TravelTrak - Installation

# Installation Requirements - Development

* Eclipse IDE (others may be used as well. This document covers instructions for Eclipse).
  + Download from <https://www.eclipse.org/downloads/>
* Java JDK 6 or newer
* MySQL 5 or newer
* Additional Required Libraries:
  + mysql.jar (<https://dev.mysql.com/downloads/connector/j/> - Platform Independent .zip version

# Installation Instructions into Development Environment

1. Install MySQL
   1. Save Database URL, Port, Username and Password
2. Install Eclipse
3. Open Eclipse
4. To Import Project to Eclipse
   1. Select File -> Import.
   2. Expand “General”, choose “Existing Projects into Workspace” and select Next
   3. Choose “Select archive file”, click browse and select your archive file.
   4. It should show up in the “Projects” window and be selected.
   5. Select Finish.
5. Add Required Libraries to project (if no errors are indicated).
   1. Right click on Project Name in the Package Explorer
      1. If “Package Explorer” or “Project Explorer” is not open, do the following:
         1. Select Window -> Show View -> Project Explorer
   2. Select “Properties”
   3. Select “Java Build Path” from left menu area
   4. Switch to the “Libraries” tab
   5. If you see “mysql.jar” and it has a red “x” on it, click on it and click “Remove”
   6. If you have not done so, add the Jar to the project in the workspace (drag and drop to the file into the project)
   7. Select “Add JARs…” and choose the jar file you wish to add.
   8. Do this for any required JAR files, and then exit.
6. Update DBURL, DBPORT, DBUSER, DBPASS in DatabaseStatics.java to match what was saved upon installing MySQL
7. Open MySQL program to manage the database
   1. Create a database for use with TravelTrak.
   2. Update the DBNAME field in DatabaseStatics.java to match the new database name
   3. Select the new database and execute the SQL script called “Initial\_DB\_Insert.sql” from the project workspace.
8. Run the program.

# Export Instructions - From Development To Production

1. Right click on Project Name and select “Export”
2. Select “Java” -> “Runnable JAR File”
3. In Launch Configuration, choose the correct class name from the corresponding project.
4. Choose an export Destination
5. Select “Extract required libraries into generated JAR”
6. Click Finish.

# Installation Requirements - Production

* Java JRE 6 or newer
* MySQL 5 or newer
* JAR file of TravelTrak program created from Eclipse Export.

# Installation Instructions - Production

1. Double click on the JAR file that was exported from the development environment.

# Project Tasks to be completed:

* Create a front-end website to manage the devices and present the data from the program.
* Integrate other devices into the program
* Update database to support additional devices
* Need to create a configuration file for the program to allow all user-configurable parameters be changed easily without having to re-compile.

TravelTrak - Program Flow

This following section is organized by the flow of the data from the time it is collected to the time it is discarded. For a more detailed description of the functions mentioned, view the comments in the appropriate file.

# Startup.java

***Purpose:***

This is the primary class that starts up the program with main(). Its primary task is to start up the necessary threads so that they may start and be ready to process data as it comes in.

***Managed Threads:***

* LoggerThread.java
* MacAddressSensorSengment.java
* MacAddressTravelTimeProcessor.java
* MacAddressUniqueIDProcessor.java
* MacAddressDataProcessor.java
* DataProcessingDaemon.java
* DeviceDaemon.java

***Data Passed:***

* None

Once these threads have started, there is nothing else that this class is responsible for.

# LoggerThread.java

***Purpose:***

This is responsible for logging any data to the filesystem while the program is running. Each running thread is provided access to the method addTolist(), which will accept log data and write it to the appropriate file. View LogItem.java to see what the values passed are.

***Managed Threads:***

* None.

***Data Passed:***

* None

# DeviceDaemon.java

***Purpose:***

This class is responsible for managing the various devices and ensuring that their threads have been started or stopped accordingly. The primary method called in this class is run() and continually runs as long as the thread is alive. Within run, it continually checks to see if any of the following occurs:

* Any new devices have been added to the database
* Any devices that were running have gone offline
* Any devices that were previously offline have since came back online
* Notifies the MacAddressSensorSegmentProcessor.java Thread of any new segments that may have been created or removed due to device status changes.

***Managed Threads:***

* QueryDeviceThread.java
* DeviceThread.java

***Additional Threads Accessed:***

* LoggerThread.java
* MacAddressSensorSegmentProcessor.java

***Data Passed:***

* MacAddressSensorSegmentProcessor.java
  + Passes a MacAddressSensorSegment object indicating what the segment link is.

# BluetoothDeviceThread.java

***Purpose:***

This class is responsible for connecting to, retrieving data, and closing network connections to a particular device in the field. The primary method called in this class is run() and continually runs as long as the thread is alive. Within run, it performs the following:

* Attempts to connect to a device
* Attempts to read data from the device
* Attempts to close the connection to the device.
* If the thread has multiple consecutive failed connection attempts, the thread will automatically shutdown (this is handled within DeviceThread.java).

***Managed Threads:***

* None.

***Additional Threads Accessed:***

* LoggerThread.java

***Data Passed:***

* None. Data passing is handled from the parent class, DeviceThread.java

***Notes:***

* This thread inherits and uses methods and variables defined in DeviceThread.java.
* This thread is also a model thread for allowing the program to run with additional types of devices, such as C1 or Acyclia Wifi (there is an Acyclia Wifi thread created, but not discussed in this document).

# DeviceThread.java

***Purpose:***

This class contains the core methods and variables necessary for any inherited classes to function. Core methods used by child classes:

* Connecting/Disconnecting to/from a device
* Sending data received to another thread for processing.

***Managed Threads:***

* None.

***Additional Threads Accessed:***

* LoggerThread.java
* DataProcessingDaemon.java

***Data Passed:***

* DataProcessingDaemon.java
  + Sends a RawMacData.java object to the processing thread for initial analysis.

***Notes:***

* This class has been designed to handle additional types of devices as well that run off of a network connection. For example, looking in the method sendDataForProcessing(), you will notice two else/if blocks that are empty pertaining to C1 and AW. These would be populated with code necessary for sending C1 data or Acyclia Wifi data out for processing. They would have their own classes called (for example) RawC1Data.java and RawWiFiData.java.

# DataProcessingDaemon.java

***Purpose:***

This class is the first stage of processing of any data received from devices. Its ultimate purpose is to look at the data received from a device, and determine which processing thread the data will be sent to.

***Managed Threads:***

* None.

***Additional Threads Accessed:***

* LoggerThread.java
* MacAddressDataProcessor.java
* *Note:* Additional processing threads would also be access here. For example, C1 data would be sent to C1DataProcessor.java, as the C1 data would not have the same type of processing done to it as Mac Addresses would.

***Data Passed:***

* MacAddressDataProcessor.java
  + Sends a RawMacData.java object to the Mac Address Processing Thread for initial analysis.
* *Note:*If C1 data processing was present, it would send (for example) a RawC1Data.java object to C1ProcessingThread.java for further processing.

***Notes:***

* None.

# MacAddressDataProcessor.java

***Purpose:***

This class takes the RawMacData.java object and converts it into a MacAddressData.java object which is the final format for the data that is passed around.

***Managed Threads:***

* None.

***Additional Threads Accessed:***

* LoggerThread.java
* MacAddressUniqueIDProcessor.java

***Data Passed:***

* MacAddressUniqueIDProcessor.java
  + Sends a MacAddressData.java object to the MacAddressUniqueIDProcessor.java for further processing.

***Notes:***

* None.

# MacAddressUniqueIDProcessor.java

***Purpose:***

This class takes the MacAddressData.java object (MAD) and attempts to re-identify the mac address into something unique so that the raw mac address is not passed around. If an existing mac address has been found, its unique ID is updated, otherwise a new one is assigned.

***Managed Threads:***

* None.

***Additional Threads Accessed:***

* LoggerThread.java
* MacAddressTravelTimeProcessor.java

***Data Passed:***

* MacAddressTravelTimeProcessor.java
  + Sends a UniqueMacAddress.java object to the MacAddressUniqueIDProcessor.java for further processing.

***Notes:***

* This class needs to be updated to remove the original mac address once a unique identifier has been assigned. The original mac has been kept for debugging purposes.

# MacAddressUniqueIDProcessor.java

***Purpose:***

This class takes the MacAddressData.java object (MAD) and attempts to re-identify the mac address into something unique so that the raw mac address is not passed around. If an existing mac address has been found, its unique ID is updated, otherwise a new one is assigned.

***Managed Threads:***

* None.

***Additional Threads Accessed:***

* LoggerThread.java
* MacAddressTravelTimeProcessor.java

***Data Passed:***

* MacAddressTravelTimeProcessor.java
  + Sends a UniqueMacAddress.java object to the MacAddressTravelTimeProcessor.java for further processing.

***Notes:***

* This class needs to be updated to remove the original mac address once a unique identifier has been assigned. The original mac has been kept for debugging purposes.

# MacAddressTravelTimeProcessor.java

***Purpose:***

This class sorts MacAddressData.java objects into various sub-threads for further processing. There are currently 8 sub-threads that are spawned by this class, each one takes a small chunk of mac address data to manage during processing.

***Managed Threads:***

* MacAddressTravelTimeProcessorSubThread.java.

***Additional Threads Accessed:***

* LoggerThread.java

***Data Passed:***

* MacAddressTravelTimeProcessorSubThread.java
  + Sends a MacAddressData.java object to a MacAddressTravelTimeProcessorSubThread.java thread to be processed.

***Notes:***

* None.

# MacAddressTravelTimeProcessorSubThread.java

***Purpose:***

This class organizes all MacAddressData.java objects into MacAddressTripInformation.java objects to allow for easy monitoring of the data being received and to help identify trip information. This class also watches to see if the MacAddressTripInformation.java object should be removed from the system, indicating the end of a trip.

***Managed Threads:***

* None.

***Additional Threads Accessed:***

* LoggerThread.java
* MacAddressUniqueIDProcessor.java
* MacAddressSensorSegmentProcessor.java

***Data Passed:***

* MacAddressUniqueIDProcessor.java
  + Sends the Unique Identifier of a MacAddressTripInformation.java object back to the MacAddressUniqueIDProcessor.java thread so that it may be removed from the system. This occurs when a trip is no longer deemed valid for a variety of reasons.
* MacAddressSensorSegmentProcessor.java
  + Sends a MacAddressTravelTimePair.java object to the MacAddressSensorSegmentProcessor.java thread so that the MacAddressTravelTimePair.java object can be determined valid or invalid.

***Notes:***

* This thread has not been throughly tested and is to be considered pre-alpha. At the time of this documents writing, there were not enough sensors out in the field to accurately test how effective this class worked.

# MacAddressSensorSegmentProcessor.java

***Purpose:***

This class takes MacAddressTravelTimePair.java objects and analyzes them to determine if the trip time is valid based.

***Managed Threads:***

* None.

***Additional Threads Accessed:***

* LoggerThread.java
* MacAddressTravelTimeProcessorSubThread.java

***Data Passed:***

* MacAddressTravelTimeProcessorSubThread.java
  + Notifies the MacAddressTravelTimeProcessorSubThread.java using the MacAddressTravelTimePair.java object of the results from the analysis.

***Notes:***

* This thread has not been throughly tested and is to be considered pre-alpha. At the time of this documents writing, there were not enough sensors out in the field to accurately test how effective this class worked.