, a *mouseOut* event will be fired when the mouse pointer moves over from the panel to its child button and immediately, a *mouseOver* event will be fired since the mouse now points at the button. The result of these two different approaches will be that the style of the first panel will remain the same, no matter what movements are done within the parent panel. However, in the second panel, switching the movements over the panel and its child button will create flicker of the styles. Listing 7.3 demonstrates this difference, showing the vital part of the code.

Listing 7.3: Difference between RollOver, RollOut, MouseOver and MouseOut events

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="horizontal">
3 <mx:Style>
4     .changeStyle {
5         background-alpha: 0;
6         border-color: #FF0000;
7     }
8 </mx:Style>
9 <mx:Script>
10 <![CDATA[
11 [Bindable] private var overLeftPanel:Boolean = false;
12 [Bindable] private var overRightPanel:Boolean = false;
13 ]]>
14 </mx:Script>
15 <mx:Panel width="50%" height="25%"
16     styleName="{ ( overLeftPanel ) ? 'changeStyle' : '' }"
17     rollOver="{ overLeftPanel = true }"
18     rollOut="{ overLeftPanel = false }">
19 <mx:Button width="100%" />
20 </mx:Panel>
```

```

21 <mx:Panel width="50%" height="25%"
22     styleName="{ ( overRightPanel ) ? 'changeStyle' : '' }"
23     mouseOver="{ overRightPanel = true }"
24     mouseOut="{ overRightPanel = false }">
25 <mx:Button width="100%" />
26 </mx:Panel>
27 </mx:Application>

```

Events can be handled either through the XML way shown in Listing 7.3 or by declaring *addEventListener()* method on the component. The *addEventListener()* has five parameters out of which the first two are required and the rest are optional. The parameters to *addEventListener()* method are as follows:

1. *Type*:

It denotes the type of the event. For example: MouseEvent.CLICK, MouseEvent.DOUBLE\_CLICK etc.

2. *Listener*:

This represents a function which takes the event as the only input argument and process it, without returning anything back.

3. *useCapture*:

This determines whether the listener works in the capture (*useCapture=true*), target or bubbling phase (*useCapture=false*). The default value of *useCapture* is set to false, but to listen for the event in all three phases, *addEventListener* has to be called twice (first with *useCapture=true* and then with *useCapture=false*).

4. *priority*:

This sets the priority of an event listener, with the default value being 0. If two event listeners share the same priority, they will be processed in the order they were added.

5. *useWeakReference* :

If this is set to false, the event listener connected to a `UIComponent` not under use will automatically be garbage collected. This is called as weak reference to the listener. By default, the value of this parameter is set to be true, thereby making a reference to all event listeners strong.

Lastly, it is important to give a brief overview of the event propagation techniques in Flex. There are three different event propagation mechanisms:

1. *Capturing phase*:

In this phase, Flex will search for event listeners in a top to bottom manner, that is from the top of the hierarchy(*application*) to the bottom most event target.

2. *Targeting phase*:

In this phase, only the event listener connected to the event target is only invoked, while not searching for any other event listeners.

3. *Bubbling phase*:

In this phase, Flex searches for event listeners from bottom to top, that is from the event target to the topmost display object (*application*).

In this subsection, we have covered the basic introduction to Flex events. It is worth mentioning that custom events can also be created, which have been covered in the next chapter.

## 7.5 Differences between Flex 3 and Flex 4

Over the existing Flex 3 SDK, Flex 4 brought a few changes in terms of architecture differences, components, layouts, use of states and effects. On the other hand, Adobe made sure that the backward compatibility was maintained with Flex 3. In other words, an Adobe Flex 3 developer will not face too many challenges while compiling Flex 3 applications with Flex 4. Following is the list of changes in Flex 4, a part of which have been extracted from the a blog (Lafferty, 2009).

### 1. **Flash Player:**

Flex 4 applications compile in Flash Player 10, rather than Flash player 9 in Flex 3.

### 2. **Multiple namespaces in Flex 4 CSS:**

With the introduction of Flex 4, additional namespaces have been added to avoid name collisions in MXML.

- *MXML 2006 (Flex 3):*

Function: For compiling Flex 3 applications with Flex 4.

Prefix: mx

URI: <http://www.adobe.com/2006/mxml>

- *MXML 2009 (Flex 4):*

Function: It's just a language namespace, and does not contain component tags.

Prefix: fx

URI: `http://ns.adobe.com/mxml/2009`

- *Spark (Flex 4):*

Function: Includes all new Spark Components. Prefix: s

URI: `library://ns.adobe.com/flex/spark`

- *MX (Flex 4):*

Function: Contains all MX components

Prefix: mx

URI: `library://ns.adobe.com/flex/,x`

Hence, if a CSS file uses type selectors for both MX and Spark Components, here's a sample code that needs to be written:

Listing 7.4: Different namespaces for type selectors in Flex 4

```
1 <fx:Style>
2 @namespace s "library://ns.adobe.com/flex/spark"
3 @namespace mx "library://ns.adobe.com/flex/mx"
4 s|Button{
5     color:#000000
6 }
7 mx|DateChooser{
8     color:#000000
9 }
10 </fx:Style>
```

### 3. Changes in the Default Theme and Preloader:

Flex 3 had a default 'Halo' theme for the look and feel of all the components. With the introduction of Flex 4, Spark theme has become the default theme for all the components, including the Flex 3 components, which can be changed by setting the compatibility mode in the Flex compiler arguments. Additionally, Flex 4 will use the *mx.preloaders.SparkDownloadProgressBar* preloader by default, whereas the Flex 3 users can still use the Flex 3 preloader by mentioning *mx.preloaders.DownloadProgressBar* in the Application tag.

#### 4. Changes in Effects:

In Flex 3 (MX), effects used to work only upon *UIComponent* based controls. With the introduction of Flex 4, the Spark effects can work on any target including MX components, spark components and other graphics primitives.

#### 5. Changes in Layout:

In Flex 3, the controls (for example: *TileList*, *List* etc.) contained their own definitions of layout (for example: absolute, vertical and horizontal) within their respective component classes. However, the layout definitions have been decoupled from the components in Flex 4. This means that instead of the components having a predefined layout, we can ourselves select what layout should be applied to a specific component. Consider the following code snippet Listing 7.5 for a layout example:

Listing 7.5: Different layouts for the same component

```
1 Default Vertical List:
2 <s:List/>
3
4 Horizontal List:
5 <s:List>
```

```

6  <s:layout><s:HorizontalLayout/></s:layout>
7  </s:List>
8
9  Tiled List:
10 <s:List>
11 <s:layout><s:TileLayout/></s:layout>
12 </s:List>

```

## 6. Changes in the syntax of states :

The syntax of the states has been made much simpler in Flex 4, when compared with Flex 3 with the following changes:

- In Flex 4, only states are defined within the states array.
- *includeIn* and *excludeFrom* attributes define a component's role in a particular state, instead of the heavy *AddChild* and *RemoveChild* syntaxes.

Since states for a pivotal part of every Flex application, I developed an application to demonstrate the differences between how states are used in Flex 3 and Flex 4. Listing 7.6 shows the application in Flex 3 and the Flex 4 code for the same application is shown in Listing 7.7. The resulting figures from Listing 7.6 and Listing 7.7 is shown in Figure 7.11.

Listing 7.6: Flex 3 code for states

```

1  <?xml version="1.0" ?>
2  <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
3      minWidth="955" minHeight="600">

```

```

4  <mx:states>
5  <mx:State name="Register">
6  <mx:AddChild relativeTo="{loginForm}" position="lastChild">
7  <mx:target>
8  <mx:FormItem id="confirm" label="Confirm:">
9  <mx:TextInput />
10 </mx:FormItem>
11 </mx:target>
12 </mx:AddChild>
13 <mx:SetProperty target="{loginPanel}" name="title"
14     value="Register" />
15 <mx:SetProperty target="{loginButton}" name="label"
16     value="Register" />
17 <mx:SetStyle target="{loginButton}" name="color"
18     value="blue" />
19 <mx:RemoveChild target="{registerLink}" />
20 <mx:AddChild relativeTo="{spacer1}" position="before">
21 <mx:target>
22 <mx:LinkButton id="loginLink" label="Return to Login"
23     click="currentState='' />
24 </mx:target>
25 </mx:AddChild>
26 <mx:SetEventHandler target="{loginButton}" name="click"
27     handler="trace('changed') />
28 </mx:State>
29 </mx:states>
30 <mx:Panel title="Login" id="loginPanel"
31     horizontalCenter="0">
32 <mx:Form id="loginForm" >
33 <mx:FormItem label="Username:"><mx:TextInput />
34 </mx:FormItem>
35 <mx:FormItem label="Password:"><mx:TextInput />
36 </mx:FormItem>

```



```

37 </mx:Form>
38 <mx:ControlBar>
39 <mx:LinkButton id="registerLink" label="Need to Register?"
40     click="currentState='Register'"/>
41 <mx:Spacer width="100%" id="spacer1"/>
42 <mx:Button label="Login" id="loginButton"
43     click="trace('done');"/>
44 </mx:ControlBar>
45 </mx:Panel>
46 </mx:Application>

```

### Listing 7.7: Flex 4 code for states

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
3     xmlns:s="library://ns.adobe.com/flex/spark"
4     xmlns:mx="library://ns.adobe.com/flex/mx" minWidth="955"
5         minHeight="600">
6     <s:states>
7         <s:State name="State1"/>
8         <s:State name="Register"/>
9     </s:states>
10    <s:Panel id="loginPanel" title.Register="Register"
11        title.State1="Login" horizontalCenter="0">
12        <s:layout><s:VerticalLayout/></s:layout>
13        <mx:Form id="loginForm">
14            <mx:FormItem label="Username:"><mx:TextInput/>
15        </mx:FormItem>
16        <mx:FormItem label="Password:"><mx:TextInput/>
17        </mx:FormItem>
18        <mx:FormItem id="confirm" label="Confirm:"
19            includeIn="Register"><mx:TextInput/>

```

```

20 </mx:FormItem>
21 </mx:Form>
22 <mx:ControlBar width="100%" height="100%">
23 <mx:LinkButton id="registerLink" label="Need to Register?"
24     click="currentState='Register'" excludeFrom="Register"/>
25 <mx:LinkButton id="loginLink" label="Return to Login"
26     excludeFrom="State1" click="currentState=''" />
27 <mx:Spacer width="100%" id="spacer1"/>
28 <mx:Button id="loginButton" label.State1="Login"
29     label.Register="Register" click.State1="trace('done');"
30     click.Register="trace('changed')" color.Register="blue"/>
31 </mx:ControlBar>
32 </s:Panel>
33 </s:Application>

```

Figure 7.11: States demo in Flex 3 and Flex 4

The image shows a login form titled "Login" in a dark blue header. Below the header is a white rectangular area containing two text input fields: "Username:" and "Password:". Below this white area is a dark blue footer containing the text "Need to Register?" and a "Login" button.

(a) Base state in Flex 3

The image shows a login form titled "Login" in a light gray header. Below the header is a white rectangular area containing two text input fields: "Username:" and "Password:". Below this white area is a light gray footer containing the text "Need to Register?" and a "Login" button.

(b) Base state in Flex 4

The image shows a register form titled "Register" in a dark blue header. Below the header is a white rectangular area containing three text input fields: "Username:", "Password:", and "Confirm:". Below this white area is a dark blue footer containing the text "Return to Login" and a "Register" button.

(c) Register state in Flex 3

The image shows a register form titled "Register" in a light gray header. Below the header is a white rectangular area containing three text input fields: "Username:", "Password:", and "Confirm:". Below this white area is a light gray footer containing the text "Return to Login" and a "Register" button.

(d) Register state in Flex 4

## CHAPTER 8

### INTERACTIVE SAFETY BELT DATA VISUALIZATION SOFTWARE

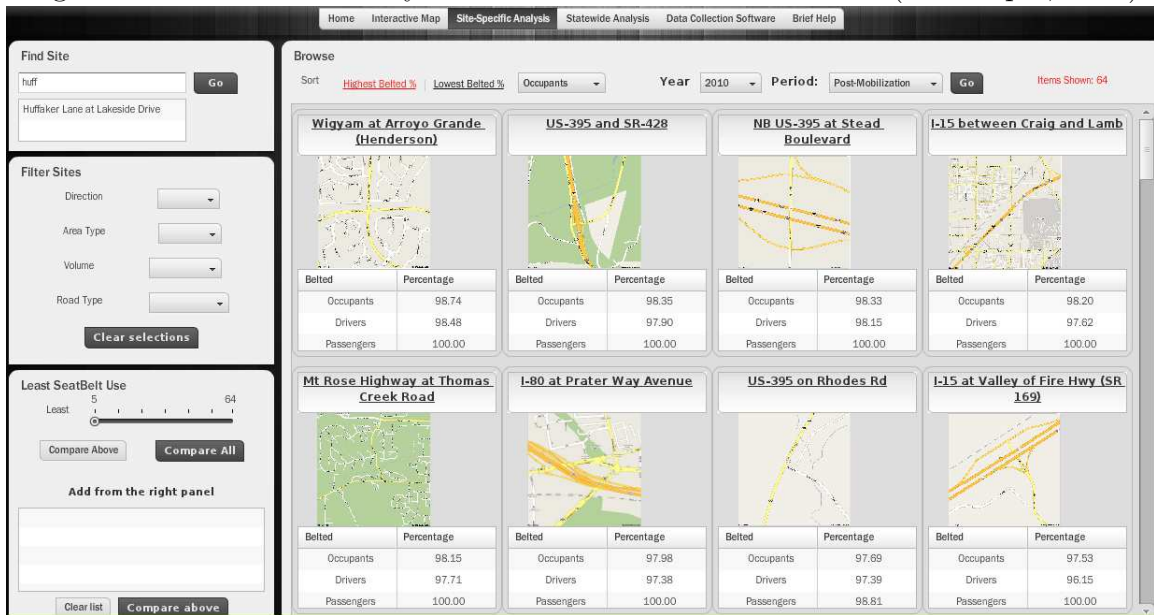
#### Abstract

In Chapter 7, the fundamentals of the three platforms: Adobe Flex, PHP and MySQL was provided. Still, developing a rich application with animations and transitions requires much more than just the basic knowledge of the Adobe Flex and the other frameworks. In this chapter, a few details of the Interactive Safety Belt Data Visualization Software have been provided. This chapter has been written for just one objective in mind: to showcase the essential techniques needed in developing an attractive data oriented application as shown in Figure 8.1. This chapter will be very useful for a person looking to develop such an application within a short period of time. Although the full code is not provided, the important concepts necessary to learn how to build such an application are certainly discussed. The source code of the application has been made available. It can be easily found by right clicking on the application and clicking on *View Source*. The application is available online at the NUTC website (Lakhanpal, 2010).

#### 8.1 Container for all components

As seen on the home page of the application, several tabs can be seen at the top of the page. In the programming literature, it has been mentioned that modularizing the code into several code fragments, where every fragment achieves a single task is a good practice. The same technique has been used in this interactive software application. The software starts with the default Application container by defining a `<mx:Application>` tag. All the subsequent controls or components used within this Application tag will be considered its children. Following the modularized approach, different components were created which are independent of each

Figure 8.1: Interactive Safety Belt Data Visualization Software (Lakhanpal, 2010)



other. Listing 8.1 provides the MXML structure of the components created within the main application tag. All the components created within the default Application had an MXML extension and were termed *homeView*, *mapAnalysis*, *dataAnalysis*, *statewideAnalysis*, *softwareDemo* and *help*. These components will be discussed one by one in this chapter, giving several details of their implementation and the associated events and event handlers. All these components have been stacked within `<mx:ViewStack>` block. A basic introduction to the ViewStack component was provided in Chapter 7 Section 7.4.1. It was also mentioned that the ToggleButtonBar can be used to provide the switching mechanism. Listing 8.1 illustrates the procedure to do so by assigning the *id* of the viewstack to the *dataprovder* property of the ToggleButtonBar component. It is important to note that the *creationPolicy* property of the Viewstack component has been set to *all*. Setting this property to *all* makes sure that the child components are forcefully created up front and not created only when they are accessed, which is the default behaviour of Flex. Additionally, Flex comes with a wide spectrum of animations which make the transition phases from one page to another appealing. Two of such animations named *WipeDown* and *WipeUp* have been demonstrated in Listing 8.1.

### Listing 8.1: MXML structure of the components with the main application

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <mx:ApplicationControlBar id="acb" width="100%">
3   <mx:HBox width="100%">
4     <mx:Spacer width="100%" />
5     <mx:ToggleBarButton height="100%" dataProvider="{Views}" width="100%" />
6     <mx:Spacer width="100%" />
7   </mx:HBox>
8 </mx:ApplicationControlBar>
9 <mx:ViewStack id="Views" width="100%" height="100%" creationPolicy="all">
10   <homeView id="home" label="Home" showEffect="WipeDown" hideEffect="WipeUp" />
11   <mapAnalysis id="map" label="Interactive Map" showEffect="WipeDown"
12     hideEffect="WipeUp" />
13   <dataAnalysis id="sitewise" label="Site-Specific Analysis" showEffect="WipeDown"
14     hideEffect="WipeUp" />
15   <statewideAnalysis id="statewide" label="Statewide Analysis"
16     showEffect="WipeDown" hideEffect="WipeUp" />
17   <softwareDemo id="softwaredemo" label="Data Collection Software"
18     showEffect="WipeDown" hideEffect="WipeUp" />
19   <help id="helpSoftware" label="Brief Help" showEffect="WipeDown"
20     hideEffect="WipeUp" />
21 </mx:ViewStack>
```

#### 8.1.1 Styling the application

A very important ingredient of a tasty-looking Flex application is the customized skinning of the default Skin Components. Flex lets us customize the look and feel of the applications in the following three ways:

1. Using Inline Styles:

Flex provides the functionality to change the default styles of the components

including but not limited to: color, fontFamily, fontSize, fontStyle, fontWeight, textAlign, textDecoration etc. The simplest way to do so is by selecting the component in the Design mode and adjusting the above styles in the Flex Properties window. Alternatively, the inline styling in Flex can also be done through the two approaches shown in Listing 8.2 from the Interactive Software application.

Listing 8.2: Interactive Software: Inline CSS styling

```
1 //MXML code:
2
3 <?xml version="1.0" encoding="utf-8"?>
4 <mx:Panel cornerRadius="10" id="panel3">
5
6 //ActionScript code
7 panel3.setStyle("cornerRadius", 10);
```

## 2. Using External CSS files:

The Flex framework contains a CSS file which defines the default color schemes and the look and feel of the Flex 3 applications. This file is called *default.css* and can be found within Flex SDK. However, Flex also provides the possibility of tweaking the default styles by letting the developers include an external CSS file. In the interactive software application, an external CSS file named *flexskin.css* was used. Listing 8.3 shows the general syntax of the CSS files, and also demonstrates how to include the CSS file within the application.

Listing 8.3: Interactive Software: External CSS styling Syntax

```
1 //Inside CSS file:
2
3 selector_name{
```

```

4     style_property: value;
5     [...]
6 }
7
8 //Including the CSS file within MXML application
9
10 <mx:Style source="/css/flexskin.css"/>

```

Listing 8.4 shows a code segment from the CSS file *flexskin.css*. In this CSS file, a few styles have been applied to all the instances of the TileList component within the interactive software application. A complete list of the CSS styles and properties is available online (w3schools, 2011a).

#### Listing 8.4: Interactive Software: External CSS styling of TileList

```

1 //Inside CSS file:
2 TileList
3 {
4 background-image:Embed("/assets/general/fondo_list.png", scaleGridTop="5",
5     scaleGridBottom="45", scaleGridLeft="5", scaleGridRight="45");
6 background-size:"100%";
7 background-color:#dddddd;
8 border-thickness:1;
9 padding-left:0;
10 padding-right:0;
11 padding-top:0;
12 padding-bottom:0;
13 vertical-gap:0;
14 horizontal-gap:0;
15 selection-color:#707070;

```

```
16  roll-over-color:#999999;  
17  }
```

As is understood from the name of the CSS property *background-image*, it embeds an image named *fondo-list.png* to be the background of the *TileList* components. However, this style can cause unexpected issues if the *TileList* component is wider or taller than the image being embedded, thus having scaling issues. To avoid such issues, Flex has provided the developers with a 9-slice scaling technique which breaks the image being embedded into 9 different sections, each being scaled independently. All text and gradients are scaled normally, however, a few rules apply for the other objects as follows:

- Content in the center region is scaled normally.
- Content in the corners is not scaled.
- Content in the top and bottom regions is scaled only horizontally. Content in the left and right regions is scaled only vertically.
- All fills (including bitmaps, video, and gradients) are stretched to fit their shapes.

These rules have been extracted from the Adobe Help website (Adobe, 2011c) which provides further details about the 9-scale implementation. The website also explains the concept of the four embedding parameters namely, *scaleGridTop*, *scaleGridBottom*, *scaleGridLeft* and *ScaleGridRight*, representing the dis-



tances from the sides of the original, unscaled image as follows:

- *scaleGridTop*:

Specifies the distance in pixels of the upper dividing line from the top of the image.

- *scaleGridBottom*:

Specifies the distance in pixels of the lower dividing line from the top of the image.

- *scaleGridLeft*:

Specifies the distance in pixels of the left dividing line from the left side of the image.

- *scaleGridRight*:

Specifies the distance in pixels of the right dividing line from the left side of the image.

The CSS code written in Listing 8.4 was appropriate for applying the style to all the instances of the `TileList` component. However, many situations arise when we need to apply a style to only a few specific instances of the component. In this case, we would have to define a custom style declaration in CSS and specify the name of the custom style selector in those components' *styleName* property. Listing 8.5 shows the custom CSS style declaration and also illustrates how to include it in the component's MXML declaration. In this code fragment, a few common styles have been defined within this custom CSS declaration. Furthermore, four images have been embedded which represent the four states

of a Flex button component: *up* - the normal state of the button, *over* - when a user bring the mouse over the button, *down* - when a user clicks a button and *disabled* - when the button is disabled. Finally, the name of the custom style selector has been assigned to the specific instance of the button component within its MXML declaration. It should be noted that the name of the CSS custom style selector started with a `'.'`. The `'.'` character defines a class selector in CSS which is used to specify a style for a group of elements, unlike the *id* selector which is defined with a `'#'` and can be used to style only one particular component. More information about the component states can be found online (Adobe, 2011b).

#### Listing 8.5: Interactive Software: External CSS styling

```
1 //Inside CSS file:
2
3 .buttonOfficial
4 {
5   up-skin: Embed("/assets/buttons/button.png", scaleGridTop="6",
6     scaleGridBottom="14", scaleGridLeft="6", scaleGridRight="14");
7   over-skin: Embed("/assets/buttons/button1.png", scaleGridTop="6",
8     scaleGridBottom="14", scaleGridLeft="6", scaleGridRight="14");
9   down-skin: Embed("/assets/buttons/button2.png", scaleGridTop="6",
10     scaleGridBottom="14", scaleGridLeft="6", scaleGridRight="14");
11
12   disabled-skin: Embed("/assets/buttons/button_dis.png", scaleGridTop="6",
13     scaleGridBottom="14", scaleGridLeft="6", scaleGridRight="11");
14   color: #e8e8e8;
15   font-family: franklin1;
16   padding-left: 6;
17   padding-right: 6;
```

```

18  text-roll-over-color: #e8e8e8;
19  text-selected-color: #cccccc;
20  }
21
22  //Inside MXML file
23
24  <mx:Button styleName="buttonOfficial" y="10" label="Go" width="47"
25          right="10" id="button1"/>

```

### 3. Loading stylesheets at runtime:

Flex also allows to change the appearance of the application and its components during the runtime. The images or other assets can be embedded into a stylesheet and loaded at the runtime, thereby decreasing the loading time since the assets are embedded within the stylesheets and not the main application. For this to happen, the developers have to compile the CSS files into a SWF file and use *StyleManager.loadStyleDeclarations* function to load it during runtime. This is shown in Listing 8.6.

#### Listing 8.6: Interactive Software: Runtime CSS styling

```

1  //Remove following from MXML:
2  <?xml version="1.0" encoding="utf-8"?>
3  <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
4  <mx:Style source="/css/flexskin.css"/>
5  </mx:Application>
6
7  //Replace with this code to MXML:
8
9  <?xml version="1.0" encoding="utf-8"?>
10 <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
11     creationComplete="applyStyleSheet()">
12 <mx:Script>

```

```
13 <![CDATA[
14     private function applyStyleSheet():void {
15         StyleManager.loadStyleDeclarations("mystyle.swf");
16     }
17 }]>
18 </mx:Script>
19 <mx:Label text="
```

Before ending this section, it is worth mentioning that Flex provides an amazing online tool to style the components simply by using the inherited style properties.

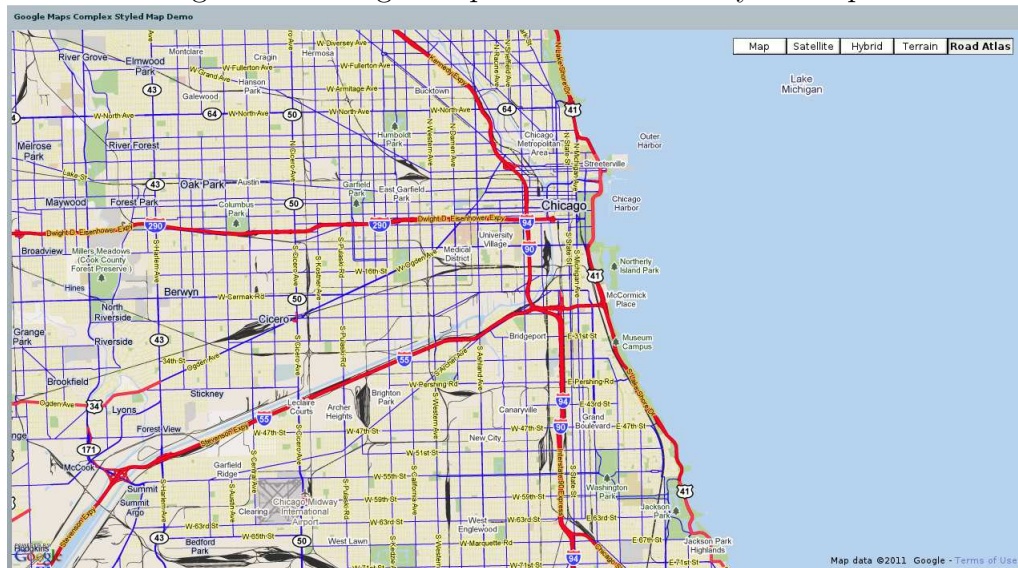
## 8.2 Google Maps API for Flash

The *Google Maps API for Flash* enables a Flex developer to embed robust, interactive Google maps in Flash applications. The following section enlists a few important features of this API.

### 8.2.1 Features in Google Maps

1. Support for 2D maps and 3D Maps
2. Six different types of Maps including:
  - *NORMAL\_MAP\_TYPE*- the default view
  - *SATELLITE\_MAP\_TYPE* - showing Google Earth satellite images
  - *HYBRID\_MAP\_TYPE* - showing a mixture of normal and satellite views
  - *PHYSICAL\_MAP\_TYPE* - showing a physical relief map of the surface of the Earth

Figure 8.2: Google Map API for Flash: Styled Maps

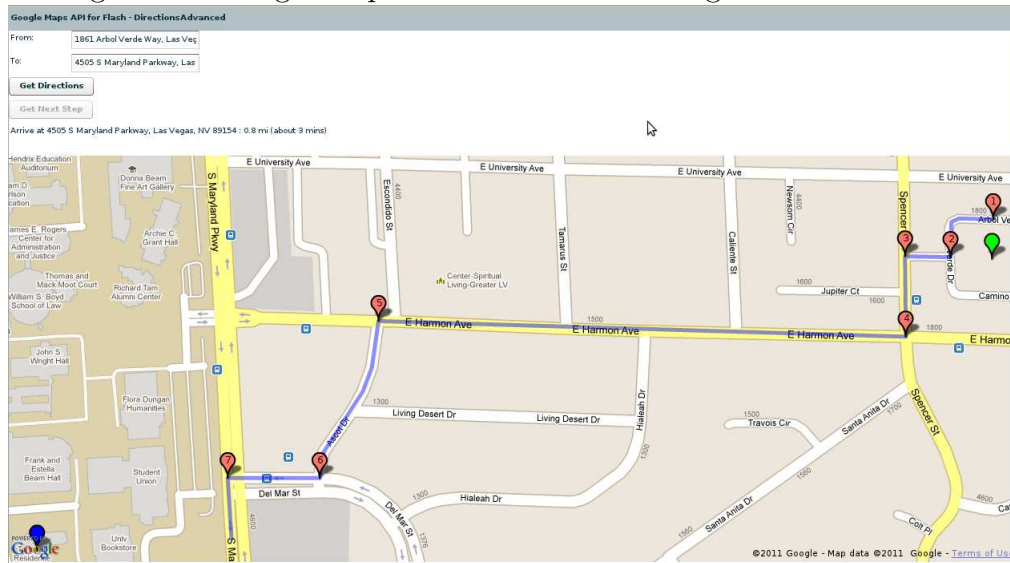


- *DEFAULT\_MAP\_TYPES* - an array of the above four types, useful for iterative processing.
- *StyledMapType*: allowing the developer to customize the visual display of components as roads, parks etc., thereby emphasizing particular components within the map.

Figure 8.2 shows a few capabilities of the Google Maps API, when the *StyledMapType* is used with different combinations of hue, saturation and other style rules. This figure approximates the look of a retro US road atlas centered at Chicago, where several operations and simplifications have been used to complete the task.

3. Able to communicate with other Google Services like GeoCoding, Driving Directions and Elevation. Geocoding refers to the process of converting an address into its geographic latitude and longitude coordinates, thereafter displaying it

Figure 8.3: Google Maps API for Flash: Driving Directions Service



as a marker on the Google Map. Furthermore, the driving directions can be found using the Google Maps API by querying either an address string or simply textual longitudes and latitudes. Both the driving and walking modes are supported for finding the directions from an origin to the destination. Last but not the least, the Elevation service can query the Google servers to return the elevation data for locations on the earth surface, including the depth locations on the ocean floor (representing them by negative values). Figure 8.3 provides a snapshot of the Driving Directions service within the Google Map API for Flash.

4. Customizable *markers*, *icons*, *polylines* and *encoded polylines* on Google Maps, which fall under a category called *overlays*. The overlays are objects tied to the latitude/longitude coordinates which designate specific points, lines or areas on the map. As the name suggests, markers are used to represent points on

the maps using custom icons while a collection of points are represented by polylines. It is worth mentioning that there is a crucial difference between *Polylines* and *Encoded Polylines*. Long and complicated polylines require a fair amount of memory, take longer to draw and draw segments regardless of the resolution at larger zoom levels. On the other hand, the encoded polylines specify a series of points using a compressed format of ASCII characters, along with providing a maximum zoom level for every line segment to draw itself on the Google Map. For example, an encoded polyline representing a drive from Orange County, California to Las Vegas, NV should not care about the line segments representing particular streets on a lower level when the map is zoomed out to the state level. The details of the Encoding algorithm are available online.

While working at Transportation Research Center UNLV, an opportunity came up to build the aforementioned encoding algorithm in PHP for a project. The project involved processing huge chunks of raw IPM data coming from Freeway and Arterial System of Transportation (RTC, 2010). The processed data had to be shown every 15 minutes as a traffic animation on Google Maps with the green, red and yellow color combinations of the encoded polylines, representing the traffic volume numbers in both directions. Since every device sensor had a latitude and longitude associated with it, a few neighbouring latitude/longitude pairs were picked up around that device and the encoding algorithm built in PHP was applied to the dataset shown in Figure 8.4. Note that

$lat/longi$  represented the actual latitude/longitude pair for the device sensor and other latitude/longitude pairs (e.g.  $lat1/long1$ ,  $lat2/long2$  etc.) were the neighbouring coordinates picked up from the map to generate lines. Also,  $difflat/diff-long$  pair was generated by performing  $(lat-lat1)/(longi-long1)$ . Similar argument holds for generating  $difflat1/diff-long1$ . Additionally,  $difflat2/diff-long2$ ,  $difflat3/diff-long3$  and  $difflat4/diff-long4$  have been deleted from Figure 8.4 to focus upon the encoded variables:  $path$  and  $encLevel$ . For every row in Figure 8.4, the generated variable  $path$  represents one encoded string for a set of five latitude/longitude pairs. Hence, this strategy for generating and sending one encoded string instead of five latitude/longitude pairs greatly improve the speed of application. The PHP encoding algorithm is presented in Listing 8.7 and the generated traffic animation with encoded polylines is shown in Figure 8.5.

Listing 8.7: Code snippet for Encoding Algorithm

```

1 public function encoding($val){
2     $temp = $val;
3     $val = round($val * 100000);
4     if ($temp < 0) $binaryValue = decbin(~($val<<1));
5     else $binaryValue = decbin($val<<1);
6     $paddingZeros = strlen($binaryValue)%5 == 0 ? strlen($binaryValue):
7         (5 - strlen($binaryValue)%5) + strlen($binaryValue);
8     $binaryValue = str_pad($binaryValue, $paddingZeros, "0", STR_PAD_LEFT);
9     if (strlen($binaryValue)>=5) $chunks[0] = substr($binaryValue, -5, 5);
10    if (strlen($binaryValue)>=10) $chunks[1] = substr($binaryValue, -10, 5);
11    if (strlen($binaryValue)>=15) $chunks[2] = substr($binaryValue, -15, 5);
12    if (strlen($binaryValue)>=20) $chunks[3] = substr($binaryValue, -20, 5);
13    if (strlen($binaryValue)>=25) $chunks[4] = substr($binaryValue, -25, 5);
14    if (strlen($binaryValue)>=30) $chunks[5] = substr($binaryValue, -30, 5);

```



```

15  if (strlen($binaryValue)>=35) $chunks[6] = substr($binaryValue, -35, 5);
16  for($i=0;$i<count($chunks); $i++){
17  $chunks[$i] = $i==(count($chunks)-1) ? chr(bindec(str_pad($chunks[$i], 6,
18      "0", STR_PAD_LEFT)) + 63) : chr(bindec(str_pad($chunks[$i], 6, "0",
19      STR_PAD_LEFT) | "100000") + 63);
20  }
21  $chunks = implode($chunks);
22  return $chunks;
23  }
24
25  public function pair($latitude, $longitude){
26  if (round($latitude*100000) == 0 || $latitude == NULL){
27  return "";
28  }
29  return ($this->encoding($latitude) . $this->encoding($longitude));
30  }
31
32  public function checkTemp($tempValue){
33  if ($tempValue != ""){
34  $lvl_gValuepair[0] = "B";
35  $lvl_gValuepair[1] = $tempValue;
36  }
37  else{
38  $lvl_gValuepair[0] = ""; $lvl_gValuepair[1] = "";
39  }
40  return $lvl_gValuepair;
41  }
42
43  public function generateString(){
44  $mysql = mysql_connect(DATABASE_SERVER, DATABASE_USERNAME,
45      DATABASE_PASSWORD);
46  mysql_select_db("trafficanimation");
47  $query = "SELECT count( primaryKey) as numRows FROM

```

```

48     'current_locationsoct27 '";
49 $result = mysql_query($query);
50 $row = mysql_fetch_object($result);
51 $count = $row->numRows;
52 for($i=1; $i<=$count; $i++){
53 $query = "SELECT lat , longi , difflat , difflong , difflat1 , difflong1 ,
54     difflat2 , difflong2 , difflat3 , difflong3 , difflat4 , difflong4
55     FROM 'current_locationsoct27 ' WHERE primaryKey = " . $i;
56 $result = mysql_query($query);
57 if (!$result) return "Error selecting data";
58 $row = mysql_fetch_object($result);
59 $encLevel = "";
60 $generatedValue = "";
61 $answerPair = $this->checkTemp($this->pair($row->lat , $row->longi));
62 $encLevel = $encLevel . $answerPair[0];
63 $generatedValue = $generatedValue . $answerPair[1];
64 $answerPair = $this->checkTemp($this->pair($row->difflat , $row->difflong));
65 $encLevel = $encLevel . $answerPair[0];
66 $generatedValue = $generatedValue . $answerPair[1];
67 $answerPair = $this->checkTemp($this->pair($row->difflat1 , $row->difflong1));
68 $encLevel = $encLevel . $answerPair[0];
69 $generatedValue = $generatedValue . $answerPair[1];
70 $answerPair = $this->checkTemp($this->pair($row->difflat2 , $row->difflong2));
71 $encLevel = $encLevel . $answerPair[0];
72 $generatedValue = $generatedValue . $answerPair[1];
73 $answerPair = $this->checkTemp($this->pair($row->difflat3 , $row->difflong3));
74 $encLevel = $encLevel . $answerPair[0];
75 $generatedValue = $generatedValue . $answerPair[1];
76 $answerPair = $this->checkTemp($this->pair($row->difflat4 , $row->difflong4));
77 $encLevel = $encLevel . $answerPair[0];
78 $generatedValue = $generatedValue . $answerPair[1];
79 $query = "UPDATE 'current_locationsoct27 ' SET path = " . $generatedValue .
80     " , encLevel = " . $encLevel . " ' WHERE primaryKey = " . $i;

```

```

81 $result = mysql_query($query);
82 if (!$result) return "Error inserting path";
83 }
84 }

```

Figure 8.4: Google Maps API for Flash: Database for generating Encoded Polylines

		lat	longi	difflat	difflat1	difflong	difflong1	path	primaryKey	encLevel
		36.0680680000	-115.2048720000	-0.0000001300	0.0007371300	0.0000004000	0.0030936000	mpc{E} c~T5CIR	1	BB
		36.0680680000	-115.2048710000	-0.0000001900	0.0007371900	-0.0000004000	0.0030934000	mpc{E} c~T5CIR	2	BB
		36.0684180000	-115.1995010000	0.0003870000	-0.0003871900	-0.0022770000	0.0022770000	src{Ez b~TmAfMIAgM@mfAaK	3	BBBB
		36.0671410000	-115.1932470000	0.0009010000	-0.0009005700	-0.0026520000	0.0026520000	sjc{Exta~TsDpOrDqO~DwT	4	BBBB
		36.0651330000	-115.1866370000	0.0010530000	-0.0010534100	-0.0031250000	0.0031247000	a~b{Enk'~TqEpRpEoRZmL	5	BBBB
		36.0645940000	-115.1831060000	0.0007240000	-0.0007237500	-0.0025580000	0.0025582000	uzb{Elu~ToC~NnC_O	6	BBB
		36.0661650000	-115.1884100000	0.0006620000	-0.0006623900	-0.0024460000	0.0024464000	qdc{Epv'~TcChNbCiNhDeP	7	BBBB
		36.0675480000	-115.1928650000	0.0009970000	-0.0009967300	-0.0041500000	0.0041497000	emc{Elra~TgE XfE}XnCqK	8	BBBB
		36.0688450000	-115.1995150000	-0.0000750000	0.0000749600	-0.0026930000	0.0026927000	iuc{E~{b~TNxOMyOz@sN	9	BBBB
		36.0684760000	-115.2049100000	0.0000001200	0.0002938800	0.0000000000	0.0027020000	_sc{Et} c~Ty@{O	10	BB
		36.0684760000	-115.2049110000	0.0000001200	0.0002938800	0.0000003000	0.0027027000	_sc{Et} c~Ty@{O	11	BB
		36.0661650000	-115.1884090000	36.0000000000	NULL	-115.0000000000	NULL	qdc{Epv'~T_gvzE~{IT	12	BB
		36.0644760000	-115.1824440000	0.0005130000	-0.0005134800	-0.0020400000	0.0020399000	_zb{Efq~TeBvKdBwK	13	BBB
		36.0620860000	-115.1804650000	-0.0000003600	0.0027293600	0.0000001000	-0.0001351000	akb{Ejd~TaPZ	14	BB
		36.0666710000	-115.1804620000	-0.0018560000	0.0018558900	-0.0001380000	0.0001385000	ugc{Ezd~TrjZsjgXZ	15	BBBB
		36.0672230000	-115.1810580000	-0.0020610000	0.0020614600	0.0000940000	-0.0000938000	ckc{Erh~TzKQ(KPmVU	16	BBBB
		36.0620880000	-115.1810900000	-0.0000002500	0.0030742500	0.0000001000	0.0001259000	akb{Exh~TeRY	17	BB
		36.0733850000	-115.1806030000	-0.0026720000	0.0026720400	0.0000003000	-0.0000026000	uqd{Eve~TlO7uO~-----BeMI	18	BBBB
		36.0772720000	-115.1803910000	-0.0016160000	0.0016156000	-0.0001660000	0.0001661000	}ie{Eld~Tbi'@cia@gWd@	19	BBBB
		36.0842480000	-115.1804240000	-0.0030950000	0.0030952000	-0.0001540000	0.0001535000	quf{Erd~TJRkRlOP	20	BBBB
		36.0888170000	-115.1804190000	-0.0019930000	0.0019928100	-0.0000950000	0.0000946000	crg{Erd~TIKRmKQmRL	21	BBBB
		36.0869540000	-115.1809970000	-0.0016040000	0.0016036900	0.0000750000	-0.0000751000	mfg{Efh~T~HQ_Inkk@_@	22	BBBB
		36.0828770000	-115.1810080000	-0.0014810000	0.0014808700	0.0001080000	-0.0001080000	_mf{Ehh~TfHUgHTmNQ	23	BBBB
		36.0773010000	-115.1809060000	-0.0014890000	0.0014893100	-0.0000370000	0.0000374000	cje{Etg~ThHFHIGqXA	24	BBBB
		36.0729930000	-115.1810870000	-0.0020200000	0.0020197700	0.0001440000	-0.0001437000	eod{Exh~Ttk[sKzSPf	25	BBBB
		36.0942980000	-115.1803750000	-0.0023760000	0.0023764100	-0.0001170000	0.0001172000	kth{Ejd~TzMW{MW}IL	26	BBBB

## 8.2.2 Google Maps in Interactive Software

The previous section unveiled a plethora of tasks that can be fulfilled by using the Google MAPS. The daytime Safety Belt surveys project is associated with collecting the Safety Belt data at 64 locations throughout Nevada. Since the observation sites are all pre-decided, it made sense to geographically display them as customized markers on a Google MAP. This is shown in Figure 8.6.

In order to use the Google Maps API for Flash in our Flex application, the developer needs to download the SDK from the their website (Google, 2011) and sign up for a developer key which will be required will including the *maps:3D* component within the Flex application. As evident in Figure 8.6, a 3-D map has been used which allows

Figure 8.5: Google Maps API for Flash: Generated Traffic Animation with Encoded Polygons

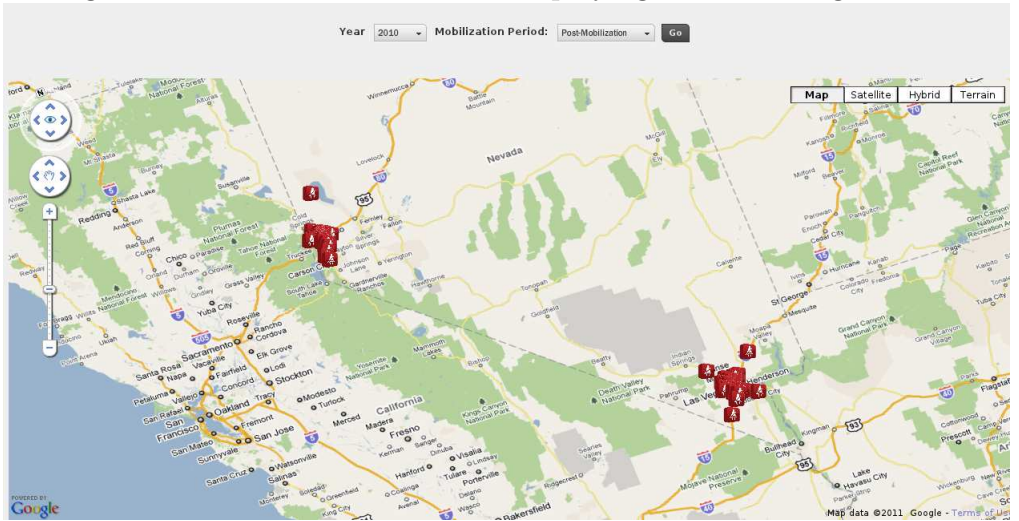


(a) Traffic Animation: Zoomed out View of a segment



(b) Traffic Animation: Zoomed in View of a segment

Figure 8.6: Interactive Software: Displaying sites on Google MAPS



the user to zoom, select a map type and navigate the map in 3-D. These controls are added before the map view is displayed to the user. Listing 8.8 shows how to add a 3-D map to a Flex application. `<maps:Map3D` component is included within the MXML file and the map is bound to `mapevent_mappreinitialize` and `mapevent_mapready` events, with `map_onMapPreInitialize()` and `map_onMapReady()` event listeners respectively. Both these event listeners are included within the `<mx:Script>` block and set the properties of the map before the map view is displayed to the user.

Listing 8.8: Interactive Software: Adding Google Map

```

1
2 <maps:Map3D width="100%" height="90%" id="map" key="your_key"
3     mapevent_mappreinitialize="{map_onMapPreInitialize();}"
4     mapevent_mapready="{map_onMapReady();}" />
5
6 <mx:Script>
7 <![CDATA[
8
9 private function map_onMapPreInitialize() : void {
10 var opt : MapOptions = new MapOptions();
11 opt.viewMode = View.VIEWMODE_ORTHOGONAL;
12 opt.attitude = new Attitude(20,30,0);
13 map.setInitOptions(opt);
14 }
15 private function map_onMapReady() : void {
16 map.setCenter(centerPos, 6);
17 map.addControl(new NavigationControl());
18 myRemote.getMarkerCoordinates();
19 }
20 ]]>
21 </mx:Script>

```

As mentioned in Section 8.2.1, custom icons can be used as markers on the Google Maps. Hence, a Safety Belt icon was embedded into the Flex application using the *Embed* metadata tag. In Flex terminology, the Safety Belt icon is called an *asset*. When an asset is embedded in the flex application, it is compiled into the application's SWF file and hence can be accessed faster than the approach when it has to be loaded from a remote location. However, one should be cautious about embedding too many assets because it can increase the loading time of the SWF file at run time. However, in our case, the icon will only be embedded once and will be used multiple times for setting the markers by typecasting the generic *Class* object as a *Bitmap* object. This is shown in Listing 8.8. In the previous Listing 8.8, a call to the remote object was made in the event listener *map\_onMapReady()* by calling the *getMarkerCoordinates()* function of the *remoteObject*, which in fact is the function written in the PHP file at the backend. The PHP file queries the MySQL database and returns the result back to the client application. Depending upon whether the results were retrieved or not, *ResultEvent* or *FaultEvent* events will be triggered which will be captured by event listeners *setMarkers(event:ResultEvent)* and *faultMarkers(event:FaultEvent)*. Now, assuming that the data is successfully received by the client, 64 Safety Belt icon based markers will be geographically attached to the latitude and longitude coordinates returned by the server. A few properties for every marker (e.g. *iconAlignment*, *tooltip* and *clickable*) can be set and event listeners can be added to them for the *click* and *doubleclick* events. Now, the stage is set for displaying Safety Belt based customized markers on the Google Map. Also, a click or double click event on any marker will be dispatched and will be captured by event

listeners *handleMarkerClick()* and *handleMarkerDoubleClick()*. In case a double click event is dispatched by the marker, the map will fly to the geographical coordinates of that marker with a higher zoom level. On the other hand, a *remoteobject* call to *getSiteFromMarker()* will be initiated if the marker is clicked once. This process is shown in Listing 8.9.

Listing 8.9: Interactive Software: Setting Markers on Google Map

```

1 // ACTIONSCRIPT code:
2
3 <mx:Script>
4 <![CDATA[
5     [@Embed(source="pictures/seatbelt1.png")]
6     private var class1:Class;
7     private function setMarkers(event:ResultEvent):void{
8         latLngArray = new Array();
9         var applicationWidth:Number = this.width;
10        var applicationHeight:Number = this.height;
11        for (var i:int=0; i<event.result.length; i++){
12            latLngArray[i] = "(" + event.result[i]["latitude"] + ", " +
13                event.result[i]["longitude"] + ")";
14            var bm:Bitmap = new class1() as Bitmap;
15            bm.width = 20;
16            bm.height = 20;
17            var markerOptions :MarkerOptions = new MarkerOptions({
18                hasShadow : false ,
19                iconAlignment: MarkerOptions.ALIGN_HORIZONTAL_CENTER |
20                    MarkerOptions.ALIGN_VERTICAL_CENTER
21            });
22            markerOptions.tooltip = "SiteName: " + event.result[i]["name"];
23            markerOptions.icon = bm;
24            markerOptions.clickable = true;

```

```

25     var marker : Marker = new Marker(new LatLng(parseFloat(event.result[i].latitude)
26         ,parseFloat(event.result[i].longitude)), markerOptions);
27     marker.addListener(MapMouseEvent.CLICK, handleMarkerClick);
28     marker.addListener(MapMouseEvent.DOUBLE_CLICK, handleMarkerDoubleClick);
29     this.map.addOverlay(marker);
30 } //end of the for loop
31 myMapTypeControl = new MapTypeControl();
32 myMapTypeControl.setControlPosition(topRight);
33 this.map.addControl(myMapTypeControl);
34 this.map.addControl(new NavigationControl());
35 myRemote.getTotalYearPeriod(databaseChosen);
36 } //end of function setMarkers
37
38 private function faultMarkers(event:FaultEvent):void{
39     Alert.show("Data not retrieved");
40 } //end of function faultMarkers
41
42 private function handleMarkerClick(event:MapMouseEvent):void{
43     if (databaseChanged == false){
44         Alert.show("Hit Go button first");
45         return;
46     }//end of if statement
47     ind = (latLngArray.indexOf(event.latLng.toString()) + 1).toString();
48     myRemote.getSiteFromMarker(ind, this.totalArray, databaseChosen);
49 }//end of function handleMarkerClick
50
51 private function handleMarkerDoubleClick(event:MapMouseEvent):void{
52     map.flyTo(new LatLng(event.latLng.lat(), event.latLng.lng()),16,
53         new Attitude(20,30,0),3);
54 }//end of function handleMarkerDoubleClick
55
56 ]]>
57 </mx:Script>

```





In Listing 8.9, it has been demonstrated how a `remoteObject` call will be initiated to *getSiteFromMarker()* function in the PHP file. The PHP function is shown in Listing 8.10.

Listing 8.10: Interactive Software: PHP function for getting data for a specific marker

```
1 public function getSiteFromMarker($id, $totalArray, $db) {
2     $mysql = mysql_connect(DATABASE_SERVER, DATABASE_USERNAME, DATABASE_PASSWORD);
3     mysql_select_db(DATABASE_SEATBELTSITES);
4     $query = "select * from sites2010 WHERE primaryId = '$id'";
5     $result = mysql_query($query);
6     $row = mysql_fetch_object($result);
7     $returnable["Name"] = $row->name;
8     $returnable["Area"] = $row->area;
9     $returnable["Volume"] = $row->volume;
10    $returnable["County"] = $row->county;
11    $perSite = $this->getBeltedPerSiteYearPeriod($id, $db);
12    $returnable["BeltedDCont"] = number_format((( $totalArray["beltedDrivers"] -
13    $perSite["beltedDrivers"] ) / $totalArray["totalDrivers"] ) * 100, 2, '.', '');
14    $returnable["BeltedPCont"] = number_format((( $totalArray["beltedPassengers"] -
15    $perSite["beltedPassengers"] ) / $totalArray["totalPassengers"] ) * 100, 2, '.', '');
16    $returnable["BeltedDPerc"] = number_format(( $totalArray["beltedDrivers"]
17    / $totalArray["totalDrivers"] ) * 100, 2, '.', '');
18    $returnable["BeltedPPerc"] = number_format(( $totalArray["beltedPassengers"]
19    / $totalArray["totalPassengers"] ) * 100, 2, '.', '');
20    $returnable["Id"] = $row->primaryId;
21    return $returnable;
22 } //end of function getSiteFromMarker
```

The PHP function from Listing 8.10 returns the result back to the client application. If the site-specific data is successfully retrieved, a call to the actionscript function *markerInfoRetrieved()* is made which initializes a datagrid, sets a new state in which a panel component rolls over the map in an animated manner. The *markerinfo retrieved* function is shown in Listing 8.11.

Listing 8.11: Interactive Software: Data retrieved for a specific marker

```

1 private function markerInfoRetrieved(event:ResultEvent):void{
2     panelTitle = new String();
3     arrayDatagrid = new Array();
4     arrayDatagrid = [
5         {Title: "Site no.", Score: event.result["Id"]},
6         {Title: "County", Score: event.result["County"]},
7         {Title: "Area", Score: event.result["Area"]},
8         {Title: "Volume", Score: event.result["Volume"]}
9     ];
10    arrayDatagridContribution = new Array();
11    arrayDatagridContribution = [
12        {Title: "Statewide BD %", Score: event.result["BeltedDPerc"]},
13        {Title: "Statewide BD % W/O", Score: event.result["BeltedDCont"]},
14        {Title: "Statewide BP %", Score: event.result["BeltedPPerc"]},
15        {Title: "Statewide BP % W/O", Score: event.result["BeltedPCont"]},
16    ];
17    if (databasePeriod == "precampaign")
18        panelTitle = event.result["Name"] + " - Year: " + databaseYear + ", Period: "
19        + "Pre-Mobilization";
20    else if (databasePeriod == "postcampaign")
21        panelTitle = event.result["Name"] + " - Year: " + databaseYear + ", Period: "
22        + "Post-Mobilization";
23    else
24        panelTitle = event.result["Name"] + " - Year: " + databaseYear + ", Period: "

```

```

25         + databasePeriod;
26     rollOverCanvasX = (this.width-rollOverCanvas.width)/2;
27     rollOverCanvasY = (this.height-rollOverCanvas.height)/2;
28     currentState = 'stateRollOverCanvas';
29     myRemote.getChartDataPerSite(ind, "1");
30     anim.play();
31 }//end of function markerInfoRetrieved

```

As mentioned above in code Listing 8.11, a panel rolls over the main screen, while the map at the back becomes a little dim. This is achieved through an animation demonstrated in Listing 8.12. The *AnimateProperty* effect used in Listing 8.12 animates a property or style of a component, which in our case is the panel component. The *AnimateProperty* effect sets the specified property of the target component from a given start value to the end value. The panel becomes visible only when the marker is clicked by setting the *alpha* value from 0 to 1, while simultaneously, the *alpha* value of the map is set from 1 to 0.3. Additionally, the *AnimateProperty* effect used on the panel component causes it to rotate from  $-45^\circ$  to  $0^\circ$ , while setting its position from the top left corner with  $x = 0$  and  $y = 0$  to the center of the screen. All these effects act in parallel since they have been combined within the `<mx:Parallel>` block. It should be noted that, if the `<mx:Sequence>` block had been used, the effects within the block would have occurred sequentially.

Listing 8.12: Interactive Software: Animation for making the Panel visible

```

1 <mx:Parallel id="anim" repeatCount="1">
2   <mx:AnimateProperty target="{mapCanvas}" duration="2000" property="alpha"

```

```

3      fromValue="1" toValue="0.3"/>
4  <mx:AnimateProperty target="{rollOverCanvas}" duration="2000" property="alpha"
5      fromValue="0.0" toValue="1.0"/>
6  <mx:AnimateProperty target="{rollOverCanvas}" duration="2000" property="rotation"
7      fromValue="-45" toValue="0"/>
8  <mx:AnimateProperty target="{rollOverCanvas}" duration="2000" property="x"
9      fromValue="0" toValue="{rollOverCanvasX}" easingFunction="Bounce.easeIn"/>
10 <mx:AnimateProperty target="{rollOverCanvas}" duration="2000" property="scaleX"
11     fromValue="1.5" toValue="1"/>
12 <mx:AnimateProperty target="{rollOverCanvas}" duration="2000" property="scaleY"
13     fromValue="1.5" toValue="1"/>
14 </mx:Parallel>

```

Figure 8.7 shows the panel container that rolls to the middle of the screen when the user clicks any marker on the Google Map. The MXML structure of this panel is very simple and will not be discussed. Basically, it contains a few components available in Flex: Checkbox, Datagrid, Button, Image, ColumnChart and Legend. It is important to note that before the panel is loaded by changing the *currentState* property from the *base* state to *stateRollOverCanvas* state as shown in Listing 8.11, a remoteobject call is made to the server to return the data corresponding to *the % belted drivers* at chosen site for the selected time period and year. The data returned from this remote object call will be bound to the *ColumnChart* component within the panel and the legend is updated accordingly.

Two separate PHP functions have been created for a remote object call at the backend. One PHP function named *getChartDataPerSite()* is called through a remote object only when the user is interested in viewing the x-axis of the Columnchart

Figure 8.7: Interactive Software: Panel obtained by a single click on the marker



as years subdivided over Pre-Mobilization and Post-Mobilization Safety Belt usage. However, if the user selects any of the radio buttons (followed by the corresponding year) in front of the label named *Optional x axis Categories* in the Figure 8.8, the X axis of the *ColumnChart* will change depending upon the selected radio button. In this case, the second PHP function named *getChartDataPerSiteOptional()* will be called by the remoteobject. This is shown in Figure 8.8.

Figure 8.8: Interactive Software: Columnchart x axis showing year, age, gender and ethnicity distribution



(a) Calling *getChartDataPerSite()*: Yearwise distribution (b) Calling *getChartDataPerSiteOptional()*: Genderwise distribution



(c) Calling *getChartDataPerSiteOptional()*: Age-wise distribution (d) Calling *getChartDataPerSiteOptional()*: Ethnicitywise distribution

The PHP functions *getChartDataPerSite()* and *getChartDataPerSiteOptional()* are shown in Listing 8.13 and 8.14 respectively. From Listing 8.13, it can be observed that *getChartDataPerSite()* makes a call to another function *chartOptions()* which takes two input parameters: (i) Id number of the site (ii) A variable named *option* which decides whether the data is being requested for the driver (*option* = 1), passenger (*option* = 2) or both the front seat occupants (*option* = 3). Since the remoteobject call to *getChartDataPerSite()* is made whenever the % of belted people is to be compared against yearwise distribution for all the years, therefore, *chartOptions()* makes a query to the MySQL databases for every year that the data is available on the server. The resultant array sent back from the *chartOptions()* function contains the total number of people at that site (drivers, passengers or total front seat occupants, depending upon the variable *option*), along with the number indicating how many of them were belted. Since the data was available for both the Pre-Mobilization and Post-Mobilization time periods for 2007, 2008, 2009 and 2010, the resultant array sent back from the *chartOptions()* function will always have a size of 8 objects. This array is put into the variable *temp* inside *getChartDataPerSite()* function and then % of belted people is calculated using the division operation for every year, and later formatted to round up the % of belted people to 2 decimal places using the *number\_format()* function. Finally, the array of percentages will be returned back to the client Flex application, which will be displayed in the Column Chart.

Listing 8.13: Interactive Software: PHP functions - *getChartDataPerSite()* and *chartOptions()*

```

1 <?php
2 define("DATABASE_2007_PRE", "precampaign2007");
3 define("DATABASE_2007_POST", "postcampaign2007");
4 define("DATABASE_2008_PRE", "precampaign2008");
5 define("DATABASE_2008_POST", "postcampaign2008");
6 define("DATABASE_2009_PRE", "precampaign2009");
7 define("DATABASE_2009_POST", "postcampaign2009");
8 define("DATABASE_2010_PRE", "precampaign");
9 define("DATABASE_2010_POST", "postcampaign");
10 class phpFunctions{
11 public function getChartDataPerSite($id, $option) {
12 if ($option=="1"){
13     $temp = $this->chartOptions($id, "1");
14 }
15 elseif ($option=="2"){
16     $temp = $this->chartOptions($id, "2");
17 }
18 elseif($option == "3"){
19     $temp1 = $this->chartOptions($id, "1");
20     $temp2 = $this->chartOptions($id, "2");
21     for($i=0; $i<8; $i++){
22         $temp[$i]["belted"] = $temp1[$i]["belted"] + $temp2[$i]["belted"];
23         $temp[$i]["total"] = $temp1[$i]["total"] + $temp2[$i]["total"];
24     }
25 }
26 $arr[0]["Year"] = "2007";
27 $arr[0]["PreCampaign"] = ($temp[0]["total"] ==0) ? -1:
28     number_format(($temp[0]["belted"] / $temp[0]["total"])*100, 2, '.', '');
29 $arr[0]["PostCampaign"] = ($temp[1]["total"] ==0) ? -1:
30     number_format(($temp[1]["belted"] / $temp[1]["total"])*100, 2, '.', '');

```



```

31 $arr[1]["Year"] = "2008";
32 $arr[1]["PreCampaign"] = ($temp[2]["total"] ==0) ? -1:
33     number_format(($temp[2]["belted"] / $temp[2]["total"])*100, 2 , '.', '');
34 $arr[1]["PostCampaign"] = ($temp[3]["total"] ==0) ? -1:
35     number_format(($temp[3]["belted"] / $temp[3]["total"])*100, 2 , '.', '');
36 $arr[2]["Year"] = "2009";
37 $arr[2]["PreCampaign"] = ($temp[4]["total"] ==0)? -1:
38     number_format(($temp[4]["belted"] / $temp[4]["total"])*100, 2 , '.', '');
39 $arr[2]["PostCampaign"] = ($temp[5]["total"] ==0)? -1:
40     number_format(($temp[5]["belted"] / $temp[5]["total"])*100, 2 , '.', '');
41 $arr[3]["Year"] = "2010";
42 $arr[3]["PreCampaign"] = ($temp[6]["total"] ==0)? -1:
43     number_format(($temp[6]["belted"] / $temp[6]["total"])*100, 2 , '.', '');
44 $arr[3]["PostCampaign"] = ($temp[7]["total"] ==0)? -1:
45     number_format(($temp[7]["belted"] / $temp[7]["total"])*100, 2 , '.', '');
46 return $arr;
47 }
48
49 public function chartOptions($id, $option){
50 $mysql = mysql_connect(DATABASE_SERVER, DATABASE_USERNAME, DATABASE_PASSWORD);
51 if ($option == "1"){
52     $check = "driversb";
53     $forpassenger = " ";
54 }
55
56 elseif($option == "2") {
57     $check = "passengersb";
58     $forpassenger = " WHERE passengersb IS NOT NULL AND passengersb !=''";
59 }
60 $returnable = array();
61 $databaseArray = array(DATABASE_2007_PRE, DATABASE_2007_POST, DATABASE_2008_PRE,
62     DATABASE_2008_POST, DATABASE_2009_PRE, DATABASE_2009_POST, DATABASE_2010_PRE,
63     DATABASE_2010_POST);

```

```

64  for ($i=0; $i<count($databaseArray);$i++){
65      mysql_select_db($databaseArray[$i]);
66      $query ="select entry_id from site" . $id. $forpassenger;
67      $result = mysql_query($query);
68      if (!$result){
69          $returnable[$i]["belted"] = 0;
70          $returnable[$i]["total"] = 0;
71      }
72      else{
73          $returnable[$i]["total"] = mysql_num_rows($result);
74          mysql_free_result($result);
75          $query ="select entry_id from site" . $id. " WHERE " . $check. " = 'Yes'";
76          $result = mysql_query($query);
77          $returnable[$i]["belted"] = mysql_num_rows($result);
78          mysql_free_result($result);
79      }
80  }
81  return $returnable;
82  }
83  }
84  ?>

```

As mentioned above, Listing 8.14 shows the code for two PHP functions: *getChartDataPerSiteOptional()* and *chartOptionsOptional()*. The remoteobject calls *getChartDataPerSiteOptional()* function whenever the user wants to see the distribution of % belted people on a site when compared against gender, age or ethnicity for a particular year. Hence, this function takes as input four parameters: (i) Site Id, (ii) Category (age, gender or ethnicity), (iii) Year (2007, 2008, 2009 or 2010) and (iv) A variable named *driverpassenger* which decides whether the data is being requested by the re-

moteobject for driver (*driverorpassenger* = 1), passenger (*driverorpassenger* = 2) or both front seat occupants (*driverorpassenger* = 3. As evident in Listing 8.14, it can be observed that a function call is made to *chartOptionsOptional()*. This function takes three arguments: (i) Site Id, (ii) Driver or passenger Option and (iii) Category (Age, Gender or Ethnicity). *chartOptionsOptional()* creates a set of MySQL queries to find out the number of total people and the number people belted out of them, within a particular category. For example, if the category chosen is Age, then, depending upon if the data is being requested for drivers, passengers or both front seat occupants, 8 queries each will be generated for the Pre-Mobilization and Post-Mobilization periods: 4 queries to find out the total number of people within age groups <15, 15-19, 20-60 and >60, and then 4 queries to find out the belted number of people within all these four age groups. Hence, an array will be returned from *chartOptionsOptional()*, whose size will vary depending upon the category chosen. This array will be used to find the % of the belted people for a specific year within the requested category and the final array of belted percentages will be sent back to the client Flex application to be displayed through a Column Chart component.

Listing 8.14: Interactive Software: PHP functions - *getChartDataPerSiteOptional()* and *chartOptionsOptional()*

```

1 <?php
2 class phpFunctions{
3     public function getChartDataPerSiteOptional($id , $currentCategory , $currentYear ,
4         $driverorpassenger) {
5         if ($driverorpassenger=="1"){
6             $temp = $this->chartOptionsOptional($id , " driver" , $currentCategory ,

```

```

7      $currentYear);
8  }
9  elseif ($driverorpassenger=="2"){
10     $temp = $this->chartOptionsOptional($id, "passenger", $currentCategory,
11         $currentYear);
12 }
13 elseif($driverorpassenger == "3"){
14     $temp1 = $this->chartOptionsOptional($id, "driver", $currentCategory,
15         $currentYear);
16     $temp2 = $this->chartOptionsOptional($id, "passenger", $currentCategory,
17         $currentYear);
18     for($i=0; $i<count($temp2); $i++){
19         $temp[$i] = $temp1[$i]+ $temp2[$i];
20     }
21 }
22 $arr[0] = ($temp[0]==0) ? -1: number_format($temp[1]/$temp[0]*100, 2, '.', '');
23 $arr[1] = ($temp[2]==0) ? -1 : number_format($temp[3]/$temp[2]*100, 2, '.', '');
24 $arr[2] = ($temp[4]==0) ? -1: number_format($temp[5]/$temp[4]*100, 2, '.', '');
25 $arr[3] = ($temp[6]==0) ? -1: number_format($temp[7]/$temp[6]*100, 2, '.', '');
26 if ($currentCategory == "Age" || $currentCategory=="Ethnicity"){
27     $arr[4] = ($temp[8]==0) ? -1: number_format($temp[9]/$temp[8]*100, 2,
28         '.', '');
29     $arr[5] = ($temp[10]==0) ? -1: number_format($temp[11]/$temp[10]*100, 2,
30         '.', '');
31     $arr[6] = ($temp[12]==0) ? -1: number_format($temp[13]/$temp[12]*100, 2,
32         '.', '');
33     $arr[7] = ($temp[14]==0) ? -1: number_format($temp[15]/$temp[14]*100, 2,
34         '.', '');
35 }
36 return $arr;
37 }
38
39 public function chartOptionsOptional($id, $driverpassenger, $currentCategory,

```

```

40     $currentYear){
41     $mysql = mysql_connect(DATABASE_SERVER, DATABASE_USERNAME, DATABASE_PASSWORD);
42     if ($currentYear!="2010"){
43         $databasepre = "precampaign" . $currentYear;
44         $databasepost = "postcampaign" . $currentYear;
45     }
46     else{
47         $databasepre = "precampaign";
48         $databasepost = "postcampaign";
49     }
50     if ($currentCategory == "Gender"){
51         $queryArray = array("select entry_id from site" . $id. " WHERE " .
52             $driverpassenger ."sex = 'Male'", "select entry_id from site" . $id.
53             " WHERE " . $driverpassenger ."sex = 'Male' AND ". $driverpassenger .
54             "sb = 'Yes'", "select entry_id from site" . $id. " WHERE " .
55             $driverpassenger ."sex = 'Female'", "select entry_id from site" . $id.
56             " WHERE " . $driverpassenger ."sex = 'Female' AND ". $driverpassenger .
57             "sb = 'Yes'");
58     }
59     elseif ($currentCategory == "Age"){
60         if ($driverpassenger!="driver"){
61             $array1 = array("select entry_id from site" . $id. " WHERE " .
62                 $driverpassenger ."age = '<15'", "select entry_id from site" . $id.
63                 " WHERE " . $driverpassenger ."age = '<15' AND ". $driverpassenger .
64                 "sb = 'Yes'");
65         }
66         else{
67             $array1 = array("", "");
68         }
69         $array2 = array("select entry_id from site" . $id. " WHERE " .
70             $driverpassenger ."age = '15-19'", "select entry_id from site" . $id.
71             " WHERE " . $driverpassenger ."age = '15-19' AND ". $driverpassenger
72             ."sb = 'Yes'", "select entry_id from site" . $id. " WHERE " . $driverpassenger

```

```

73     ."age = '20-60'", "select entry_id from site" . $id. " WHERE " .
74     $driverpassenger ."age = '20-60' AND ". $driverpassenger .
75     "sb = 'Yes'", "select entry_id from site" . $id. " WHERE " . $driverpassenger
76     ."age = '>60'", "select entry_id from site" . $id. " WHERE " . $driverpassenger
77     ."age = '>60' AND ". $driverpassenger . "sb = 'Yes'");
78     $queryArray = array_merge($array1, $array2);
79 }
80 elseif ($currentCategory == "Ethnicity"){
81     $queryArray = array("select entry_id from site" . $id. " WHERE " .
82     $driverpassenger ."race = 'Caucasian'", "select entry_id from site" . $id.
83     " WHERE " . $driverpassenger ."race = 'Caucasian' AND ". $driverpassenger
84     ."sb = 'Yes'", "select entry_id from site" . $id. " WHERE " .
85     $driverpassenger ."race = 'Hispanic'", "select entry_id from site" . $id.
86     " WHERE " . $driverpassenger . "race = 'Hispanic' AND ". $driverpassenger
87     ."sb = 'Yes'", "select entry_id from site" . $id. " WHERE " . $driverpassenger
88     ."race = 'Black'", "select entry_id from site" . $id. " WHERE " .
89     $driverpassenger ."race = 'Black' AND ". $driverpassenger .
90     "sb = 'Yes'", "select entry_id from site" . $id. " WHERE " . $driverpassenger
91     ."race = 'Others'", "select entry_id from site" . $id. " WHERE " .
92     $driverpassenger ."race = 'Others' AND ". $driverpassenger . "sb = 'Yes'");
93 }
94 $databaseArray = array($databasepre, $databasepost);
95 $totalCount = 0;
96 for($d=0; $d<count($databaseArray);$d++){
97     mysql_select_db($databaseArray[$d]);
98     for($i=0; $i<count($queryArray);$i++){
99         $returnable[$totalCount] = ($queryArray[$i]=="")? 0:
100         mysql_num_rows(mysql_query($queryArray[$i]));
101         $totalCount++;
102     }
103 }
104 return $returnable;
105 }

```

```
106 }  
107 ?>
```

Listing 8.15 shows how to display the arrays returned from the remoteobject call to *getChartDataPerSite()* and *getChartDataPerSiteOptional()* functions on the Column Chart component. *eachSiteResults()* will be called when the data is retrieved successfully from the remoteobject call to *getChartDataPerSite()* function. Similarly, the actionsript function *eachSiteResultsOptional()* will be called when the server sends back the data response from the *getChartDataPerSiteOptional()* function. The MXML code of the Column Chart component is very easy to construct and follow. Since the comparison at every site is being done for the Pre-Mobilization and Post-Mobilization periods, no matter what the X-axis variables of the Column Chart component are, therefore the data corresponding to both these periods can be fixed as *ColumnSeries* component. However, if we had a different scenario where there was a need to change the *ColumnSeries* component dynamically, an Actionscript approach was available which will be discussed later in this chapter. The *Bindable* ArrayCollection *chartArray* causes the Column Chart to display the *yField* variables (fixed as *Pre* and *Post* variables), which are updated every time a remoteobject call is made to the server. Last but not the least, when the data is retrieved back from the server, *hideDataEffect* and *ShowDataEffect* events are fired sequentially which cause the data to be hidden and then become visible with an animation.

### Listing 8.15: Interactive Software: Displaying the RemoteObject data on Column Chart Component

```
1 <mx:Script>
2 <![CDATA[
3     private function eachSiteResults(event:ResultEvent):void{
4         chartArray = new ArrayCollection( [
5             { Year: "2007", Pre: event.result[0]["PreCampaign"],
6               Post: event.result[0]["PostCampaign"]},
7             { Year: "2008", Pre: event.result[1]["PreCampaign"],
8               Post: event.result[1]["PostCampaign"]},
9             { Year: "2009", Pre: event.result[2]["PreCampaign"],
10              Post: event.result[2]["PostCampaign"]},
11             { Year: "2010", Pre: event.result[3]["PreCampaign"],
12              Post: event.result[3]["PostCampaign"]}
13         ]);
14     }
15
16     private function eachSiteResultsOptional(event:ResultEvent):void{
17         if (currentCategory=="Gender"){
18             chartArray = new ArrayCollection( [
19                 { Year: "Male", Pre: event.result[0], Post: event.result[2]},
20                 { Year: "Female", Pre: event.result[1], Post: event.result[3]},
21             ]);
22         }
23         else if (currentCategory=="Age"){
24             chartArray = new ArrayCollection( [
25                 { Year: "<15", Pre: event.result[0], Post: event.result[4]},
26                 { Year: "15-19", Pre: event.result[1], Post: event.result[5]},
27                 { Year: "20-60", Pre: event.result[2], Post: event.result[6]},
28                 { Year: ">60", Pre: event.result[3], Post: event.result[7]}
29             ]);
30         }
```



```

31 else if (currentCategory=="Ethnicity"){
32     chartArray = new ArrayCollection( [
33         { Year: "Caucasian", Pre: event.result[0], Post: event.result[4]},
34         { Year: "Hispanic", Pre: event.result[1], Post: event.result[5]},
35         { Year: "African-American", Pre: event.result[2], Post: event.result[6]},
36         { Year: "Others", Pre: event.result[3], Post: event.result[7]}
37     ]);
38 }
39 }
40
41 ]]>
42 </mx:Script>
43
44 <mx:SeriesSlide id="slideIn" duration="1000" direction="up"/>
45 <mx:SeriesSlide id="slideOut" duration="1000" direction="down"/>
46 <mx:ColumnChart width="100%" height="60%" id="columnChart1" showDataTips="true"
47     dataProvider="{chartArray}">
48     <mx:horizontalAxis>
49         <mx:CategoryAxis categoryField="Year"/>
50     </mx:horizontalAxis>
51     <mx:series>
52         <mx:ColumnSeries showDataEffect="{slideIn}" yField="Pre"
53             displayName="Pre-Mobilization Seatbelt Usage"/>
54         <mx:ColumnSeries hideDataEffect="{slideOut}" yField="Post"
55             displayName="Post-Mobilization Seatbelt Usage"/>
56     </mx:series>
57 </mx:ColumnChart>
58 <mx:Legend dataProvider="{columnChart1}" width="100%" height="5%" />

```

### 8.3 Custom Item Renderers

The `ListBase` class in Flex contains the following List components: `DataGrid`, `HorizontalList`, `List`, `Menu`, `MenuBar`, `TileList` and `Tree`. All these list components get their data from a data provider and display it through a default view. However, Flex allows the developers to override the default views and make the applications appealing to the general public. Custom Item renderers provide us with several advantages:

1. A fascinating user interface can be developed by replacing the default text display in a list item with a more compelling alternative.
2. We can combine multiple elements (for example: label and image) in a single list item.
3. The display of the data can be programmatically controlled within the custom item renderer and transferred to the main application through custom events.

#### 8.3.1 Creating Custom Item Renderers/Item Editors

Overall, there are three ways to create custom item renderers/item editors, the details of which are provided on their Help Resource website (Adobe, 2011e). To give a brief overview of the process, the custom item renderers can be developed in the following ways:

##### 1. **Drop-in Item Renderers/Item Editors**

A drop-in item renderer or item editor is a Flex component that is specified as the value of the *itemRenderer* or *itemEditor* property of a list control. The following code snippet 8.16 demonstrates the way how a *NumericStepper* can be made the default view of a *DataGrid* column.

### Listing 8.16: Drop-in item renderer

```
1 <mx:DataGrid id="myDG" dataProvider="{myDP}"
2     variableRowHeight="true" editable="true" >
3 <mx:columns>
4 <mx:DataGridColumn dataField="label1"
5     headerText="Order #" />
6 <mx:DataGridColumn dataField="quant" headerText="Qty"
7     itemEditor="mx.controls.NumericStepper"
8     editorDataField="value" />
9 </mx:columns >
10 </mx:DataGrid>
```

## 2. Inline Item Renderers/Item Editors

A major disadvantage with Drop-in item renderers is that they can only be specified as the values of a list control property, but cannot be configured at all. To overcome this difficulty, inline item renderers were introduced which give the developer freedom to configure properties of the control being used as the custom item renderer or item editor. The following code snippet 8.17 demonstrates the way how properties can be set for the *NumericStepper* as the default view of a *DataGrid* column.

### Listing 8.17: Inline Item renderer

```
1 <mx:DataGrid id="myDG" dataProvider="{myDP}"
2     variableRowHeight="true" editable="true" >
3 <mx:columns>
4 <mx:DataGridColumn dataField="label1"
5     headerText="Order #" />
6 <mx:DataGridColumn dataField="quant"
7     editorDataField="value" headerText="Qty">
8 <mx:itemEditor>
9 <mx:Component>
```

```

10 <mx:NumericStepper stepSize="1" maximum="50" />
11 </mx:Component>
12 </mx:itemEditor>
13 </mx:DataGridColumn>
14 </mx:columns>
15 </mx:DataGrid>

```

### 3. Components as Item Renderers/Item Editors

As the name suggests, custom components created in Flex can be used as Item renderers again and again, thus reducing the time and effort required in writing the Inline or Drop on Item Renderers. The following code snippet 8.18 demonstrates the way how a component is defined, which is later used inside the *itemEditor* property as the default view of a *DataGrid* column.

#### Listing 8.18: Components as Item Renderers or Item Editors

```

1 //NSEditor.mxml inside myComponents folder:
2 <mx:NumericStepper stepSize="1" maximum="50"
3     xmlns:mx="http://www.adobe.com/2006/mxml" />
4
5 //MainApplication.mxml inside src folder:
6
7 <mx:DataGrid id="myDG" dataProvider="{myDP}"
8     variableRowHeight="true" editable="true">
9 <mx:columns>
10 <mx:DataGridColumn dataField="label1"
11     headerText="Order #" />
12 <mx:DataGridColumn dataField="quant"
13     itemEditor="myComponents.NSEditor"
14     editorDataField="value" />
15 </mx:columns>
16 </mx:DataGrid>

```

The image obtained from all the three code snippets is shown in Figure 8.9.

Figure 8.9: Custom Item Renderers/Item Editors

Order #	Qty
Order #2314	7
Order #2315	4

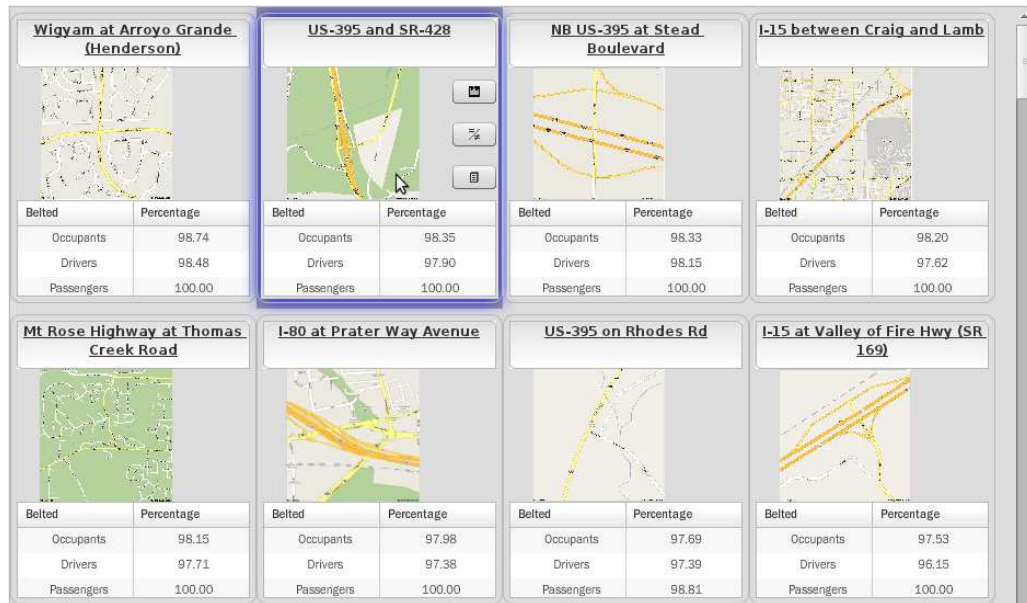
### 8.3.2 Custom Item Renderers in Interactive Safety Belt Software

In the above section, three different strategies to create custom item renderers/item editors were illustrated. In the Interactive Safety Belt Software, the format provided in Listing 8.18 was followed to create custom components which were used as custom item renderers. As mentioned before, these components gave us flexibility to set the properties and combine different controls, create and handle complex events etc. In this section, the item renderers developed for one of the components, *TileList*, is exhibited with a small code snippet.

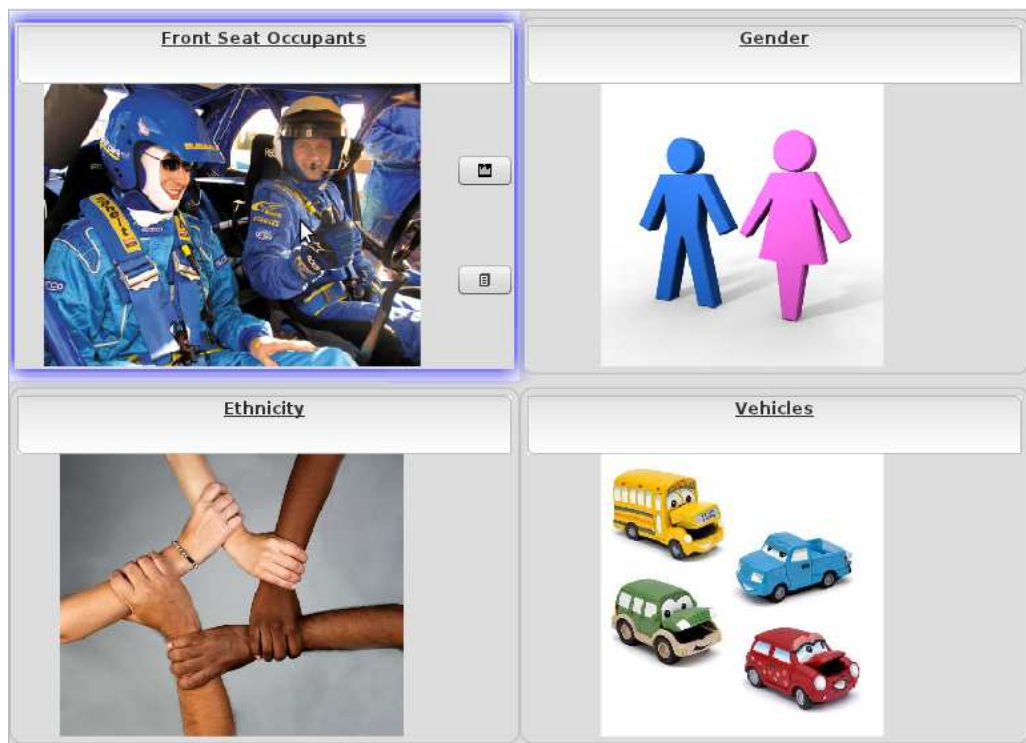
Figure 8.10 displays the custom item renderers that were developed for two different modules of the interactive software: *Site-Specific Analysis* and *Statewide Analysis*

As can be seen in both the figures, the default text view of tilelist has been altered to create a much more enticing view. In this section, the structure and the different events and effects used in the item renderer components *TileListItem.mxml*

Figure 8.10: TileList Custom Item Renderers



(a) Custom Item Renderer: Site Specific Analysis



(b) Custom Item Renderer: Statewide Analysis

and *statewideTileList.mxml* have been described. The full source code for both these components can be found online in the *components* folder of the software website (Lakhanpal, 2010).

1. Structure of the item renderer components:

The MXML code for both the item renderers is similar to each other, with a minor change depending upon what is required. The written MXML code for the tilelist item renderers in both the *Site-Specific Analysis* and *Statewide Analysis* modules had a few common components: image, buttons and textarea. Additionally, the *Site-Specific Analysis* module also contained a datagrid to display the belted information about the front seat occupants. Since the MXML code for the *Statewide Analysis* module is a proper subset of the *Site-Specific Analysis* module, hence, the MXML code for the item renderer in *Site-Specific Analysis* module is given in Listing 8.19.

Listing 8.19: Structure of TileList Item Renderer

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <mx:VBox rolloverEffect="{glowImage}" dataChange="updateRenderer()"
3     horizontalScrollPolicy="off" verticalScrollPolicy="off"
4     mouseOver="mouseOverHandler()" mouseOut="mouseOutHandler()"
5     xmlns:mx="http://www.adobe.com/2006/mxml" height="265" width="225"
6     borderStyle="solid" cornerRadius="10">
7 <mx:TextArea textAlign="center" width="100%" height="5%" id="idName"
8     editable="false" fontWeight="bold" fontStyle="normal"
9     textDecoration="underline" borderStyle="none" cornerRadius="10"
10    fontSize="12" fontFamily="Verdana" />
11 <mx:HBox width="100%" height="55%">
```

```

12 <mx:Image horizontalAlign="center" horizontalCenter="0" width="100%"
13     height="100%" id="idImage" />
14 <mx:VBox width="0.1%" height="100%">
15 <mx:Spacer height="100%" />
16 <mx:Button visible="false" width="100%" click="buttonHandler(event, 1)"
17     icon="@Embed(source='pictures/flex_graph.png')">
18     tooltip="Graphical Data" id="buttonDetails" />
19 <mx:Spacer height="100%" />
20 <mx:Button visible="false" width="100%" click="buttonHandler(event, 2)"
21     tooltip="Add to Compare List"
22     icon="@Embed(source='pictures/icon_compare.png')"> id="buttonCompare" />
23 <mx:Spacer height="100%" />
24 <mx:Button visible="false" click="buttonHandler(event, 3)"
25     icon="@Embed(source='pictures/icon_details.png')"> width="100%"
26     tooltip="Tabular Data" id="buttonTabular" />
27 <mx:Spacer height="100%" />
28 </mx:VBox>
29 </mx:HBox>
30 <mx:DataGrid id="tileDataGrid" height="40%" width="100%" rowCount="3"
31     verticalScrollPolicy="off" horizontalCenter="0"
32     dataProvider="{ beltedDatagrid }">
33 <mx:columns>
34 <mx:DataGridColumn headerText="Belted" dataField="Title" />
35 <mx:DataGridColumn headerText="Percentage" dataField="Score" />
36 </mx:columns>
37 </mx:DataGrid>
38 </mx:VBox>

```

## 2. Displaying and updating data in the item renderer:

As shown in Listing 8.19, the *datachange* property of the item renderer component can be used to update the data. Listing 8.20 shows the function *updateRenderer()* responsible for updating the data.



### Listing 8.20: Updating data in Item Renderers

```
1 private function updateRenderer():void{
2     idName.text = data.Name;
3     idImage.source = "http://nutc.unlv.edu/PuneetLakhanpal/pictures/image" +
4         data.Id + ".png";
5     beltedDatagrid = new ArrayCollection();
6     var arrayDatagrid:Array = new Array();
7     arrayDatagrid = [{ Title: "Occupants", Score: data.BeltedOPerc },
8         { Title: "Drivers", Score: data.BeltedDPerc },
9         { Title: "Passengers", Score: data.BeltedPPerc }];
10    beltedDatagrid.source = arrayDatagrid;
11 }
```

### 3. Events in Item Renderers:

A major advantage of the components being used as Item renderers is that all the effects within the Flex SDK are available to these item renderers. In the next section, we discuss how to launch a custom event from within this *TileList* component.

## 8.4 Creating Custom Events

In Section 8.3.2, we created components which we called custom item renderers. From within these components, it is possible to launch a customized event to let the application know that something has happened from within the custom component. In this section, we provide the details of how to create a custom actionscript event and how to launch it within the custom component.

### 8.4.1 Creating Actionscript Event class

Consider the custom item renderer titled *TileListItem.mxml* for *Site-Specific Analysis*. This custom component is based on *VBox* component on which, if a mouse is rolled over, three buttons appear. When we click on one of these buttons, for example, 'Graphical data', we need to tell the parent component *dataAnalysis.mxml* (Site-Specific Analysis module) that it needs to display the data for a particular site in a graphical format. Similarly, if 'Tabular Data' button is clicked in this item renderer, it should be able to tell the parent component that it now needs to display the data for the particular site in a tabular format. Therefore, the easiest thing is to send an integer (or similar datatype) back to the parent component which it can identify and respond accordingly. In Listing 8.21 shows how to create a custom event class in actionscript.

Listing 8.21: Creating Actionscript Event class

```
1 package events
2 {
3     import flash.events.Event;
4     public class CustomEventAddToCompare extends Event
5     {
6         public var Name:String
7         public var Id:int;
8         public var Option:int;
9         public var DetailsOrSide:String;
10        public function CustomEventAddToCompare(type:String, name:String, id:int,
11            option:int, detailsorside:String, bubbles:Boolean=false,
12            cancelable:Boolean=false)
13        {
14            super(type, bubbles, cancelable);
```

```

15         this.Name = name;
16         this.Id = id;
17         this.Option = option;
18         this.DetailsOrSide = detailsorside;
19     }
20     override public function clone():Event{
21         return new CustomEventAddToCompare(type, Name, Id, Option, DetailsOrSide);
22     }
23 }
24 }

```

Listing 8.21 shows that in order to create a custom event, we need to extend the *Event* class. Also, we need to *override* the *clone* function used by the event framework to provide our custom implementation. Since every event in Flex is inherited from *flash.events.Event* class, it needs to be imported as shown in Listing 8.21. Within every actionscript class is a constructor with the same name as the class where data initialization can be done. In this case, *CustomEventAddToCompare()* function is the constructor of *CustomEventAddToCompare()* class. It takes a number of arguments. A few of these arguments: *type*, *bubbles* and *cancelable* come by default. However, the rest of the variables i.e. *Name*, *Id*, *Option* and *DetailsOrSide* are user-defined. In Listing 8.21, it can be seen that these variables have been defined with a global scope above our constructor. Therefore, in this class, we are sending data in four user defined variables to the parent component.

The meaning of *type*, *bubbles* and *cancelable* variables is as follows:

1. *type*:

This variable is a String which contains the name of the event. For example: 'click'. This property is set inside the event constructor.

2. *bubbles*:

This boolean variable specifies whether the event is a bubbling event. This is an optional parameter sent to the event constructor, By default, most event classes set this property to *false*.

3. *cancelable*:

This is a boolean variable which specifies whether the event can be canceled. This is an optional parameter passed to the Event constructor. By default, most event classes set this property to *false*, therefore specifying that the event cannot be cancelled.

Inside our constructor, we first call the super class to do some internal job. Then, we assign the incoming arguments to the global variables using *this* keyword. Afterwards. we override the *clone* function to create and return a copy of the current instance. Therefore, we can clone the instance to send the data we want, which in this case is the data contained in *type*, *Name*, *Id*, *Option* and *DetailsOrSide* variables.

#### 8.4.2 Using [Event] metadata tag in MXML

As previously mentioned, metadata tags are used to provide information to the Flex compiler that describes how a component will be used in a Flex application. After creating the *CustomEventAddToCompare* actionscript event class in the previous section, now we need to use *[Event]* metadata tag to let the compiler know that we will be sending an event from the custom item renderer component back to its parent

component. Listing 8.22 shows how to define the event in MXML and dispatch it.

Listing 8.22: Dispatching event in MXML

```
1 // Inside TileListItem.mxml
2 <mx:Metadata>
3     [Event(name="addToCompareEvent", type="events.CustomEventAddToCompare")]
4 </mx:Metadata>
5
6 <mx:Script>
7 <![CDATA[
8     import events.CustomEventAddToCompare;
9     private function buttonHandler(event:MouseEvent, option:int):void{
10         var customeventObject:CustomEventAddToCompare =
11             new CustomEventAddToCompare("addToCompareEvent", data.Name, data.Id,
12                 option, "Details", true, true);
13         dispatchEvent(customeventObject);
14     }
15 ]]>
16 </mx:Script>
17
18 <mx:Button visible="false" width="100%" click="buttonHandler(event, 1)"
19     icon="@Embed(source='pictures/flex_graph.png')" toolTip="Graphical Data"
20     id="buttonDetails"/>
21
22 // Inside Parent component dataAnalysis.mxml
23
24 <mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" initialize="init()"
25     xmlns:local="*" xmlns:maps="com.google.maps.*" width="100%"
26     height="100%" xmlns:components="components.*">
27
28 <mx:Script>
29 <![CDATA[
30     import events.CustomEventAddToCompare;
```

```

31     private function init():void{
32         iddataAllSites.addEventListener("addToCompareEvent", buttonHandler);
33         [...]
34     }
35
36     private function buttonHandler(event:CustomEventAddToCompare):void{
37         [...]
38         var obj:Object = new Object();
39         obj["Id"] = event.Id;
40         obj["Name"] = event.Name;
41         if (event.Option ==1){
42             /*Do Something*/
43         else if (event.Option ==2)
44             /*Do Something else*/
45         [...]
46     }
47     ]]>
48 </mx:Script>
49
50 <mx:TileList itemsChangeEffect="{tileListEffect}" dataProvider="{dataAllSites}"
51 paddingLeft="5" paddingRight="5" paddingTop="5" paddingBottom="5" width="100%"
52 itemRenderer="components.TileListItem" height="95%" id="iddataAllSites">
53 </mx:TileList>
54 [...]
55 </HBox>

```

In Listing 8.22, first we specify an *[Event]* metadata tag. Inside this tag, *addToCompareEvent* argument specifies the name of the event. The *CustomEventAddToCompare* argument specifies the class that defines the event. When we specify this metadata tag, the Flex compiler inserts the required code for enabling the component

*TileListItem.mxml* to register event listeners while compiling the application. After defining the event metadata, we then dispatch the event inside the *buttonHandler()* function. This function is called by the button with 'id' *buttonDetails* which sends it an integer and the event as arguments. Inside this function, we create a new instance of the class *CustomEventAddToCompare* and send it seven arguments, as required in the constructor of this class. The constructor was shown in Listing 8.21. Then, event returned from the *clone* method will contain five variables as mentioned in the previous section. After this cloned event is received in *customeventObject*, it needs to be dispatched. This is done using the *dispatchEvent()* function. After this event, we need to add an event listener inside the parent component, which in this case is *dataAnalysis.mxml*. Inside this parent component, we add an event listener as shown on Line of Listing 8.22. The event listener will be added when the *initialize* event of *dataAnalysis.mxml* parent component is triggered. *buttonHandler()* function handles this event in the manner shown in Listing 8.22.

With this, we finish the chapter on the Interactive Safety Belt Data Visualization Software.

## CHAPTER 9

### CONCLUSIONS, SUMMARY AND FUTURE WORK

#### 9.1 Summary

This thesis presented a new angle to approach traffic safety research. Apart from modeling and analysis in Injury Severity and Daytime Safety Belt Usage Surveys, emphasis was given on presenting the systems which visually interpret the collected data in these fields.

A literature survey of Injury Severity was performed and different factors were explored for increasing or decreasing Injury Severity. Then, a few models used in Injury Severity studies was presented to give an idea of the concepts involved. Then, injury severity and crash analysis systems were presented which could be queried not just to see the trends but also explore the connection between the variables.

A detailed account of Daytime Safety Belt Usage Surveys in Nevada was provided. Two new data collection softwares, iPhone and Windows Mobile 6 on Personal Digital Assistants, were designed for these surveys. Then, a comparison of these new softwares was done with paper and other templates that have been used in the past. The results showed that the new softwares revolutionized the speed of data collection while staying accurately consistent with the past data collection templates.

A conceptual feedback framework was developed to guide and control the Daytime Safety Belt Usage campaigns. A theoretical model was proposed to study the spread of the campaign information among the people.

Lastly, An interactive data visualization software was developed for Daytime Safety Belt Usage Surveys using the Adobe Flex 3 SDK. The explanation of the



codes written for iPhone application and the interactive data visualization software were presented and discussed.

## 9.2 Contributions

The contributions of this thesis work are listed below.

1. New data collection softwares using iOS SDK on iPhone and Windows Mobile 6 Professional SDK on PDA have been presented. The development of the PDA and iPhone 4 softwares revolutionizes the current trends in data collection.
2. A feedback framework has been proposed to control the safety belt campaigns year after year. A model is proposed to study campaign information diffusion through people and observe how different people respond to different messages in the campaigns.
3. A comparative study between five data collection templates is presented. The results showed that new designs(iPhone and 2010 PDA design) collect data at higher speed than other templates at similar accuracy levels. .
4. An Interactive Data Visualization Software has been designed and presented for Daytime Safety Belt Usage Surveys. This software is capable of analyzing past five years of data and presents the results in both, tabular and graphical formats.

### 9.3 Future Work

There are many areas of this thesis that can be enhanced by further research. These are listed below.

1. In Chapter 4, the proposed 2010 PDA and iPhone designs for data collection need to be enhanced with adaptive defaults, with the defaults selected on the basis of the area information and the incoming traffic.
2. In Chapter 5, a feedback framework was proposed to control the campaign design. Furthermore, a conceptual SEICRM model was proposed to study the spread of campaign information among the individuals. Numerical simulations for the SEICRM model and the feedback framework need to be performed. The parameters in the model equation need to be estimated from the relevant data.
3. In Chapter 6, an iPhone application was built for Daytime Safety Belt Usage Surveys. However, technical difficulties arose while transferring the data from the iPhone wirelessly over wifi/3G/4G. The application started quitting unexpectedly when trying to wirelessly send the data to the NUTC MySQL server. Thus, wireless concepts were not discussed in this thesis. Thus, there is a need to dig deeper and enable the wireless communication between iPhone application and MySQL servers.
4. In Chapter 8, an interactive database visualization software was built using the Adobe Flex 3 SDK. With the new upcoming releases of SDKs from Adobe, the existing code can be upgraded by incorporating newer definitions and components. Furthermore, an actionscript library needs to be added and programmed

in the Flex application to allow the users to directly print the obtained graphs and export the data in XLS/CSV or other formats.

- Abdel-Aty, M. (2003). Analysis of driver injury severity levels at multiple locations using ordered probit models. *Journal of Safety Research*, 34(5), 597–603.
- Adobe (2011a). Adobe flex 3 component explorer. Retrieved from Adobe Examples Website. <http://examples.adobe.com/flex3/componentexplorer/explorer.html>.
- Adobe (2011b). Adobe help resource center: About skinning. Retrieved from Adobe Flex 3 Help Website. [http://livedocs.adobe.com/flex/3/html/help.html?content=skinning\\_2.html](http://livedocs.adobe.com/flex/3/html/help.html?content=skinning_2.html).
- Adobe (2011c). Adobe help resource center: Embedding asset types. Retrieved from Adobe Flex 3 Help Website. [http://livedocs.adobe.com/flex/3/html/help.html?content=embed\\_4.html](http://livedocs.adobe.com/flex/3/html/help.html?content=embed_4.html).
- Adobe (2011d). Adobe help resource website. Retrieved from Adobe Flex 3 Help Website. <http://livedocs.adobe.com/flex/3/html/help.html>.
- Adobe (2011e). Creating an item renderer and item editor. Retrieved from Adobe Flex 3 Help Website. [http://livedocs.adobe.com/flex/3/html/help.html?content=cellrenderer\\_7.html](http://livedocs.adobe.com/flex/3/html/help.html?content=cellrenderer_7.html).
- Adobe (2011f). Download a free trial of flash builder 4.5 premium edition.

- Retrieved from Adobe Website. [https://www.adobe.com/cfusion/tdrc/index.cfm?product=flash\\_builder](https://www.adobe.com/cfusion/tdrc/index.cfm?product=flash_builder).
- Adobe (2011g). Free adobe flash platform technologies. Retrieved from Adobe Developer Connection Website. <http://www.adobe.com/devnet-apps/flex/free/>.
- Agency (2011). Proucl software. Retrieved from U.S. Environmental Protection Agency Website. <http://www.epa.gov/osp/hst1/tsc/software.htm>.
- Apple (2011a). Apple developer programs. Retrieved from Apple Developer Website. <http://developer.apple.com/programs/>.
- Apple (2011b). ios dev center. Retrieved from Apple Developer Website. <http://developer.apple.com/devcenter/ios>.
- Ballesteros, M., Dischinger, P., & Langenberg, P. (2004). Pedestrian injuries and vehicle type in maryland. *Accident Analysis and Prevention*, 36, 73–81.
- Boufous, S., Finch, C., Hayen, A., & Williamson, A. (2008). The impact of environmental, vehicle and driver characteristics on injury severity in older drivers hospitalized as a result a traffic crash. *Journal of Safety Research*, 39, 65–72.
- Bureau, U. C. (2005). Population distribution in 2005. Tech. rep.
- Cargoship (2007). Installing lamp on ubuntu for newbies. Retrieved from HowtoForge Website. [http://www.howtoforge.com/ubuntu\\_lamp\\_for\\_newbies](http://www.howtoforge.com/ubuntu_lamp_for_newbies).
- CDC (2008). About nvdrs. Retrieved from Centers for Disease Control and Prevention Website. <http://www.cdc.gov/ncipc/wisqars/NVDRS/About-NVDRS.htm>.

- CDC (2011). Wisqars. Retrieved from Centers for Disease Control and Prevention Website. <http://www.cdc.gov/injury/wisqars/index.html>.
- Chang, L.-Y., & Wang, H.-W. (2006). Analysis of traffic injury severity: An application of non-parametric classification tree techniques. *Accident Analysis & Prevention*, 38(5), 1019–1027.
- Chimba, D., & Sando, T. (2009). The prediction of highway traffic accident injury severity with neuromorphic techniques. *Advances in Transportation Studies*, 2009 A(19), 17–26.
- Conroy, C., Tominaga, G., Erwin, S., Pacyna, S., Velky, T., Kennedy, F., Sise, M., & Coimbra, R. (2008). The influence of vehicle damage on injury severity of drivers in head-on motor vehicle crashes. *Accident Analysis & Prevention*, 40, 1589–1594.
- Division, T. I. (2008). Annual traffic report. Tech. rep., Nevada Department of Transportation.
- Duncan, C. S., Khattak, A. J., & Council, F. M. (1998). Applying the ordered probit model to injury severity in truck-passenger car rear-end collisions. *Transportation Research Record*, 1635, 63–71.
- Eluru, N., C.Bhat, & Hensher, D. (2008). A mixed generalized ordered response model for examining pedestrian and bicyclist injury severity level in traffic crashes. *Accident Analysis and Prevention*, 40(3), 1033–1054.
- FCC (2010). Tvq tv database query. Retrieved from Federal Communications Commission Website. <http://transition.fcc.gov/fcc-bin/audio/tvq.html>.

Google (2011). Google map api for flash. Retrieved from Google Code Website.

<http://code.google.com/apis/maps/documentation/flash/>.

Gray, R., Quddus, M., & Evans, A. (2008). Injury-severity analysis of accidents involving young male drivers in great britain. *Journal of Safety Research*, 39, 483–495.

iOS (2009a). Core graphics framework reference. Retrieved from iOS Developer Library Website. [http://developer.apple.com/library/ios/#documentation/CoreGraphics/Reference/CoreGraphics\\_Framework/\\_index.html](http://developer.apple.com/library/ios/#documentation/CoreGraphics/Reference/CoreGraphics_Framework/_index.html).

iOS (2009b). Uinavigationcontroller class reference. Retrieved from iOS Developer Library Website. [http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UINavigationController\\_Class/Reference/Reference.html](http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UINavigationController_Class/Reference/Reference.html).

iOS (2009c). Uipickerviewdelegate protocol reference. Retrieved from iOS Developer Library Website. [http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIPickerViewDelegate\\_Protocol/Reference/UIPickerViewDelegate.html](http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIPickerViewDelegate_Protocol/Reference/UIPickerViewDelegate.html).

iOS (2010a). Foundation framework reference. Retrieved from iOS Developer Library Website. [http://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/ObjC\\_classic/\\_index.html](http://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/ObjC_classic/_index.html).

iOS (2010b). Uiapplicationdelegate protocol reference. Retrieved from iOS Developer Library Website. <http://developer.apple.com/library/ios/navigation/>.

iOS (2010c). Uapplicationdelegate protocol reference. Retrieved from iOS Developer Library Website. [http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIApplicationDelegate\\_Protocol/Reference/Reference.html](http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIApplicationDelegate_Protocol/Reference/Reference.html).

iOS (2010d). Uikit framework reference. Retrieved from iOS Developer Library Website. [http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIKit\\_Framework/\\_index.html](http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIKit_Framework/_index.html).

iOS (2011a). Uiviewcontroller class reference. Retrieved from iOS Developer Library Website. [http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIViewController\\_Class/Reference/Reference.html](http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIViewController_Class/Reference/Reference.html).

iOS (2011b). Uiwindow class reference. Retrieved from iOS Developer Library Website. [http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIWindow\\_Class/UIWindowClassReference/UIWindowClassReference.html](http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIWindow_Class/UIWindowClassReference/UIWindowClassReference.html).

iOS (2011c). View controller basics. Retrieved from iOS Developer Library Website. [http://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/AboutViewControllers/AboutViewControllers.html#//apple\\_ref/doc/uid/TP40007457-CH112-SW10](http://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/AboutViewControllers/AboutViewControllers.html#//apple_ref/doc/uid/TP40007457-CH112-SW10).

Keeling, M. J., & Rohani, P. (2007). *Modeling Infectious Diseases in Humans and Animals*. Princeton University Press.

Kim, J.-K., Ulfarsson, G., Shankar, V., & S.Kim (2008). Age and pedestrian injury



- severity in motor-vehicle crashes: a heteroskedastic logit analysis. *Accident Analysis and Prevention*, 40(5), 1695–1702.
- L. Breiman, R. O., J.H. Friedman, & Stone, C. (1998). *Classification and Regression Trees*. Chapman & Hall/CRC.
- Lafferty, J. (2009). Differences between flex 3 and flex 4. Retrieved from Adobe Developer Connection Website. [http://www.adobe.com/devnet/flex/articles/flex3and4\\_differences.html](http://www.adobe.com/devnet/flex/articles/flex3and4_differences.html).
- Lakhanpal, P. (2010). Interactive data visualation software. Retrieved from NUTC Website. <http://nutc.unlv.edu/PuneetLakhanpal/MainApplication.html>.
- Lee, C., & Abdel-Aty, M. (2005). Comprehensive analysis of vehicle-pedestrian crashes at intersections in florida. *Accident Analysis and Prevention*, 37, 775–786.
- M. Dalrymple, S. K. (2009). *Learn Objective-C on the Mac*. Apress.
- Manski, C., & McFadden, D. (1981). *Structure Analysis of Discrete Data with Econometric Applications*. Cambridge/MA: MIT.
- Mark, D., Nutting, J., & LaMarche, J. (2011). *Beginning iPhone 4 Development: Exploring the iOS SDK*. Apress.
- Martinez, R. (Spring 1998). The benefits of buckling up: Seat belts save lives. Seat belt survey Legislation, NHTSA. Retrieved from UAB Publications Website. <http://main.uab.edu/show.asp?durki=45962>.

- McKelvey, W., & Zavoina, T. (Summer 1975). A statistical model for analysis of original level dependent variables. *Journal of Mathematical Sociology*, (pp. 103–120).
- Microsoft (2007). Windows mobile 6 professional and standard software development kits refresh. Retrieved from Microsoft Download Center Website. <http://www.microsoft.com/download/en/default.aspx>.
- Microsoft (2011). Visual studio 2008 professional edition. Retrieved from Microsoft Dreamspark Website. <https://www.dreamspark.com/Products/Product.aspx?ProductId=1>.
- NCSA (2009). Traffic safety facts: Seat belt use in 2009-overall results. Tech. rep., National Highway Traffic Safety Administration.
- NDOT (2004). Functional classification maps. Retrieved from Nevada Department of Transportation Website. [http://www.nevadadot.com/reports\\_pubs/class\\_maps/](http://www.nevadadot.com/reports_pubs/class_maps/).
- Nevada (2008). State and county quickfacts. Retrieved from US Census Bureau Website. <http://quickfacts.census.gov/qfd/states/32000.html>.
- NHTSA (1998). Part 1340 - uniform criteria for state observational surveys of seat belt use. Retrieved from NHTSA website. [http://law.justia.com/us/cfr/title23/23cfr1340\\_main\\_02.html](http://law.justia.com/us/cfr/title23/23cfr1340_main_02.html).
- NHTSA (2009). Campaign history factsheet. Retrieved Traffic Safety Marketing

- Website. [http://www.trafficsafetymarketing.gov/ciot/planner2011/PEAK/7597\\_CIoT11\\_CampaignHistory-Factsheet\\_2-23\\_v2a.docx](http://www.trafficsafetymarketing.gov/ciot/planner2011/PEAK/7597_CIoT11_CampaignHistory-Factsheet_2-23_v2a.docx).
- NHTSA (2010). Lives saved in 2008 by restraint use and minimum drinking age laws. Tech. rep., U.S. Department of Transportation, Washington, DC.
- O'Donnell, C., & Connor, D. (1996). Predicting the severity of motor vehicle accident injuries using models of ordered multiple choice. *Accident Analysis and Prevention*, 28(6), 739–753.
- Pai, C. (2009). Motorcyclist injury severity in angle crashes at t-junctions: indentifying significant factors and analyzing what made motorists fail to yield to motorcycles. *Safety Science*, 47, 1097–1106.
- Pai, C., & Saleh, W. (2008). Modelling motorcyclist injury severity by various crash types at t-junction in the uk. *Safety Science*, 46, 1234–1247.
- Quddus, M., Wang, C., & Ison, S. (2009). The impact of road traffic congestion on crash severity using ordered response models. TRB 2009 Annual Meeting CD-ROM.
- RTC (2010). Freeway and arterial system of transportation. Retrieved from RTC Website. <http://www.nvfast.org>.
- Savolainen, P., & Mannering, F. (2007). Probabilistic models of motorcyclists' injury severities in single- and multi-vehicle crashes. *Accident Analysis and Prevention*, 39(5), 955–963.

- Shorten, A. (2011). What's new in flash builder 4.5. Retrieved from Adobe Developer Connection Website. [http://www.adobe.com/devnet/author\\_bios/andrew\\_shorten.html](http://www.adobe.com/devnet/author_bios/andrew_shorten.html).
- S.P. Washington, F. M., M.G. Karlaftis (2003). *Statistical and Econometric Methods for Transportation Data Analysis*. Chapman & Hall/CRC, Boca Raton.
- Sybase (2011). Register and download. Retrieved from Sql Anywhere Website. <http://www.sybase.com/detail?id=1016644&contentOnly=true>.
- Sze, N., & S.C, W. (2007). Diagnostic analysis of the logistic model for pedestrian injury severity in traffic crashes. *Accident Analysis and Prevention*, 39, 1267–1278.
- TMS (2003). About us and contact. Retrieved from Television Monitoring Services Website. <http://www.televisionmonitoring.com/>.
- U.S. Department of Transportation Federal Highway Administration, N. D. (2008). Annual vehicle miles of travel. Tech. rep., Nevada Department of Transportation.
- Vivoda, J., & Eby, D. (2006). Using personal digital assistants (pdas) for collection of safety belt use data in the field. *Behavior Research Methods, Instruments and Computers*, 38, 158–164.
- w3schools (2011a). Css reference. Retrieved from w3schools Website. [http://www.w3schools.com/css/css\\_reference.asp](http://www.w3schools.com/css/css_reference.asp).
- w3schools (2011b). Sql. Retrieved from w3schools Website. <http://www.w3schools.com/sql>.

WampServer (2011). Download wamp. Retrieved from WampServer Website. <http://www.wampserver.com/download.php>.

Y. Xie, Y. Z., & Liang, F. (2009a). Crash injury severity analysis using bayesian ordered probit model. *Journal of Transportation Engineering*, 135(1), 18–25.

Y. Xie, Y. Z., & Liang, F. (2009b). Crash injury severity analysis using bayesian ordered probit models. *Journal of Transportation Engineering*, 135(1), 18–25.

## VITA

Graduate College  
University of Nevada, Las Vegas

Puneet Lakhanpal

### Degrees:

- Bachelor of Technology, Electronics and Communication Engineering, 2009  
Indian Institute of Technology, Guwahati

### Special Honors and Awards:

- 1<sup>st</sup> place, Fall Transportation Conference UNLV, FALL 2009. 'Driver response to In-vehicle device interaction and unexpected incidents'
- 1<sup>st</sup> place, Graduate Celebration Poster Competition, SPRING 2011. 'An Optimal Data Entry System using PDAs for Seat-belt Usage Studies'.

### Publications:

- Lakhanpal P., Sriom S., Khurana M., Robust Text Independent speaker recognition based on hybrid LPC and MFCC algorithm. Proceedings of IEEE, 15<sup>th</sup> International Conference on Advanced Computing and Communication, 2007.

Thesis Title: Traffic Safety: Modeling, Analysis and Visualization

### Thesis Examination Committee:

- Chairperson, Pushkin Kachroo, Ph.D., P.E.
- Committee Member, Masha Wilson, Ph.D.
- Committee Member, Rama Venkat, Ph.D.
- Graduate Faculty Representative, Monika Neda, Ph.D.