Development Environment Setup and User Manual

for

AGORA

Version 1.1

A Versatile Environment for the Development of IntelliDrive Applications

(Visual … Extensible … Rule-Based)

Department of Computer Science, Western Michigan University



Table of Contents

Table of Contents ..... **ii**
Revision History ..... **ii**
1. Introduction **1**
1.1 Purpose 1
1.2 Intended Audience 1
1.3 References 1
2. Installation Procedure **1**
2.1 VIS/A3/Gov 2.0 1
2.2 TMC 6
3. User Manual **9**
3.1 Gov 2.0 9
3.2 A3 11
3.3 Carbon Emissions Tracker 11
3.4 Smart Cone 13
3.5 Course/Fine Context 14
3.6 VOIP Call 15
3.7 Police Car/Ambulance/Fire Truck Alert 16
3.8 Tow Truck Alert 18
3.9 Sphinx 19
Appendix A: Glossary **20**

Revision History

Name	Date	Reason For Changes	Version
Vinay B Gavirangaswamy	01/24/2010	Initial Draft	1.0
Vinay B Gavirangaswamy	02/16/2010	Changed project name from VII to AGORA, changed paragraph formatting, and reformatted cover page.	1.1

1. Introduction

1.1 Purpose

AGORA refers to a set of application and infrastructure which constitutes intellidrive environment that is being developed at Computer Science Department, Western Michigan University. This project consists of several software applications and hardware components. This infrastructure consists of On-Board Equipment (OBE), Road Side Equipment (RSE), and Traffic Management Centers (TMC), which work together to increase the safety and efficiency of the transportation network.
This document is intended to help internal and external team member to get up to speed with the development environment, installation procedures and user manual for the different systems that are being developed.

1.2 Intended Audience

Document is primarily intended for members of MDOT team which consists of graduate students working under the guidance of Dr. Ala Al-Fuqaha and Dr. Dionysios Kountanis.

1.3 References

- http://www.oracle.com/us/technologies/java/index.htm
- http://netbeans.org/
- http://www.mysql.com/
- http://www.eclipse.org/
- http://javafx.com/
- http://www8.garmin.com/support
- https://glassfish.dev.java.net/

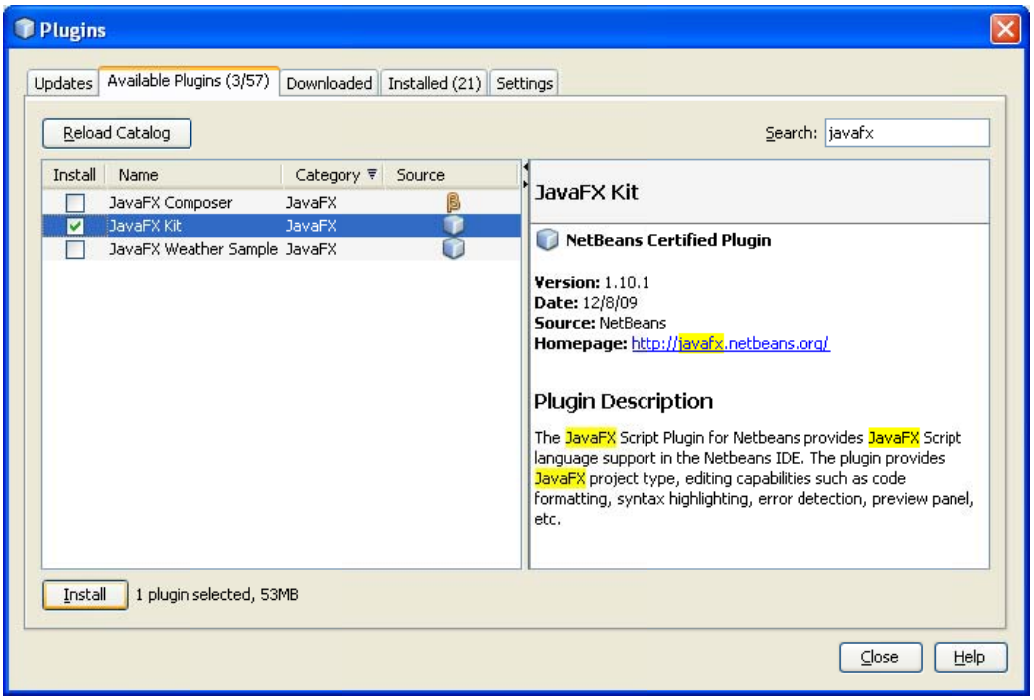
2. Installation Procedure

2.1 VIS/A3/Gov 2.0

Operating System: Microsoft Windows XP Professional Service Pack 3

Install JDK 6 Update 17 with NetBeans 6.8
1. Download from http://java.sun.com/javase/downloads/widget/jdk_netbeans.jsp
2. Run installer (accept all defaults).

Install JAVAFX Plugin for NetBeans
1. Open NetBeans.
2. Open Plugins Manager Tools > Plugins
3. Switch to Available Plugin tab and type *javafx* in the search box.
4. Check *JavaFX Kit* box.
5. Click Install (accept all defaults).



Install Spanner (used for accessing Garmin GPS)

1. Download from http://www8.garmin.com/support/download_details.jsp?id=1627
2. Unzip to desired folder (default: C:\garmin).
3. Run SpannerSetup (accept all defaults).

Add Spanner as a Startup Program

1. Copy Spanner program

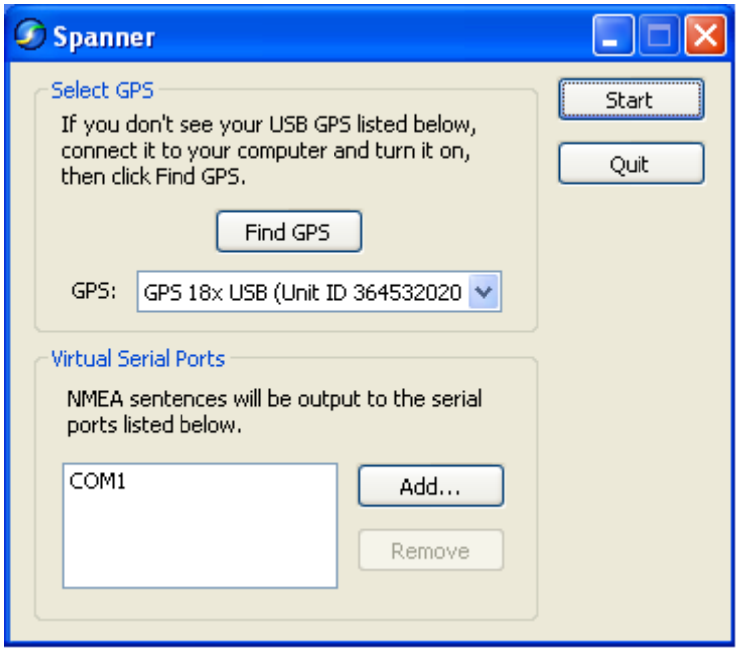
Start > All Programs > Garmin > Spanner > Right-click > Copy

2. Paste to Startup Folder

C:\Documents and Settings\All Users\Start Menu\Programs\Startup > Right-click > Paste

Test GPS Connectivity

1. Attach the GPS device to an available USB port.
2. Run the Spanner program.
3. Select the GPS from the drop down box.
4. Click the Start button.

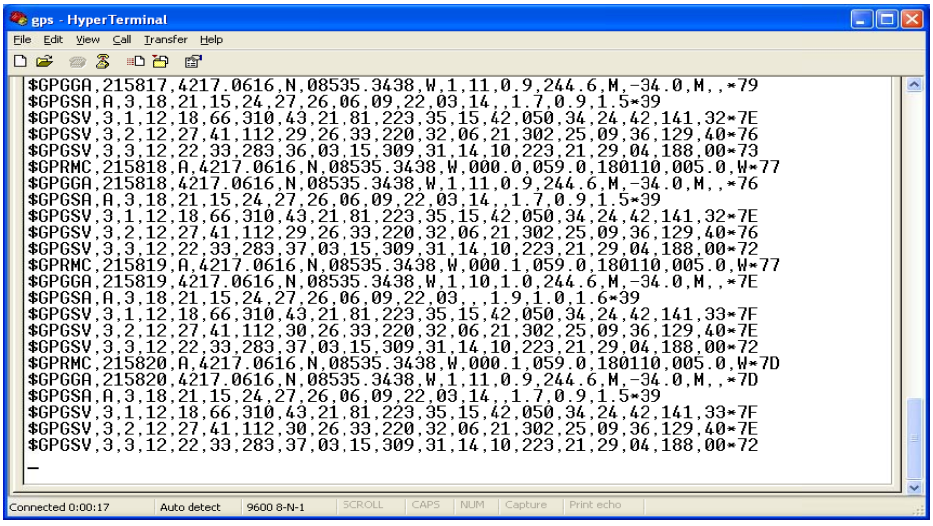


5. Open HyperTerminal

Start > All Programs > Accessories > Communications > HyperTerminal

6. Type *gps* in the Name box.
7. Click OK.
8. Connect using: COM1
9. Click OK.
10. Change Bits per second to 9600.
11. Click OK.

*If a connection to the GPS is made, data should be displayed similar the image below.



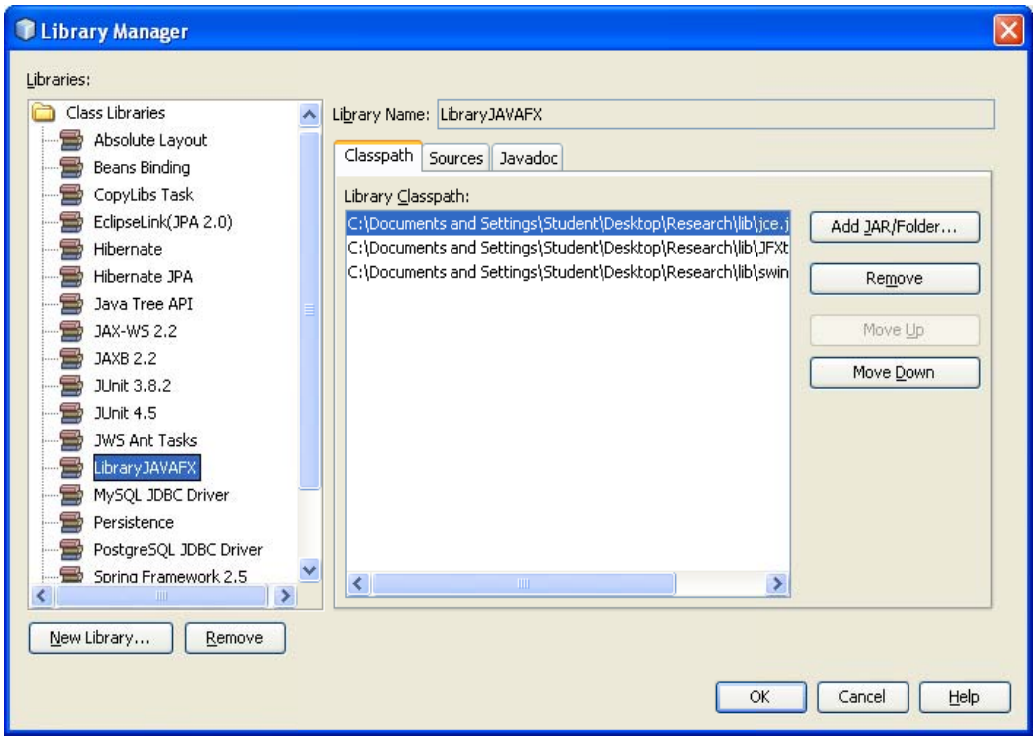
Install Java Media Framework (JMF) 2.1.1e

1. Download from http://java.sun.com/javase/technologies/desktop/media/jmf/2.1.1/download.html

2. Run installer (accept all defaults).

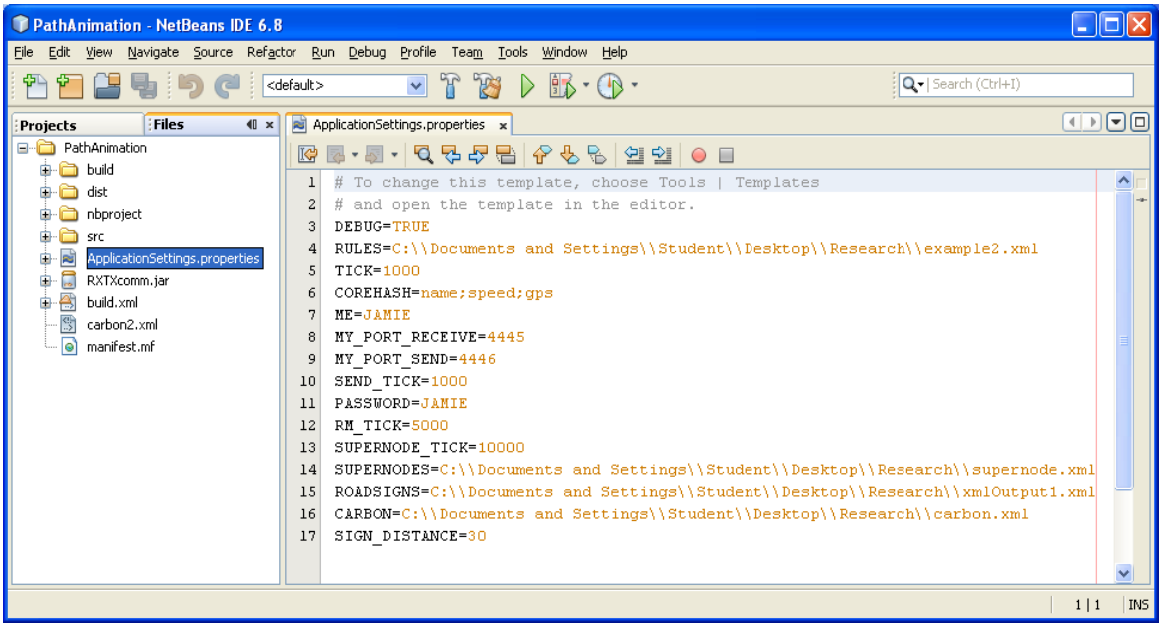
Add OBDII Library:
1. Copy rxtxSerial.dll to C:\Documents and Settings\Student\.netbeans\6.8\javafx-sdk\lib\desktop

Setup MDOT Project in NetBeans
1. Copy MDOT files/folders to desired location.
(Default: C:\Documents and Settings\Student\Desktop\Research)
2. Open NetBeans.
3. Add new project.
File > Open Project > (browse to mdot folder)/PathAnimationNew > Open Project
4. Create LibraryJAVAFX
Expand PathAnimation in Projects Tab
Right-Click Libraries > Add Library > Manage Libraries > New Library
Library Name: LibraryJAVAFX
Library Type: Class Libraries
Click OK.
Add JAR/Folder > (browse to MDOT directory)/lib
Select: jce.jar
JFXtras-0.5.jar
swing-layout-1.0.3.jar
Add JAR/Folder
Click OK.
Select LibraryJAVAFX from Add Library window > Add Library



5. Change path in Application Settings File.
Click on the Files tab.

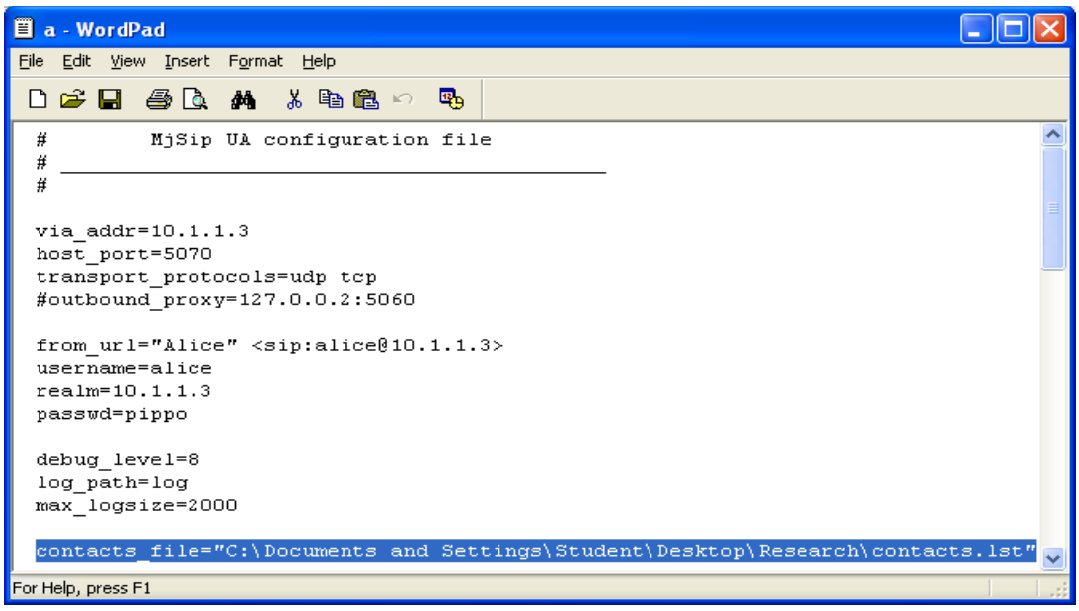
Expand PathAnimation.
Double click ApplicationSettings.properties file.
Change path of RULES, SUPERNODES, ROADSIGNS to point to corresponding xml files. (example2.xml, supernode.xml, xmlOutput.xml - found in MDOT folder)



Change Path/Settings in VOIP config file:

1. Create a folder named *log* in the Research folder.
2. Edit .cfg file (in Research folder) that corresponds to a VOIP user (a.cfg or b.cfg)

contacts_file="C:\Documents and Settings\Student\Desktop\Research\contacts.lst"



Note: If building and running the NetBeans project, you get the error message:

Unable to start java.exe: The system cannot find the file specified.

Java did not get installed on the system (even though it should be in the NetBeans bundle). Download and install Java 6 Update 18 from http://www.java.com/en/download/manual.jsp

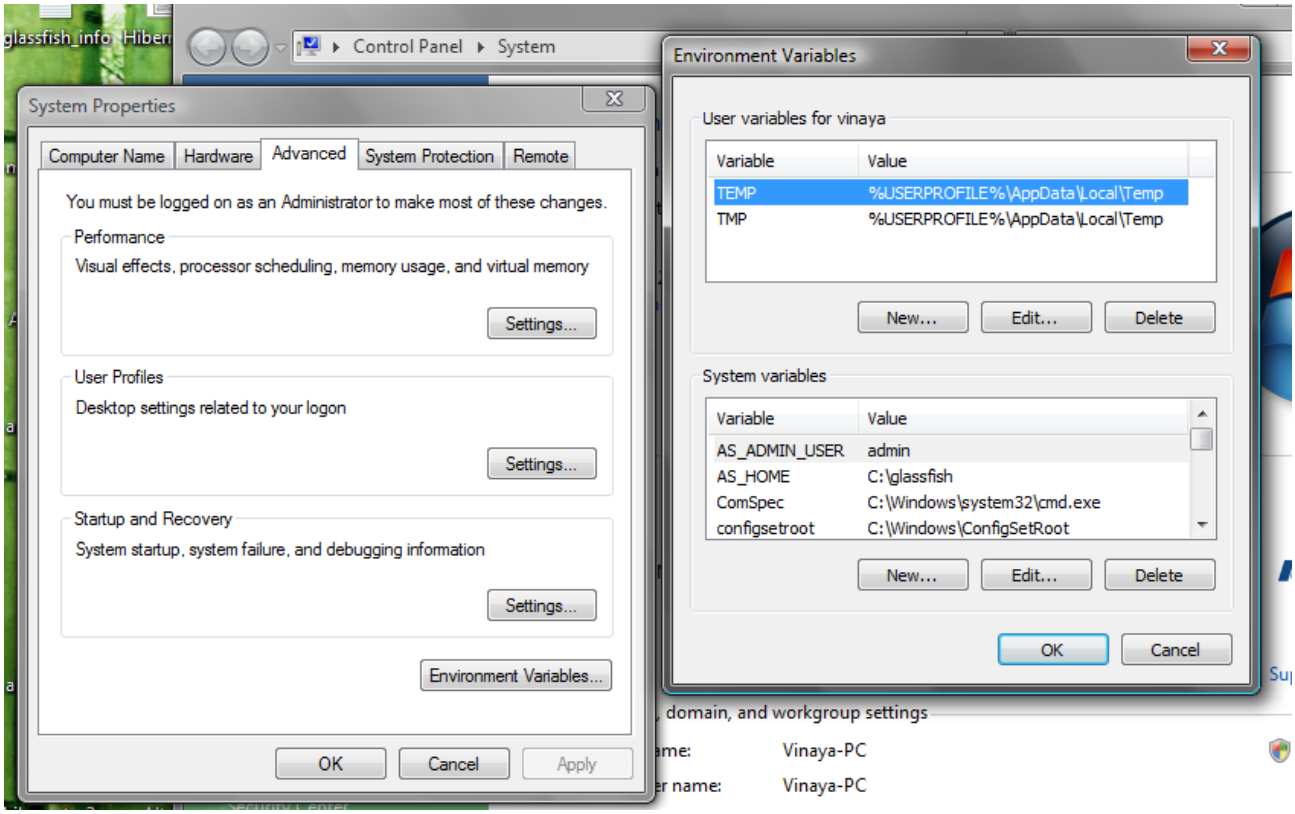
2.2 TMC

Download and install

Glassfish

Version: 2.1.1
URL: https://glassfish.dev.java.net/public/downloadsindex.html
Install location: c:\glassfish
Install instruction: Please refer to glassfish web site for install instructions
Environment variables:
Please setup following system environment variable as shown below

Variable Name	Value
AS_ADMIN_USER	admin
AS_HOME	C:\glassfish
JAVA_HOME	<your system java install directory>

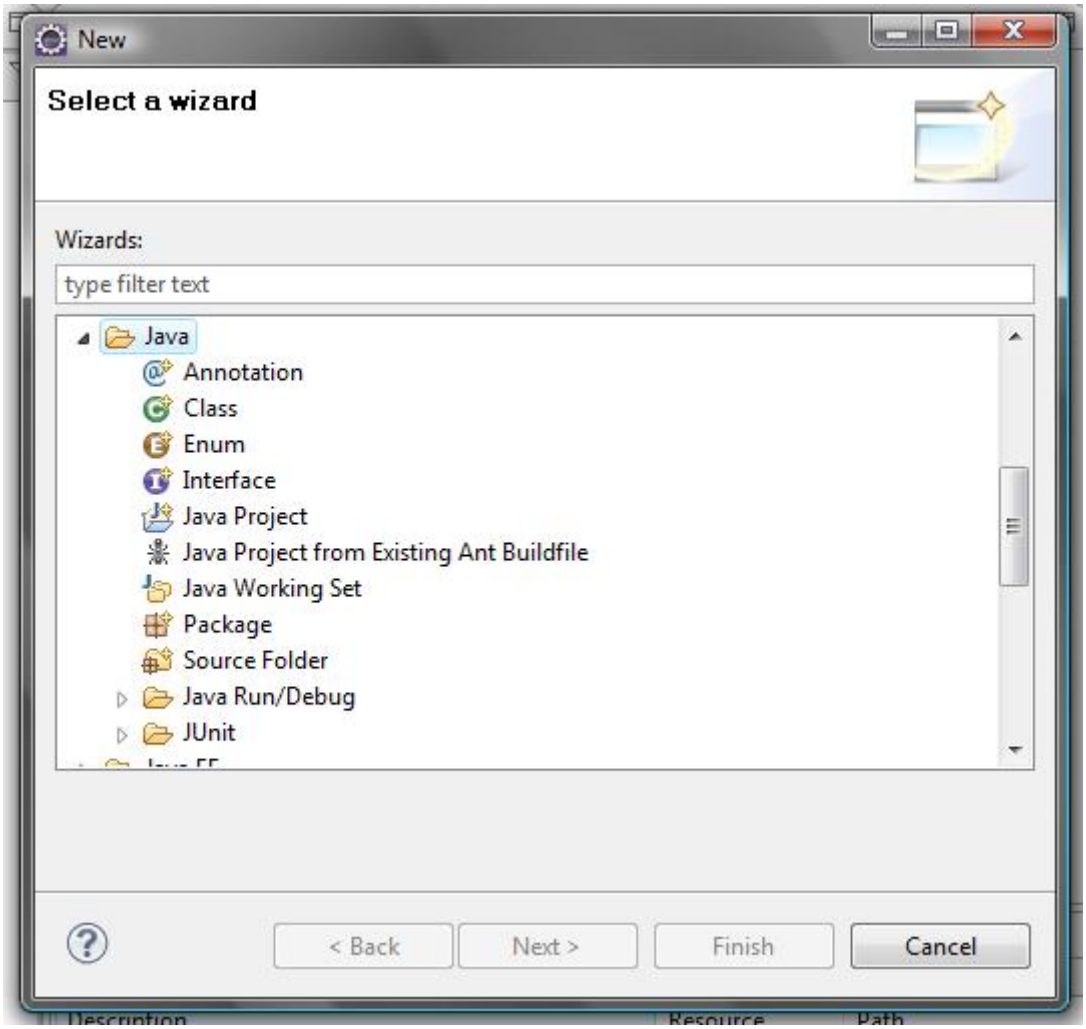


Eclipse Java EE IDE for Web Developers

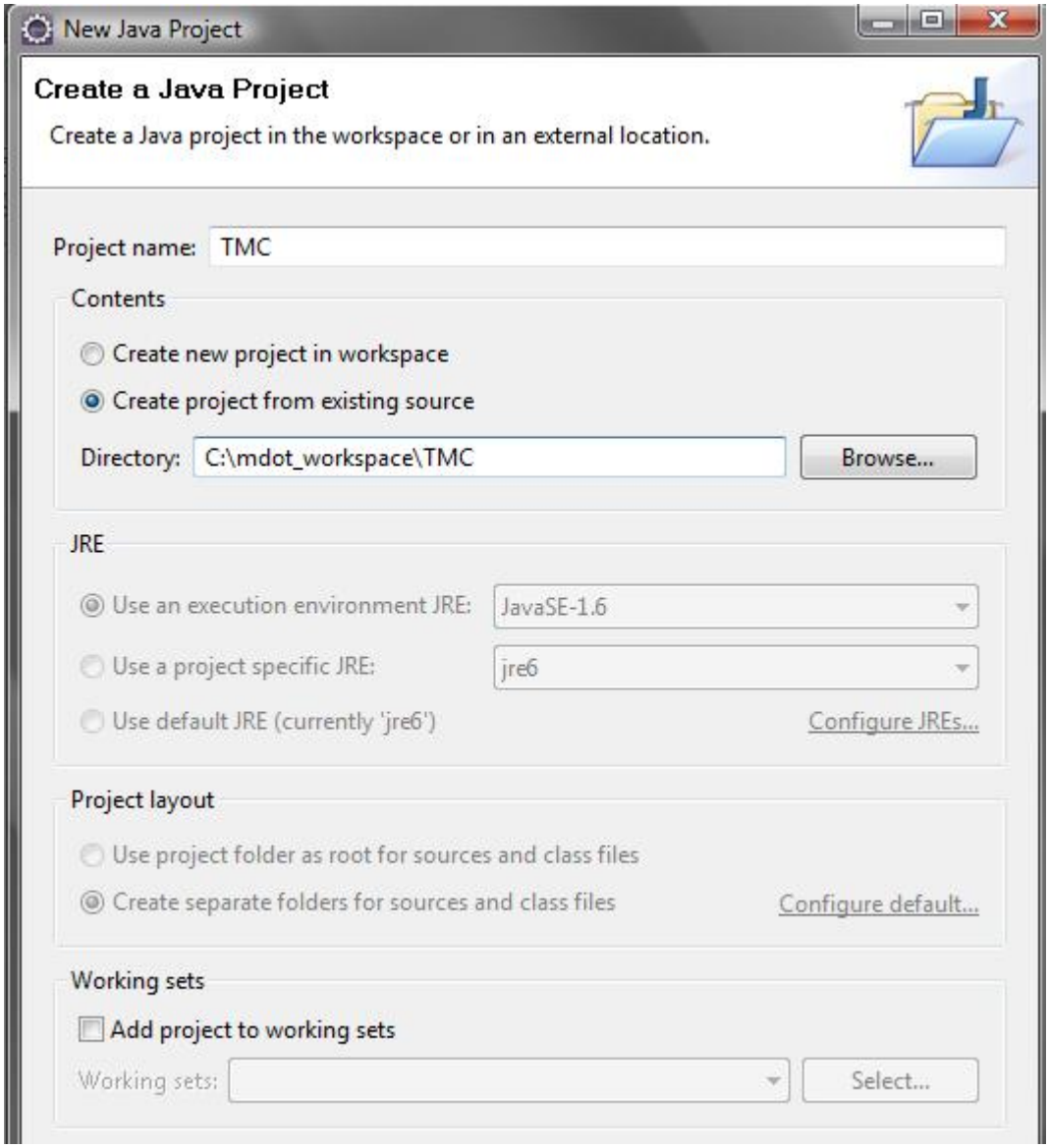
Version: any
URL: http://www.eclipse.org/downloads/
Install location: c:\eclipse
Install instruction: unzip downloaded archive to above specified directory

TMC Application Setup:

Copy TMC.zip to a directory called c:\mdot_workspace
Upzip archieve
Open eclipse and select c:\mdot_workspace as your workspace
Goto file->new->other



And enter following information in the next screen



Accept default options for the rest and click Finish.

Build and deploy Instructions:

Run ant script under TMC\build\build.xml. This will create a war archived file under TMC\bin\war. Deploy this war file in glassfish application server.
Note: To read on how to deploy application on glassfish server please refer to appropriate section of glassfish application sever documentation.

3. User Manual

3.1 Gov 2.0

3.1.1 Navigation Instructions

Application Icon →Gov 2.0 Icon

3.1.2 Screen Shots

Application Selection Screen:



Figure 1: Gov 2.0 selection in VIS

Description:

The above picture shows the applications currently available in VIS, where the exclamation sign icon opens the Advisory Alert application (A3) and the speaker icon opens the Gov 2.0 application.

Gov 2.0 Application Screen:

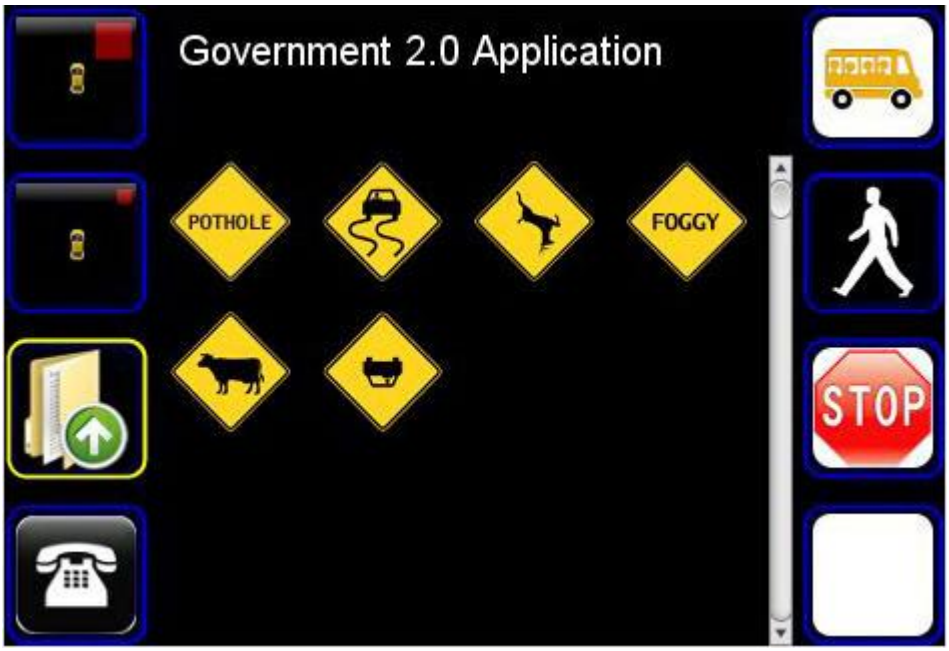


Figure 2: Gov 2.0

Description:

The above picture shows the Gov 2.0 application. This application displays all the hazards, as icons, a user can encounter in the transportation system. The application interface is kept simple for ease of use with icons showing the available hazards. On encountering a hazard, a user can click (touch) the corresponding icon, which is then sent to and registered with TMC. Now, based on the frequency of a certain hazard being reported (at a particular location), the transportation authority can prioritize these hazards and take measures accordingly. This information is also conveyed to the users of VIS through the on-demand A3 application (or dynamic signs/alerts).

3.2 A3

3.2.1 Navigation Instructions

Application Icon →A3 Icon

A3 Application Screen:



Figure 3: A3

Description:

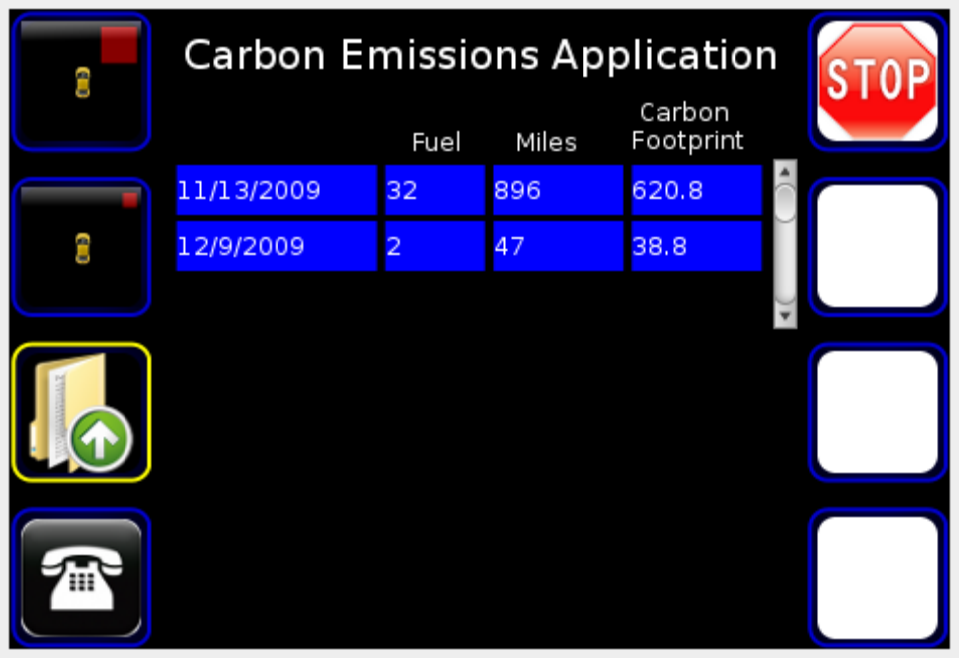
The above picture shows the Advisory Alert application (A3). This application displays a list of hazards based on the current location of the vehicle, retrieved from TMC. The application interface is divided into two parts, the alert icon and the alert description. In the above picture of A3, the left column shows the hazard icons for quick visual feedback to the user. On the right column is the description of the corresponding hazard. For example, the snap-shot above shows a pothole, an icy condition and a dead animal hazard.

3.3 Carbon Emissions Tracker

3.3.1 Navigation Instructions

Application Icon →Car Emissions Tracker Icon

Carbon Emissions Tracker Application Screen:



Description:

With the increasing threat of global warming, individuals are becoming more conscious about their impact on the environment. The Carbon Emissions Tracker allows individuals to track the amount of carbon their vehicle puts into the atmosphere each time they turn the car on by simply displaying it on the screen in a trip-by-trip format.

How It Works:

The Carbon Emissions Tracker calculates the miles traveled, fuel used and carbon foot print for each “trip” which is defined to be the time from which the engine is started, to the time it shut off. The number of miles traveled is calculated using the GPS device which is attached to the system. Every 10 seconds, the GPS is polled for the new location of the car. The distance between the new latitude and longitude location and the old latitude and longitude location is then calculated and added to the running distance total.

The Carbon Emissions Tracker is able to calculate the amount of fuel consumed during a trip by using data read from the OBD (On-Board Diagnostics) System that is standard on cars manufactured starting in 1996.

The values of mass air flow and speed are requested from the OBD and then plugged into the following equation which returns the instantaneous miles per gallon (MPG) of the vehicle.

Instantaneous MPG = (14.7 * 6.17 * 454 * SPEED) / (3600 * MASS AIR FLOW)

- 14.7 grams of air to 1 gram of gasoline - ideal air/fuel ratio
- 6.17 pounds per gallon - density of gasoline
- 454 grams per pound - conversion
- 3600 seconds per hour conversion

To find the average MPG during a trip, the instantaneous rate is averaged over time.

The carbon footprint during a trip requires a few calculated values. First, to calculate the CO2 emissions from a gallon of fuel, the carbon emissions are multiplied by the ratio of the molecular weight of CO2 (m.w. 44) to the molecular weight of carbon (m.w.12): 44/12.

CO2 emissions from a gallon of gasoline = 2,421 grams x 0.99 x (44/12)

= 8,788 grams

= 8.8 kg/gallon

= 19.4 pounds/gallon

Note: If C02 for emissions for diesel is need, substitute 2421 grams with 2778 grams.

Next, number of gallons used is calculated by dividing miles traveled by the average MPG. Finally the C02 emissions constant (19.4)is multiplied by the gallons which results in the carbon footprint.

The values of all the calculated variables for a trip are written to an XML file (carbon.xml) for easy storage and retrieval. The values automatically populate the grid when the application starts, so no interaction is needed by the driver.

3.4 Smart Cone

3.4.1 Navigation Instructions

Application Icon →Smart Cone Icon

Smart Cone Application Screen:



Figure 5: Smart Cone Application

Description:

Construction zones can quickly produce hazardous conditions for both drivers and construction workers. The goal of the Smart Cone application is to quickly create an “alert” area warns the driver that they are in a construction zone and should be on the lookout and slow down. With the Smart

Cone application, and the click of a button, a construction zone can be communicated rapidly, and removed just as quickly.

How It Works:

The Smart Cone application is simply a way of changing a vehicle “type” from a standard vehicle type to a “Smart Cone” type. When the green go button is clicked in the application, the type value in the vehicle's hash table is update to be the Smart Cone type. This information is then propagated in the vehicle's hash table to the neighboring vehicles via the process described in section 2.3. Neighboring vehicles look through their context in search of the “Smart Cone”, and if it is found, displays the construction cone icon in their alert area.

Similarly, when the application's stop button is clicked, the vehicle type is changed back to that of a standard vehicle, and the propagation process through the hash table is again followed.

3.5 Course/Fine Context

3.5.1 Navigation Instructions

Application Icon →Course/Fine Context Icon

Course/Fine Application Screen:



Description:

With the ever increasing volume of traffic, it is important for a driver to know his/her location with respect to other vehicles. This becomes increasingly important during hazardous weather conditions and rush hour. The Course/Fine Context applications allow for visual representation of the car's context; where there are in relation to other cars on the road; with a series of red flashing squares representing neighboring vehicles.

How It Works:

The Course/Fine Context applications get information from the core services of the system. When a car receives data from a neighboring vehicle, it compares the latitude and longitude locations and "decides" where the vehicle is in relation to itself. The car then places this information into that car's local hash table. Every 500ms, the GUI thread checks all the vehicles in the local hash context for this location information. It then generates a list of squares that should be turned red, and proceeds to do so. The same process repeats indefinitely.

The only difference between the Course and Fine Context applications is the number of squares. The Course Context has 8 locations, while the Fine Context has 48. Since the Fine Context squares are smaller, it is able to represent vehicles that are farther away.

3.6 VOIP Call

3.6.1 Navigation Instructions

Application Icon →VOIP Call Icon

VOIP Call Application Screen:



Description:

In the case of an emergency, having a way to call for help is essential. The VOIP Call application allows the system to send and receive calls to registered IP addresses which could for example be that of a dispatch center.

How It Works:

When the user clicks the "call" button on the application, the system uses MjSip (open source java-based implementation of a SIP stack available at www.mjsip.org) to initiate a connection between the caller and the callee. Once the connection is established, the voice data is processed with Java Media Framework (JMF).

3.7 Police Car/Ambulance/Fire Truck Alert

3.7.1 Navigation Instructions

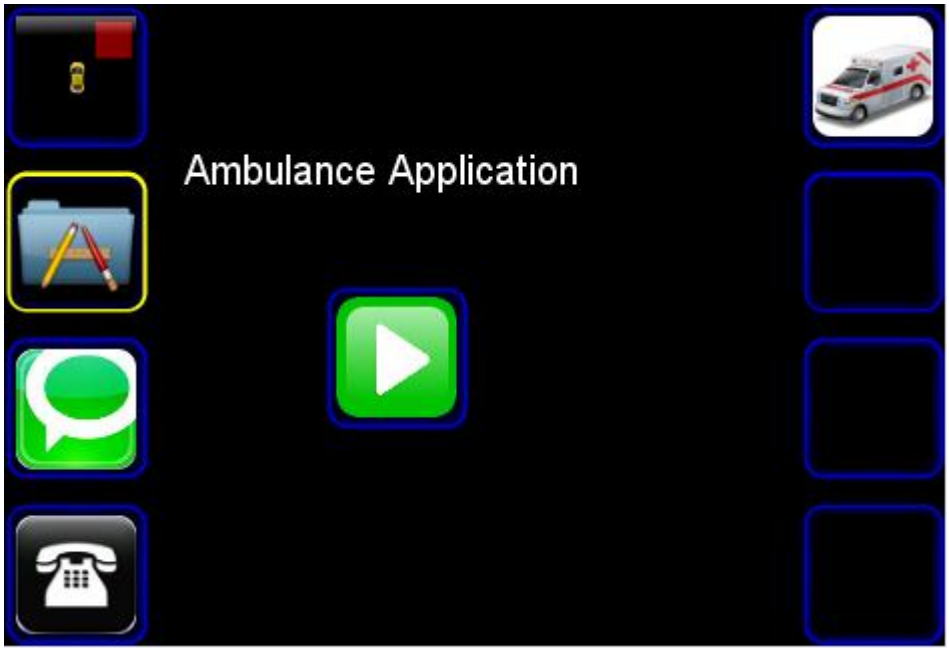
Application Icon →Police Car/Ambulance/Fire Truck Alert Icon

Police Car/Ambulance/Fire Truck Alert Application Screen:

Police:



Ambulance:



Fire:



Description:

When there is an accident, fire, or any other event that requires emergency personnel, response time is critical as seconds can save lives. The purpose of the Police Car, Ambulance, and Fire Truck Alert applications is to alert the driver that one of these vehicles is approaching and proper protocol should be followed as is the law of the state. In Michigan for example, the driver should slow down, pull over and stop while these vehicles pass.

How It Works:

The Police Car/Ambulance/Fire Truck Alert applications are very similar to the Smart Cone and School Bus Applications in that when the green go button is pushed, the vehicle type is changed from a standard vehicle to a police car, ambulance or fire truck. This information is propagated to the vehicle's neighbors via the process described in section 2.3. The vehicle type remains the specified type until the stop button on the application is pressed. When the application is running, neighbors will see a vehicle of type police car, ambulance or fire truck in their context and display the associated icon in the alert area.

3.8 Tow Truck Alert

3.8.1 Navigation Instructions

Application Icon → Tow Truck Alert Icon

Tow Truck Alert Application Screen:



Description:

When a Tow Truck is called to the scene of an accident or other related event, it is often the case that a partial lane blockage will follow while the truck maneuvers the disabled vehicle into proper position. These lane blockages can be hazardous to other drivers, especially during times of high traffic or sharp and windy roads. The sooner that other driver can know about the Tow Truck, the more time they have to react and reduce speed or change lanes. The Tow Truck application is used to alert drivers to the above mentioned situations.

How It Works:

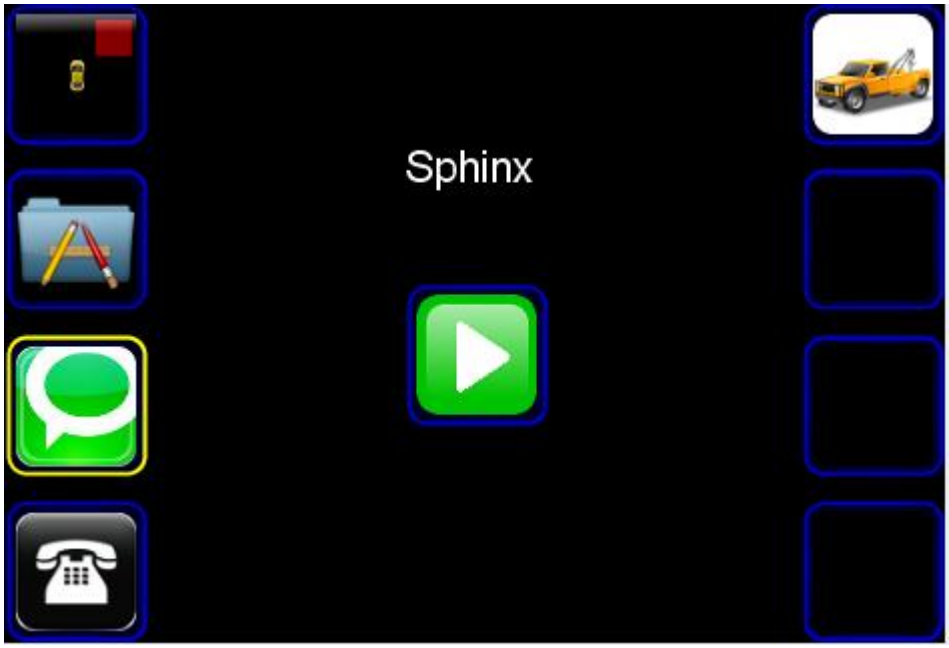
The Tow Truck application is very similar to the Police Car/Ambulance/Fire Truck Alert in that when the green go button is pushed, the vehicle type is changed from a standard vehicle to a tow truck. This information is propagated to the vehicle's neighbors via the process described in section 2.3. The vehicle type remains “tow truck” until the stop button on the application is pressed. When the application is running, neighbors will see a vehicle of type tow truck in their context and display the associated icon in the alert area.

3.9 Sphinx

3.9.1 Navigation Instructions

Application Icon → Sphinx Icon

Sphinx Application Screen:



Description:

Hands free operation is critical when it comes to software running in vehicles. The fewer things that a driver has to control create a safer environment for everyone. The integration of Sphinx, a speech recognizer developed by the Sphinx group at Carnegie Mellon University, allows the driver to switch between screens using simple single word voice commands.

How It Works:

When the user clicks the start button on the application, the system activates Sphinx4 and waits for the user to speak one of the recognized words. When the system hears a word that it knows, the main screen is changed to screen associated with that word. Sphinx continues to operate until the user clicks the stop button. Currently, the system uses the following words to change between the main application screens:

Words	**Screen**
application	Application
course	Course Context
fine	Fine Context
call	VOIP Call

voice	Sphinx

Appendix A: Glossary

MDOT – Michigan Department of Transportation
OBE – On-Board Equipment
RSE – Road-Side Equipment
TMC – Traffic Management Centers
A3– Advisory Alert Application
LSA – Life Safety Application
VIS – Vehicle Integrated Software
OSIL- Operating System Isolation Layer
OS- Operation System
CAL- Communication Abstraction Layer
DSRC- Dedicated short range communications