Pedestrian Mid-block Crossing Application User Guide

Pedestrian Mid-block Crossing Application User Guide

Version 1.0.1

2/6/2018

Leidos

# Revision History

|  |  |  |
| --- | --- | --- |
| Revision # | Author | Revision |
| 1 | Kyle Rush | Initial release of document |
| 2 | Kyle Rush | Rename of application |

Contents

[Revision History 2](#_Toc504464414)

[Overview 4](#_Toc504464415)

[Scope 4](#_Toc504464416)

[Installation 5](#_Toc504464417)

[Dependencies 5](#_Toc504464418)

[Compilation 5](#_Toc504464419)

[Server 5](#_Toc504464420)

[Android Application 5](#_Toc504464421)

[Deployment 6](#_Toc504464422)

[Server 6](#_Toc504464423)

[Android Application 6](#_Toc504464424)

[Configuration 7](#_Toc504464425)

[Server 7](#_Toc504464426)

[Android Application 8](#_Toc504464427)

[Usage 9](#_Toc504464428)

[Server 9](#_Toc504464429)

[Android Application 9](#_Toc504464430)

[FAQ 13](#_Toc504464431)

[Troubleshooting 13](#_Toc504464432)

[Figure 1 The Android application's initial screen 9](#_Toc504463796)

[Figure 2 Pedestrian mode 10](#_Toc504463797)

[Figure 3 Additional menu contents 11](#_Toc504463798)

[Figure 4 The Driver mode of the Android application 12](#_Toc504463799)

# Overview

The Pedestrian Mid-block Crossing Application is designed to allow pedestrians at equipped midblock crosswalks to communicate their intent to cross to drivers approaching the crosswalk. The software system consists of one Android application with two modes (referred to as “Pedestrian” and “Driver) and a cloud server. These devices communicate through a 4G cellular data connection using a RESTFUL HTTP API to exchange JSON messages. The application in Pedestrian mode allows the user to send a request for permission to cross to the server. If the server accepts the Pedestrian’s request it sends this message out to all Driver mode devices querying the server for status. If the driver device receives an alert and judges itself to be approaching the intersection (based on configurable geofences received from the server) then it will display an audio-visual alert to the driver, notifying them of the pedestrian’s presence. This app is not intended to function in any regulatory capacity. It only aims to notify the driver and enable them to make informed decisions, thus the alert it provides is tailored towards providing this information rather than providing instructions. When the Driver mode is not displaying an alert it displays a Google maps view centered on the device’s location to look like a standard in-vehicle navigation app.

This application was used to conduct research on driver awareness of pedestrians at Turner-Fairbanks Highway Research Center. As such, the application includes data collection capabilities for research analysis. Both devices transmit their position and heading back to the server on a regular basis. The server logs this data to a file for later processing and tracks for several events to occur (entering or exiting a geofence, device registration, etc.) which are also logged with timestamps.

# Scope

The intent of this document is to provide context on the application and to provide instructions and information with regards to the installation, configuration and operation of the software. The source code itself is documented so details of that level are not covered here. This document is targeted for an audience familiar with software development, particularly that of Java and Android development. This document assumes the reader has a basic level of familiarity with Android and Linux and is comfortable with the usage of a system terminal or command line. For the purposes of server deployment this guide assumes the server itself to already be configured and will only detail the deployment and configuration of the software itself, rather than the server.

# Installation

## Dependencies

Compiling the Pedestrian Mid-block Crossing Applicaiton requires the installation of Gradle version 2.0 or higher and the Java Development Kit version 8.

## Compilation

### Server

The server component of the application is compiled via a standard Gradle build process enhanced by Spring Boot, which is pulled in at build time. To run this build process run:

*cd server*

*gradle bootRepackage*

from the root folder of the Pedestrian Mid-block Crossing Application OSADP distribution. This build process will automatically download any Java dependencies that may be required by the server component. After the compilation is complete the output JAR file should be found at “crossing-request-main/build/libs/crossing-request-main-1.0-SNAPSHOT.jar”, relative to the root of the distribution. This JAR file should be fully executable and contain all that is needed to run the application.

### Android Application

The Android application component can be compiled using either the Android Studio or stand-alone Gradle or Gradle wrapper. It is recommended to use the Android Studio interface to create a signed release APK for the application to be installed on the devices, instructions can be found for this at: <https://developer.android.com/studio/publish/app-signing.html>. To build a debug build from Gradle directly run the following commands from the root of the distribution:

*cd app*

*gradlew assembleDebug*

This will generate a debug-level APK file at “app/app/build/outputs/apk” relative to the root of the distribution. This debug APK can be loaded onto a device via the Android Device Bridge, however the configuration needed to do so (drivers, etc.) depends on the manufacturer of the device. This document will assume that the above steps for generating a signed release build were followed when going into details on deployment.

## Deployment

### Server

The server component can be deployed to any cloud hosting environment capable of running Java webservers. The server component expects to be able to communicate over port 5000 for its HTTP interface so the firewall configuration for the server should allow for this. Otherwise the server requires no special system permissions and may be run as any user.

The application may be deployed to any folder on the host machine as long as a “data” and a “datalogs” folder are created in the working directory of the deployed JAR file. These folders will be used to store the geofence configuration and log data respectively. The user running the JAR file should have read/write access to both of these folders and any files inside them. Once these folders are created the “pedestrian\_geofence.json” and “motorist\_geofence.json” files from the “server/crossing-request-main/data” folder should be deployed into the “data” folder.

For information on server configuration and setup on Amazon Web Services (as an example cloud host) please see the AWS documentation for Elastic Beanstalk setup: <https://docs.aws.amazon.com/console/quickstarts> Please ensure to follow the instructions for setting up a Java platform environment.

### Android Application

The Android application may be deployed to the target device by connecting it to a PC for file transfer (exact procedure to do so may vary by manufacturer). Copy the compiled and signed release APK from the PC to the device and then use the device’s file manager to locate and install the APK file. The target device must have installation of apps from unknown sources enabled and any restrictions against side-loading of applications must be disabled.

# Configuration

## Server

The server component utilizes three configuration files to determine its behavior at runtime: a Java properties file and two geofence JSON files. The “application.properties” located at “server/crossing-request-main/src/main/resources/application.properties” has the following parameters available for configuration:

*notification.duration\_s:* The number of seconds the pedestrian’s request should remain active for, as a floating point value

*server.port:* The port to use for the HTTP REST API, as an integer value

*geometry.pedDescriptionFile:* The relative or full path to the geofence file for the pedestrian app mode

*geometry.driverDescriptionFile:* The relative or full path to the geofence file for the driver app mode

*logging.eventFile:* The relative or full path to the file where the server should record event data

*logging.dataFile:* The relative or full path to the file where the server should record event data

*logging.file:* The relative or full path to where standard server logs should be recorded

*logging.datapath:* The relative or full path to where the server should store newly generated data files and event files (after eventFile and dataFile have been rolled over)

The JSON files both use the same format, the following is an example of this format:

{

"geofenceName": "driverGeofence",

"geofenceUniqueID": "driverDirectionalMultiGeofence103017",

"geofenceModificationDate": "10/30/2017 11:16PM",

"role": "MOTORIST",

"regions": [

{

"heading": 270.0,

"headingEpsilon": 89.0,

"vertices": [

{"latitude": 38.955146, "longitude": -77.148921},

{"latitude": 38.954746, "longitude": -77.148986},

{"latitude": 38.954788, "longitude": -77.147489},

{"latitude": 38.955336, "longitude": -77.147564}

]

},

{

"heading": 90.0,

"headingEpsilon": 89.0,

"vertices": [

{"latitude": 38.954860, "longitude": -77.148893},

{"latitude": 38.954965, "longitude": -77.148891},

{"latitude": 38.955926, "longitude": -77.150049},

{"latitude": 38.955201, "longitude": -77.150134}

]

}

]

}

The geofenceName, geofenceUniqueID, geofenceModificationDate, and role fields are all strictly metadata for human record keeping. The geofenceName field is a human readable name for the geofence, the geofenceUniqueIdD is a unique identifier for the geofence in its current state, the geofenceModificationDate is a string date stamp for the last modified time of the current geofence, and the role field indicates whether it is for pedestrian or driver. The remainder of the fields, under the regions array, describe the geometric regions to which the geofence pertains. This particular example has two regions – approaches from both directions – but the format can support one or many. Each region consists of a heading and headingEpsilon pair, as well as an array of latitude and longitude coordinates. The heading and heading epsilon describe the directions of travel to which the region applies, any heading within the range [heading – headingEpsilon, heading + headingEpsilon] is considered to be compliant (or inside). The array of vertices contains latitude and longitude pairs for the vertices of the geofenced region. These points, walked in-order, form the borders of the geofence region. These points must form a convex polygon or the algorithms used by the software to determine “insideness” may perform inaccurately. The arrays shown here all contain four coordinates but any number greater than three may be used by adding additional elements to the array. A vehicle is considered inside the geofence if it’s heading is in the aforementioned range **and** it’s latitude and longitude fall within the bounds of the vertices.

## Android Application

The Android application has no runtime configuration but must be updated with the IP address or URL of the server instance being used. To do so edit the file “app/app/src/main/java/gov/dot/fhwa/saxton/crossingrequest/utils/Constants.java” and change the value *“serverBaseUrl”* on line 21 to contain either the IP address or URL of the server, then recompile and reload the application to the device. This value is initially configured as “127.0.0.1” or “localhost” and must be configured with your server IP prior to usage.

# Usage

## Server

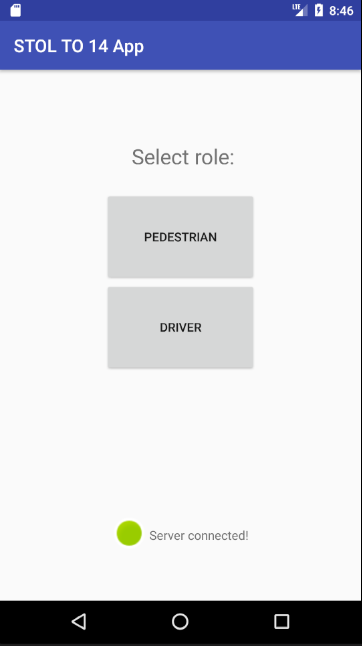
The server may be started by simply running the following command from the folder where you installed the JAR file, “data” folder, and “datalogs” folder:

*java -jar crossing-request-main-1.0-SNAPSHOT.jar*

Once running, the server requires minimal interaction from the user and may be left running for as long as is needed. It is recommended to set the server to startup on system boot up and to restart in the event of a crash to ensure that it is always available to the devices if they attempt to connect.

## Android Application

The Android application may be launched by clicking the “TO 14 App” icon on your device’s app drawer or home screen. This will bring up the following screen:



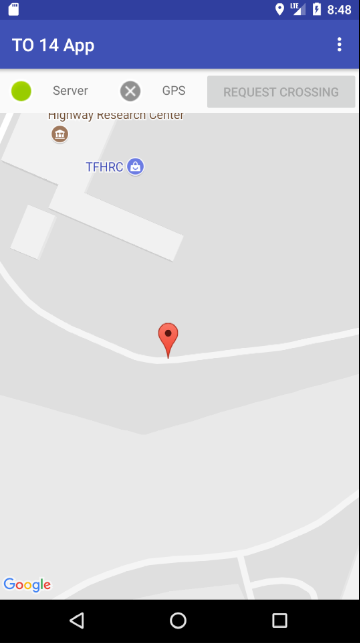
Server status indicator

Role selector buttons

Figure The Android application's initial screen

From this screen you can select which mode the device should operate in, pedestrian or driver. This screen also shows the current connection status with the server. If the device can successfully send an initial status message to the server the indicator will be green as shown. If the device is unsuccessful at communicating it will grey out and the text will say “server disconnected”.

Upon pressing the “PEDESTRIAN” button to select the pedestrian mode for the device, the following screen will be shown:



Additional menu

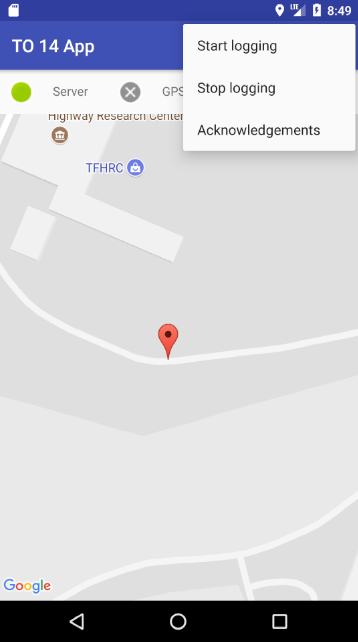
User’s Location

Request Crossing Button

Status indicators

Figure Pedestrian mode

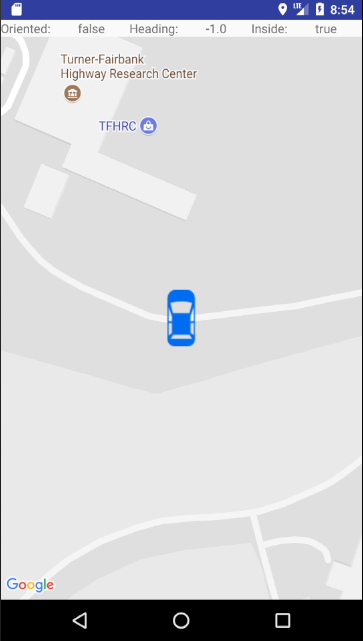
In this mode, the user’s current location is shown as a pin on a Google Map representation of the local area. The server status indicator from the initial screen persists here, shown in the upper left corner of the app window. Additionally there is now a GPS status indicator directly next to it showing whether a GPS status update has been received from the device within the last three seconds. There is also a “REQUEST CROSSING” button shown in the top right of the application window. This button is greyed out until the user enters the configured pedestrian geofence which will be displayed as a black outlined region on the map. Upon entering the geofence the button will become available and allow the user to send a crossing request to the server. If the user presses the Request Crossing button while inside the geofence the device will transmit the request to the server where it will be validated. Upon successful validation of the request the device will display a notification confirming the acceptance and the Request Crossing button will disable and turn pink for the duration of the broadcast.



Additional menu

Figure Additional menu contents

The pedestrian mode screen also has an additional menu were other functionality can be accessed. By clocking the three dots icon in the top right of the application window the menu shown in Figure 3 will be displayed. “Start logging” causes the server to roll over all future data logs and event logs to new files marked with the current time and datestamp. “Stop logging” ends data collection for the current log files and causes the server to discard all data that would be logged until the “Start logging” button is pressed. Acknowledgements displays additional licensing information for the graphics used in the driver view of the application.



User location

Figure The Driver mode of the Android application

The driver mode of the application is designed to mimic the interface of common in-vehicle navigation software. The vehicle icon is oriented according to the devices reported heading by rotating the map such that the current heading is towards the top of the device. As the location updates the view will move and rotate to display the new locations and headings. When the device receives a crossing request broadcast that it deems to be relevant (based on geofence parameters) it will display a notification from the top of the screen containing the text “Pedestrian Ahead” with images of standard crosswalk signage accompanied by a beep and text-to-speech “Pedestrian Ahead”.

# FAQ

* Question: Can the server be configured for multiple crosswalks?
  + The server only supports a single crosswalk but can support multiple approaches for a single crosswalk.
* Question: Can more than one device be used in driver mode simultaneously?
  + The server can communicate with any number of driver devices as each device evaluates its geofence locally prior to displaying any alerts.
* Question: Can more than one device be used in pedestrian mode simultaneously?
  + The server can handle multiple pedestrian devices as well. Only one crossing request per cross walk may be active at a time but any connected device can initiate the crossing request.
* Question: Can the data logging functionality be disabled or turned off?
  + Yes, if “Stop logging” is pressed on the Pedestrian mode of the Android application the server will stop recording any data it may receive from the pedestrian and/or driver devices. Please note that this does not stop the data from being sent and unless the server and devices are configured for secure communications this data may be visible to other users on the network.

# Troubleshooting

* *Problem:* The driver device does not receive an alert when approaching an active cross walk request.
  + *Possible solution:* Examine the geofence file for the driver geofence, ensure that the geofence is properly constructed such that the driver is in the fence when they should receive the alert.
  + *Possible solution:* Check the cellular network connectivity of the device. Use the Pedestrian mode of the device to check if the device can successfully communicate with the server.
  + *Possible solution:* Check the GPS accuracy of the device, ensure that the actual location of the device lines up closely with the displayed location of the device on the Driver mode interface. Check that location services using GPS are enabled in the Android OS settings.
* *Problem:* The pedestrian device cannot successfully request a crossing from the server.
  + *Possible solution:* Check the cellular connectivity of the device; ensure that the indicator on the pedestrian screen is green to indicate successful communication with the server.
  + *Possible solution:* Check the GPS accuracy of the device; ensure that the GPS location closely matches the user’s real location, and that both are within the outlined geofence region.
  + *Possible solution:* Ensure that a crossing request is not currently active for the crosswalk as only one crossing request may be active at a time.
* *Problem:* The devices cannot load the geofence from the server.
  + *Possible solution:* Ensure that the geofence files are stored in the location configured in the “application.properties” configuration file.
  + *Possible solution:* Check that the filesystem permissions for the folder in which the geofence files are stored allow the user running the server to read the files.
  + *Possible solution:* Check the cellular connectivity of the devices on the initial screen of the application.