Motorbike Statistics

Generated by Doxygen 1.8.13

Contents

Chapter 1

motorbikestatistics

Motorcycle statistics device for analysing rider performance

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

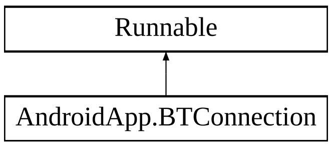
Chapter 5

Class Documentation

5.1 AndroidApp.BTConnection Class Reference

Thread class for a new bluetooth connection to a device.

Inheritance diagram for AndroidApp.BTConnection:



Public Member Functions

- BTConnection (BluetoothDevice btDevice) throws IOException

  Constructor for BTConnection class.
- void setRXHandler (Handler newHandler)

  Setter function for RXHandler.
- void run ()

  Main run procedure for new Runnable thread created.
- void stop ()

  Procedure to stop the bluetooth connection thread from running.
- boolean isRunning ()

  Function to check whether main connection thread is running.
- boolean isConnected ()

  Function to check whether BT connection is still valid.

Public Attributes

- final Handler txHandler

  Handler class for transmission of data.

Private Member Functions

- void connect () throws IOException

  Procedure to create a connection to logging device.
- void close () throws IOException

  Closes the BT connection socket, exceptions thrown on failure.

Private Attributes

- BluetoothDevice btDevice

  Bluetooth Device object, holds information for chosen slave.
- Handler RXHandler = null

  Handler function where received data is sent to.
- BluetoothSocket btSocket = null

  Socket created for bluetooth connection, used for TX/RX.
- volatile boolean running = false

  Indicates whether main run thread is in progress.

Static Private Attributes

- static final String TAG = "BTConnection"

  Tag using for debugging.
- static final UUID uuid = UUID.fromString("00001101-0000-1000-8000-00805f9b34fb")

  UUID to allow Serial connection via BT.
- static final String NEW_LINE = "\r\n"

  New line string.

5.1.1 Detailed Description

Thread class for a new bluetooth connection to a device.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 BTConnection()

```
AndroidApp.BTConnection.BTConnection (
             BluetoothDevice btDevice ) throws IOException  [inline]
```

Constructor for BTConnection class.

Sets the BT device interface used for this class and attempts a connection.

Parameters

btDevice	- Device used for creating connection.

```
                          {
        this.btDevice = btDevice;

        /* Connect the device so ready to use run */
        connect();
    }
```

5.1.3 Member Function Documentation

5.1.3.1 setRXHandler()

```
void AndroidApp.BTConnection.setRXHandler (
             Handler newHandler )  [inline]
```

Setter function for RXHandler.

Parameters

newHandler	- The new Handler where RX'd data will be sent to.

```
                                              {
        RXHandler = newHandler;
    }
```

5.1.3.2 run()

```
void AndroidApp.BTConnection.run ( )  [inline]
```

Main run procedure for new Runnable thread created.

If connected procedure waits for data to be received. Parsing this received into lines and then splitting each line into a JSONObject. If a valid JSONObject is found it is then sends to the receive handler in a seperate thread (using messages).

```
                       {
        InputStream RXStream;

        /* Indicate that we are now running main thread */
        running = true;

        if (isConnected()) {
            /* Get our input stream for receiving bytes */
            try {
                RXStream = btSocket.getInputStream();
            } catch (IOException e) {
                Log.e(TAG, "Unable to get RXStream", e);
                running = false;
                return;
            }

            /*
             * While still connected and not signalled to stop we receive data
             * and then send it to the handler
             */
            String recvBuff = "";
            while (isRunning() && isConnected()) {
                try {
                    int bytesAvailable = RXStream.available();

                    if (bytesAvailable > 0) {
                        byte[] packetBytes = new byte[bytesAvailable];
                        int bytesRead = RXStream.read(packetBytes, 0, bytesAvailable);

                        recvBuff += new String(packetBytes);
                    }

                    if (RXHandler != null) {

                        if (recvBuff.indexOf(NEW_LINE) > 0) {

                            String jsonLine = recvBuff.substring(0, recvBuff.indexOf(NEW_LINE));

                            /*
                             * Having to send data to main thread using messages
                             * as we are multithreading.
                             * If we try and use a standard call to function
                             * will cause a crash.
                             */
                            Bundle dataBundle =  new Bundle();
                            dataBundle.putString("JSON", jsonLine);

                            Message message = RXHandler.obtainMessage();
                            message.setData(dataBundle);
                            message.sendToTarget();

                            recvBuff = recvBuff.replace(jsonLine + NEW_LINE, "");
                        }
                    }

                } catch (IOException e) {
                    Log.e(TAG, "Unable to read data", e);
                    running = false;
                    return;
                }
            }

        }

        /* Close bluetooth socket */
        try {
            this.close();
        } catch (IOException e) {
            /* Do nothing */
        }

        /* Null BT socket to show needs to reconnect */
        btSocket = null;
        running = false;
    }
```

5.1.3.3 isRunning()

`boolean AndroidApp.BTConnection.isRunning ( )  [inline]`

Function to check whether main connection thread is running.

Returns

boolean - Whether thread is running.

```
                               {
        return running;
    }
```

5.1.3.4 isConnected()

```
boolean AndroidApp.BTConnection.isConnected ( )  [inline]
```

Function to check whether BT connection is still valid.

Returns

boolean - Whether connection is still available.

```
                                {
        boolean result = false;

        if (btSocket != null) {
            if (btSocket.isConnected())
                result = true;
        }
        return result;
    }
```

5.1.3.5 connect()

```
void AndroidApp.BTConnection.connect ( ) throws IOException  [inline], [private]
```

Procedure to create a connection to logging device.

Creates a raw Serial socket via UUID and then attempts to connect. Exceptions thrown on failure.

```
                                               {

        /* Attempt to make connection to remote device, throw exception if not */
        try {
            btSocket = btDevice.createRfcommSocketToServiceRecord(
    uuid);
        } catch (IOException e) {
            Log.e(TAG, "Unable to create RFCOMM", e);
            throw e;
        }

        try {
            btSocket.connect();
        } catch (IOException e) {
            Log.e(TAG, "Unable to connect", e);

            /* Close our socket as unable to connect */
            try {
                this.close();
            } catch (IOException e2) {
                throw e2;
            }
            throw e;
        }
    }
```

5.1.4 Member Data Documentation

5.1.4.1 txHandler

```
final Handler AndroidApp.BTConnection.txHandler
```

Initial value:

```
= new Handler(Looper.getMainLooper()) {


        @Override
        public void handleMessage(Message msg) {


            if (isConnected() && isRunning()) {
                OutputStream TXStream;


                try {
                    TXStream = btSocket.getOutputStream();

                    String txString = (String)msg.obj;
                    TXStream.write(txString.getBytes());
                } catch (IOException e) {
                    Log.e(TAG, "Unable to use TXStream", e);
                    return;
                }
            }
        }
    }
```

Handler class for transmission of data.

Messages containing data to be transmitted are sent from main UI thread.

The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/BTConnection.java

5.2 AndroidApp.BTDeviceItem Class Reference

Class used for holding core UI information of a bluetooth devices.

Public Member Functions

- BTConnection getConnection ()

  Getter for the bluetooth connection of specified device.
- void setConnection (BTConnection newConn)

  Setter for setting the DeviceItem object's connection.
- BluetoothDevice getDevice ()

  Getter for BT device object (contains name, HWID etc.).
- String getStatus ()

  Getter for current status of BTDeviceItem.
- void setStatus (String newStatus)

  Setter for current status of BTDeviceItem.
- int getIconID ()

  Getter for icon ID to use in ListView.
- void setIconID (int newID)

  Setter for icon ID to use in ListView.
- BTDeviceItem (BluetoothDevice device, String status, int iconID)

  Constructor for BTDeviceItem class.

Private Attributes

- BTConnection connection = null

  Variable for BTConnection if device is already connected.
- int iconID

  ID of icon to use within the ListView.
- BluetoothDevice device

  Device object that holds info such as name, HWID etc.
- String status

  Status of the device, unpaired, paired, connected.

5.2.1 Detailed Description

Class used for holding core UI information of a bluetooth devices.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 BTDeviceItem()

```
AndroidApp.BTDeviceItem.BTDeviceItem (
              BluetoothDevice device,
              String status,
              int iconID )  [inline]
```

Constructor for BTDeviceItem class.

Called when new BluetoothDevice is found during discovery, so that it can be added to the device ListView.

Parameters

device	- BluetoothDevice containing HWID, name, etc.
status	- Current status of the discovered device.
iconID	- Icon ID to display within the ListView.

```
{
    this.device = device;
    this.status = status;
    this.iconID = iconID;
}
```

5.2.3 Member Function Documentation

5.2.3.1 getConnection()

```
BTConnection AndroidApp.BTDeviceItem.getConnection ( )  [inline]
```

Getter for the bluetooth connection of specified device.

Returns

BTConnection - Connection between app & logging device.

```
                                          {
            return connection;
        }
```

5.2.3.2 setConnection()

```
void AndroidApp.BTDeviceItem.setConnection (
                BTConnection newConn )  [inline]
```

Setter for setting the DeviceItem object's connection.

Parameters

newConn	- New connection between app & logging device.

```
                                                         {
            connection = newConn;
        }
```

5.2.3.3 getDevice()

```
BluetoothDevice AndroidApp.BTDeviceItem.getDevice ( )  [inline]
```

Getter for BT device object (contains name, HWID etc.).

Returns

BluetoothDevice - The bluetooth device object.

```
                                         {
            return device;
        }
```

5.2.3.4 getStatus()

```
String AndroidApp.BTDeviceItem.getStatus ( )  [inline]
```

Getter for current status of BTDeviceItem.

Returns

String - Current status: unpaired, paired or connected.

```
                         {
        return status;
    }
```

5.2.3.5 setStatus()

```
void AndroidApp.BTDeviceItem.setStatus (
             String newStatus )  [inline]
```

Setter for current status of BTDeviceItem.

Parameters

newStatus	- New string for status.

```
                                     {
        status = newStatus;
    }
```

5.2.3.6 getIconID()

```
int AndroidApp.BTDeviceItem.getIconID ( )  [inline]
```

Getter for icon ID to use in ListView.

Returns

int - Icon ID to use.

```
                         {
        return iconID;
    }
```

5.2.3.7 setIconID()

```
void AndroidApp.BTDeviceItem.setIconID (
             int newID )  [inline]
```

Setter for icon ID to use in ListView.

Parameters

newID	- New icon ID to use.

```
                                        {
        iconID = newID;
    }
```

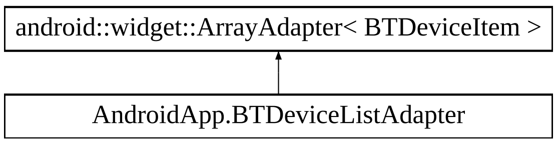
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/BTDeviceItem.java

5.3 AndroidApp.BTDeviceListAdapter Class Reference

Adapter class used for displaying bluetooth devices.

Inheritance diagram for AndroidApp.BTDeviceListAdapter:


android::widget::ArrayAdapter< BTDeviceItem >
AndroidApp.BTDeviceListAdapter

Classes

- class ViewHolder

  Class that holds all data displayed for each ListItem.

Public Member Functions

- BTDeviceListAdapter (Context cnt, int layoutResourceId, ArrayList$<$ BTDeviceItem $>$ data)

  Constructor for the ListView adapter.
- View getView (int position, View convertView, ViewGroup parent)

  Function for returning the view of each list item (BTDeviceItem).

Private Attributes

- int layoutResourceId

  Resource ID for current layout.
- Context context

  Context that the ListView is operating in.
- ArrayList$<$ BTDeviceItem $>$ data

  ArrayList of all bluetooth device items to display.

5.3.1 Detailed Description

Adapter class used for displaying bluetooth devices.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 BTDeviceListAdapter()

```
AndroidApp.BTDeviceListAdapter.BTDeviceListAdapter (
            Context cnt,
            int layoutResourceId,
            ArrayList< BTDeviceItem > data )  [inline]
```

Constructor for the ListView adapter.

Calls the constructor of the superclass as well as setting other relevant information needed.

Parameters

cnt	- Context of the adapter to be operating in.
layout↩ ResourceId	- Resource ID for current layout.
data	- ArrayList of devices to display in ListView.

```
                                                                                                {
        super(cnt, layoutResourceId, data);
        this.context = cnt;
        this.layoutResourceId = layoutResourceId;
        this.data = data;
    }
```

5.3.3 Member Function Documentation

5.3.3.1 getView()

```
View AndroidApp.BTDeviceListAdapter.getView (
            int position,
            View convertView,
            ViewGroup parent )  [inline]
```

Function for returning the view of each list item (BTDeviceItem).

If a view for selected item has not been created inflater initialises it. A holder is then used to hold all the information that will be displayed on the UI to the user.

Parameters

position	- Index of item in array to use/reference to.
convertView	- View to be used for specified item.
parent	- Object where the created view will be placed on.

Returns

View - The result view of item with updated/current information.

References AndroidApp.BTDeviceItem.getDevice(), AndroidApp.BTDeviceItem.getIconID(), and AndroidApp.BT↩ DeviceItem.getStatus().

```
                                                                      {

        ViewHolder holder;

        if (convertView == null)
        {
            /* Create new view via inflater as it does not exist. */
            LayoutInflater inflater = (LayoutInflater)context.getSystemService(Context.
    LAYOUT_INFLATER_SERVICE);
            convertView = inflater.inflate(layoutResourceId, parent, false);

            /* Create holder that will contain information to display. */
            holder = new ViewHolder();
            holder.imageStatus = (ImageView)convertView.findViewById(R.id.imageListStatus);
            holder.name = (TextView)convertView.findViewById(R.id.textListName);
            holder.address = (TextView)convertView.findViewById(R.id.textListAddress);
            holder.status = (TextView)convertView.findViewById(R.id.textListStatus);
            convertView.setTag(holder);
        }
        else
        {
            /* Get current holder to use instead of creating new one. */
            holder = (ViewHolder)convertView.getTag();
        }

        /* Get BTDeviceItem for specified item and update holder info. */
        BTDeviceItem btItem = getItem(position);
        holder.imageStatus.setImageResource(btItem.getIconID());
        holder.name.setText(btItem.getDevice().getName());
        holder.address.setText(btItem.getDevice().getAddress());
        holder.status.setText(btItem.getStatus());

        return convertView;
    }
```

The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/BTDeviceListAdapter.java

5.4 AndroidApp.BTDeviceListAdapter.ViewHolder Class Reference

Class that holds all data displayed for each ListItem.

5.4.1 Detailed Description

Class that holds all data displayed for each ListItem.

The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/BTDeviceListAdapter.java

5.5 AndroidApp.DataItem< T > Class Template Reference

Class used for holding and displaying a piece of data within the statistic ListView UI.

Public Member Functions

- DataItem (String name, boolean avgMinMax)

  Constructor for creation of a DataItem.
- DataItem (String name, boolean avgMinMax, T value)

  Constructor for creation of a DataItem.
- String getName ()

  Getter for name of data item.
- boolean getEnabledAvgMinMax ()

  Getter for whether additional functionality enabled.
- T getCurrent ()

  Getter for current reading value.
- Double getAverage ()

  Getter for average of readings.
- T getMinimum ()

  Getter for minimum of readings.
- T getMaximum ()

  Getter for maximum of readings.
- void setCurrent (T value)

  Setter for current reading value.

Private Member Functions

- Double add (Number a, Number b)

  Function to allow addition of numbers with variable types.
- Double divide (Number numerator, Number denominator)

  Function to allow division of numbers with variable types.
- boolean greaterThan (Number a, Number b)

  Function to chcek whether A is greater than B.
- boolean lessThan (Number a, Number b)

  Function to chcek whether A is less than B.

Private Attributes

- String name

  The name of the statistic.
- boolean enableAvgMinMax

  Whether averaging, min & max values should be calculated.
- T current = null

  Current reading value.
- Double average = 0.0

  Average reading value.
- Double averageSum = 0.0

  Sum of all readings, used for averaging.
- int averageCount = 0

  Number of readings, used for averaging.
- T minimum = null

  Minimum reading value.
- T maximum = null

  Maximum reading value.

5.5.1 Detailed Description

Class used for holding and displaying a piece of data within the statistic ListView UI.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 DataItem() [1/2]

```
AndroidApp.DataItem< T >.DataItem (
             String name,
             boolean avgMinMax )  [inline]
```

Constructor for creation of a DataItem.

Sets up the name of the data item as well as Whether averaging, minimum and maximum readings will be used

Parameters

name	- Name of the data item.
avgMinMax	- Whether additive functionality shall be available.

```
                                                     {
        this.name = name;
        this.enableAvgMinMax = avgMinMax;
    }
```

5.5.2.2 DataItem() [2/2]

```
AndroidApp.DataItem< T >.DataItem (
             String name,
             boolean avgMinMax,
             T value )  [inline]
```

Constructor for creation of a DataItem.

Similar to other constructor however allows setting of an initial value.

Parameters

name	- Name of the data item.
avgMinMax	- Whether additive functionality shall be available.
value	- Initial reading value.

```
                                                              {
```

```
        this.name = name;
        this.enableAvgMinMax = avgMinMax;
        this.current = value;

        if ((avgMinMax) && (current instanceof Number)) {
            this.average = (Double)value;
            this.averageSum = (Double)value;
            this.averageCount++;

            this.minimum = value;
            this.maximum = value;
        }
    }
```

5.5.3 Member Function Documentation

5.5.3.1 getName()

String AndroidApp.DataItem< T >.getName ( ) [inline]

Getter for name of data item.

Returns

String - DataItem name.

```
                            {
        return name;
    }
```

5.5.3.2 getEnabledAvgMinMax()

boolean AndroidApp.DataItem< T >.getEnabledAvgMinMax ( ) [inline]

Getter for whether additional functionality enabled.

Returns

boolean - Averaging, Minimum & Maximum enabled.

```
                                         {
        return enableAvgMinMax;
    }
```

5.5.3.3 getCurrent()

`T AndroidApp.DataItem< T >.getCurrent ( ) [inline]`

Getter for current reading value.

Returns

T - Current reading value.

```
                            {
        return current;
    }
```

5.5.3.4 getAverage()

`Double AndroidApp.DataItem< T >.getAverage ( ) [inline]`

Getter for average of readings.

Returns

Double - Average of all readings.

```
                                 {
        return average;
    }
```

5.5.3.5 getMinimum()

`T AndroidApp.DataItem< T >.getMinimum ( ) [inline]`

Getter for minimum of readings.

Returns

T - Minimum value.

```
                            {
        return minimum;
    }
```

5.5.3.6 getMaximum()

```
T AndroidApp.DataItem< T >.getMaximum ( )  [inline]
```

Getter for maximum of readings.

Returns

T - Maximum value.

```
                        {
        return maximum;
    }
```

5.5.3.7 setCurrent()

```
void AndroidApp.DataItem< T >.setCurrent (
            T value )  [inline]
```

Setter for current reading value.

If additive functionality enabled and the reading is of types number then we go ahead and update our min, max & average values as well will the passed in new reading.

Parameters

T	- New reading.

```
                                {
        this.current = value;

        if ((enableAvgMinMax) && (current instanceof Number)) {

            /* Sets the average */
            averageCount++;
            averageSum = add(averageSum, (Number)value);
            average = divide(averageSum, averageCount);

            /* Sets the new minimum and maximums if true */
            if ((minimum == null) || lessThan((Number)current, (Number)
    minimum)) {
                minimum = current;
            }
            if ((maximum == null) || greaterThan((Number)current, (Number)
    maximum)) {
                maximum = current;
            }
        }
    }
```

5.5.3.8 add()

```
Double AndroidApp.DataItem< T >.add (
            Number a,
            Number b )  [inline], [private]
```

Function to allow addition of numbers with variable types.

Parameters

a	- First operand.
b	- Second operand.

Returns

Double - Sum.

```
                                                    {
          return new Double(a.doubleValue() + b.doubleValue());
     }
```

5.5.3.9 divide()

```
Double AndroidApp.DataItem< T >.divide (
             Number numerator,
             Number denominator )  [inline], [private]
```

Function to allow division of numbers with variable types.

Parameters

numerator	- Numerator of divisior.
denominator	- Denominator of divisor.

Returns

Double - Result of division.

```
                                                                            {
          return new Double(numerator.doubleValue() / denominator.doubleValue());
     }
```

5.5.3.10 greaterThan()

```
boolean AndroidApp.DataItem< T >.greaterThan (
             Number a,
             Number b )  [inline], [private]
```

Function to chcek whether A is greater than B.

Parameters

a	- First operand.
b	- Second operand.

Returns

boolean - Whether A is greater than B.

```
                                                   {
        return a.doubleValue() > b.doubleValue();
    }
```

5.5.3.11 lessThan()

```
boolean AndroidApp.DataItem< T >.lessThan (
              Number a,
              Number b )  [inline], [private]
```

Function to chcek whether A is less than B.

Parameters

a	- First operand.
b	- Second operand.

Returns

boolean - Whether A is less than B.

```
                                                {
        return a.doubleValue() < b.doubleValue();
    }
```

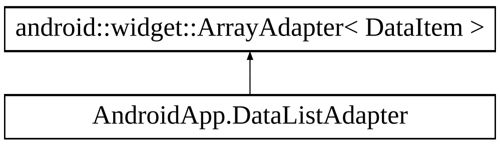
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/DataItem.java

5.6 AndroidApp.DataListAdapter Class Reference

Adapter class used for displaying statistics.

Inheritance diagram for AndroidApp.DataListAdapter:


android::widget::ArrayAdapter< DataItem >
AndroidApp.DataListAdapter

Classes

- class ViewHolder

  Class that holds all data displayed for each ListItem.

Public Member Functions

- DataListAdapter (Context cnt, int layoutResourceId, ArrayList< DataItem > data)

  Constructor for the ListView adapter.
- View getView (int position, View convertView, ViewGroup parent)

  Function for returning the view of each list item (DataItem).

Private Attributes

- Context context

  Context that the ListView is operating in.
- int layoutResourceId

  Resource ID for current layout.
- ArrayList< DataItem > data

  ArrayList of all statistic items to display.

5.6.1 Detailed Description

Adapter class used for displaying statistics.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 DataListAdapter()

```
AndroidApp.DataListAdapter.DataListAdapter (
              Context cnt,
              int layoutResourceId,
              ArrayList< DataItem > data )  [inline]
```

Constructor for the ListView adapter.

Calls the constructor of the superclass as well as setting other relevant information needed.

Parameters

cnt	- Context of the adapter to be operating in.
layout↵ ResourceId	- Resource ID for current layout.
data	- ArrayList of statistics to display in ListView.

```
                                                                                                   {
          super(cnt, layoutResourceId, data);

          this.context = cnt;
          this.layoutResourceId = layoutResourceId;
          this.data = data;
      }
```

5.6.3 Member Function Documentation

5.6.3.1 getView()

```
View AndroidApp.DataListAdapter.getView (
            int position,
            View convertView,
            ViewGroup parent )  [inline]
```

Function for returning the view of each list item (DataItem).

If a view for selected item has not been created inflater initialises it. A holder is then used to hold all the information that will be displayed on the UI to the user.

Parameters

position	- Index of item in array to use/reference to.
convertView	- View to be used for specified item.
parent	- Object where the created view will be placed on.

Returns

View - The result view of item with updated/current information.

References AndroidApp.DataItem< T >.getAverage(), AndroidApp.DataItem< T >.getCurrent(), AndroidApp.↩ DataItem< T >.getEnabledAvgMinMax(), AndroidApp.DataItem< T >.getMaximum(), AndroidApp.DataItem< T >.getMinimum(), and AndroidApp.DataItem< T >.getName().

```
                                                                            {

        ViewHolder holder;

        if (convertView == null)
        {
            /* If view does not already exist. */
            LayoutInflater inflater = (LayoutInflater)context.getSystemService(Context.
    LAYOUT_INFLATER_SERVICE);
            convertView = inflater.inflate(layoutResourceId, parent, false);

            holder = new ViewHolder();
            holder.name = (TextView)convertView.findViewById(R.id.datalist_name);
            holder.current = (TextView)convertView.findViewById(R.id.datalist_current);
            holder.average = (TextView)convertView.findViewById(R.id.datalist_average);
            holder.minimum = (TextView)convertView.findViewById(R.id.datalist_minimum);
            holder.maximum = (TextView)convertView.findViewById(R.id.datalist_maximum);
            convertView.setTag(holder);
        }
        else
        {
            /* If view already exists. */
            holder = (ViewHolder)convertView.getTag();
        }

        DataItem dataItem = getItem(position);

        /* Set our holder with current data of item */
        holder.name.setText(dataItem.getName());

        Object current = dataItem.getCurrent();
        if (current != null) {
            DecimalFormat df = new DecimalFormat("#.####");
            df.setRoundingMode(RoundingMode.CEILING);

            /* To aid aesthetics rounding is used. */
            if (current instanceof Double) {
                holder.current.setText(df.format(current));
            } else {
                holder.current.setText(current.toString());
            }

            /*
             * Displays added functionality if available.
             * Not all statistics need it, for example averaging of LAT/LNG.
             */
            if (dataItem.getEnabledAvgMinMax()) {
                holder.average.setText(df.format(dataItem.getAverage()));
                holder.minimum.setText(df.format(dataItem.getMinimum()));
                holder.maximum.setText(df.format(dataItem.getMaximum()));
            } else {
                holder.average.setText("N/A");
                holder.minimum.setText("N/A");
                holder.maximum.setText("N/A");
            }
        }

        return convertView;
    }
```

The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/DataListAdapter.java

5.7 AndroidApp.DataListAdapter.ViewHolder Class Reference

Class that holds all data displayed for each ListItem.

5.7.1 Detailed Description

Class that holds all data displayed for each ListItem.

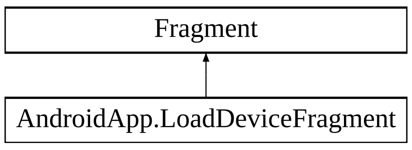
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/DataListAdapter.java

5.8 AndroidApp.LoadDeviceFragment Class Reference

UI Class for loading saved trips from device.

Inheritance diagram for AndroidApp.LoadDeviceFragment:



Classes

- class TripItemListener

  Listener used to identify when a trip has been pressed.

Public Member Functions

- LoadDeviceFragment ()

  Constructor for UI fragment.
- View onCreateView (LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)

  Function called when fragment is shown on UI.
- void setBTConnection (BTConnection btConnection)

  Setter for current BT connection.

Public Attributes

- final Handler RXHandler

  Handler used for receiving trip names.

Private Member Functions

- final void addTrip (JSONObject jsonData)

  Adds a trip to the ListView specifying name and filesize.

Private Attributes

- BTConnection btConnection = null

  Current connectected logging device (via bluetooth).
- ArrayList< TripItem > tripList

  List of all trips saved on the logging device.
- ArrayAdapter< TripItem > lvAdapter

  Array adapter for displaying trips in ListView.

Static Private Attributes

- static final String NEW_LINE = "\r\n"

  New line string.
- static final String LOAD_TRIP_CHAR = "3"

  Command string to be sent to device to load a specific trip.

5.8.1 Detailed Description

UI Class for loading saved trips from device.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 LoadDeviceFragment()

```
AndroidApp.LoadDeviceFragment.LoadDeviceFragment ( )  [inline]
```

Constructor for UI fragment.

Creates a new arraylist of trips that is empty and ready to be filled from the logging device.

```
                            {
        tripList = new ArrayList<TripItem>();
    }
```

5.8.3 Member Function Documentation

5.8.3.1 onCreateView()

```
View AndroidApp.LoadDeviceFragment.onCreateView (
             LayoutInflater inflater,
             ViewGroup container,
             Bundle savedInstanceState )  [inline]
```

Function called when fragment is shown on UI.

Sets up the ListView on the screen using our custom ArrayAdapter specificed.

Parameters

inflater	- Inflater used to load fragment on UI.
container	- Container where fragment will be shown.
savedInstanceState	- Information holding past state.

Returns

View - Modified view to display on the UI.

```
                                                                                          {
        View myView = inflater.inflate(R.layout.loaddevice_layout, container, false);

        /* Get our ListView via ID, set headers and create our ArrayAdapter for it */
        ListView lvTripList = (ListView)myView.findViewById(R.id.loaddevice_triplist);
        lvTripList.setOnItemClickListener(new TripItemListener());

        ViewGroup headerView = (ViewGroup)inflater.inflate(R.layout.trip_list_header, lvTripList, false);
        lvTripList.addHeaderView(headerView);

        lvAdapter = new TripListAdapter(getActivity(), R.layout.trip_list_item,
   tripList);
```

```
        lvTripList.setAdapter(lvAdapter);

        tripList.clear();
        lvAdapter.notifyDataSetChanged();

        return myView;
    }
```

5.8.3.2 setBTConnection()

```
void AndroidApp.LoadDeviceFragment.setBTConnection (
            BTConnection btConnection )  [inline]
```

Setter for current BT connection.

Set from main UI activity, allows cross tab communication with the logging device.

Parameters

btConnection	- Logging device bluetooth connection.

```
                                                    {
        this.btConnection = btConnection;
    }
```

5.8.3.3 addTrip()

```
final void AndroidApp.LoadDeviceFragment.addTrip (
            JSONObject jsonData )  [inline], [private]
```

Adds a trip to the ListView specifying name and filesize.

Parameters

jsonData	- JSON object holding trip name and size.

```
                                                    {
        try {

            /* Get name and size from json object */
            String tripName = jsonData.getString("name");
            int fileSize = jsonData.getInt("size");

            /* Add new trip to our list & notify list view */
            TripItem newTrip = new TripItem(tripName, fileSize);
            tripList.add(newTrip);
            lvAdapter.notifyDataSetChanged();

        } catch (JSONException e) {
            /* Do nothing */
        }
    }
```

5.8.4 Member Data Documentation

5.8.4.1 RXHandler

```
final Handler AndroidApp.LoadDeviceFragment.RXHandler
```

Initial value:

```
= new Handler(Looper.getMainLooper()) {

        @Override
        public void handleMessage(Message msg) {

            Bundle msgData = msg.getData();
            String jsonString = msgData.getString("JSON");

            if (jsonString != null) {


                try {
                    JSONObject tmpJSON = new JSONObject(jsonString);
                    addTrip(tmpJSON);

                } catch (JSONException e) {

                }
            }
        }
    }
```

Handler used for receiving trip names.

Receives trip information from the bluetooth connection thread. Handler has to be used as system is multithreaded.

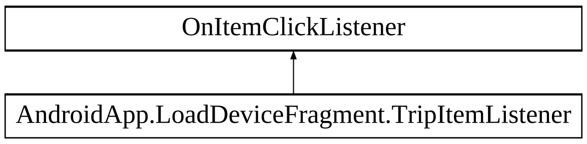
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/LoadDeviceFragment.java

5.9 AndroidApp.LoadDeviceFragment.TripItemListener Class Reference

Listener used to identify when a trip has been pressed.

Inheritance diagram for AndroidApp.LoadDeviceFragment.TripItemListener:



Public Member Functions

- void onItemClick (AdapterView<?> parent, View view, int position, long id)

    Loads a trip the user has specified.

5.9.1 Detailed Description

Listener used to identify when a trip has been pressed.

5.9.2 Member Function Documentation

5.9.2.1 onItemClick()

```
void AndroidApp.LoadDeviceFragment.TripItemListener.onItemClick (
              AdapterView<?> parent,
              View view,
              int position,
              long id )  [inline]
```

Loads a trip the user has specified.

User has selected a trip via the ListView, method switches to the statistic fragment and sends a message to logging device to load the specified trip (via name).

References AndroidApp.TripItem.getTripName(), AndroidApp.BTConnection.isConnected(), AndroidApp.↩RealtimeFragment.RXHandler, AndroidApp.BTConnection.setRXHandler(), and AndroidApp.BTConnection.tx↩Handler.

```
                                                                                        {

            if (btConnection != null && btConnection.
      isConnected()) {
                TripItem tripItem = (TripItem) parent.getItemAtPosition(position);

                /*
                 * Create a new statistics fragment.
                 * This will receive the stored data from the logging device.
                 */
                RealtimeFragment statFragment = new RealtimeFragment();
                btConnection.setRXHandler(statFragment.RXHandler);

                /* Transmit over the name of the trip we want to load */
                Message message = new Message();
                message.obj = (String) LOAD_TRIP_CHAR + tripItem.getTripName();
                message.setTarget(btConnection.txHandler);
                message.sendToTarget();

                FragmentManager fragmentManager = getFragmentManager();
                fragmentManager.beginTransaction()
                        .replace(R.id.content_frame, statFragment)
                        .commit();
            }
        }
```

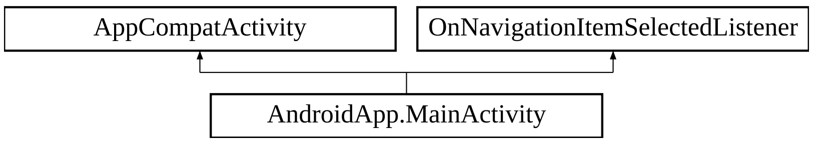
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/LoadDeviceFragment.java

5.10 AndroidApp.MainActivity Class Reference

Main activity class for fragment navigation.

Inheritance diagram for AndroidApp.MainActivity:



Public Member Functions

- void onBackPressed ()

  Responsible for closing navigation drawer when back button pressed.
- boolean onNavigationItemSelected (MenuItem item)

  Changes active fragment when a tab has been pressed.

Protected Member Functions

- void onCreate (Bundle savedInstanceState)

  Function called when main activity is loaded.

Static Private Attributes

- static final String REALTIME_CHAR = "1"

  Command for switching to realtime logging.
- static final String LIST_SAVED_CHAR = "2"

  Command for loading all saved trip details.
- static RealtimeFragment rtFragment = null

  UI fragment for realtime statistic display.
- static LoadDeviceFragment ldFragment = null

  UI fragment for loading previous trips.
- static PairDeviceFragment pdFragment = null

  UI fragment for pairing to a logging device.

5.10.1 Detailed Description

Main activity class for fragment navigation.

5.10.2 Member Function Documentation

5.10.2.1 onCreate()

```
void AndroidApp.MainActivity.onCreate (
             Bundle savedInstanceState )  [inline], [protected]
```

Function called when main activity is loaded.

Procedure is called when application is first started, sets up UI and creates relevant fragments.

Parameters

savedInstanceState	- Information holding last previous state.

```
                                                    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
        ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
                this, drawer, toolbar, R.string.navigation_drawer_open, R.string.navigation_drawer_close);
        drawer.setDrawerListener(toggle);
        toggle.syncState();

        NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
        navigationView.setNavigationItemSelectedListener(this);

        /* Create our fragments for different sections of UI */
        rtFragment = new RealtimeFragment();
        ldFragment = new LoadDeviceFragment();
        pdFragment = new PairDeviceFragment();
    }
```

5.10.2.2 onNavigationItemSelected()

```
boolean AndroidApp.MainActivity.onNavigationItemSelected (
            MenuItem item )  [inline]
```

Changes active fragment when a tab has been pressed.

Responsible for changing to the new chosen fragment on the UI. Opening of realtime and loaddevice fragments not possible when not connected to the logging device.

Method also responsible for change system state machine on the logging device, this is done by transmitting command code.

Parameters

item	- New selected fragment/tab to display.

References AndroidApp.PairDeviceFragment.getBTConnection(), AndroidApp.BTConnection.isConnected(), AndroidApp.LoadDeviceFragment.RXHandler, AndroidApp.RealtimeFragment.RXHandler, AndroidApp.LoadDeviceFragment.setBTConnection(), AndroidApp.BTConnection.setRXHandler(), and AndroidApp.BTConnection.txHandler.

```
                                                            {

        Fragment activeFragment = null;

        /* Handle navigation view clicks here */
        FragmentManager fragmentManager = getFragmentManager();
        int id = item.getItemId();

        switch (id) {
            case R.id.nav_realtime: {
                /* Get our bluetooth connection from pairing fragment */
                BTConnection btConn = pdFragment.getBTConnection();

                if (btConn != null && btConn.isConnected()) {
                    /* We set our RX handler and also send our command to indicate mode change */
                    btConn.setRXHandler(rtFragment.
    RXHandler);
                    Message message = new Message();
                    message.obj = (String) REALTIME_CHAR;
                    message.setTarget(btConn.txHandler);
                    message.sendToTarget();

                    /* Change to our new active fragment */
                    activeFragment = rtFragment;
                } else {
                    /* Indicate that we are not connected to device */
                    View rootView = findViewById(R.id.content_main);
                    Snackbar.make(rootView, "Please connect to a device first.", Snackbar.LENGTH_LONG)
                            .setAction("Action", null).show();
                }
                break;
            }

            case R.id.nav_loaddevice: {
                /* Get our bluetooth connection from pairing fragment */
                BTConnection btConn = pdFragment.getBTConnection();

                if (btConn != null && btConn.isConnected()) {
                    /* We set our RX handler and also send our command to indicate mode change */
                    ldFragment.setBTConnection(btConn);

                    btConn.setRXHandler(ldFragment.RXHandler);
                    Message message = new Message();
                    message.obj = (String) LIST_SAVED_CHAR;
                    message.setTarget(btConn.txHandler);
                    message.sendToTarget();

                    /* Change to our new active fragment */
                    activeFragment = ldFragment;
                } else {
                    /* Indicate that we are not connected to device */
                    View rootView = findViewById(R.id.content_main);
                    Snackbar.make(rootView, "Please connect to a device first.", Snackbar.LENGTH_LONG)
                            .setAction("Action", null).show();
                }
                break;
            }

            case R.id.nav_pairdevice: {
                activeFragment = pdFragment;
            }

        }

        if (activeFragment != null) {
            /* Replaces content frame with newly selected one */
            fragmentManager.beginTransaction()
                    .replace(R.id.content_frame, activeFragment)
                    .commit();
        }

        DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
        drawer.closeDrawer(GravityCompat.START);
        return (activeFragment != null);
    }
```

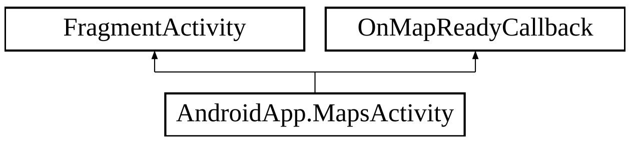
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/MainActivity.java

5.11 AndroidApp.MapsActivity Class Reference

Maps activity class for displaying map data.

Inheritance diagram for AndroidApp.MapsActivity:



Classes

- class StatisticWindowAdapter

    Adapter used for displaying statistics at a certain marker that user has clicked on.

Public Member Functions

- void onMapReady (GoogleMap googleMap)

    Manipulates the map once available.

Protected Member Functions

- void onCreate (Bundle savedInstanceState)

    Fills our maps array with points to plot on the map.

Private Member Functions

- boolean getJSONObjects ()

    Gets point data and convert to array of JSON objects.
- JSONObject findJSONByLatLng (LatLng position)

    Finds JSONObject from ArrayList via LAT/LNG coordinates.
- float calcDistance (LatLng start, LatLng end)

    Calculates the absolute distance between two points.

Private Attributes

- GoogleMap mMap

    Google maps object for plotting.
- ArrayList< JSONObject > jsonList = new ArrayList<JSONObject>()

    ArrayList holding all trip data.

5.11.1 Detailed Description

Maps activity class for displaying map data.

5.11.2 Member Function Documentation

5.11.2.1 onCreate()

```
void AndroidApp.MapsActivity.onCreate (
            Bundle savedInstanceState )  [inline], [protected]
```

Fills our maps array with points to plot on the map.

Called when maps activity is first started. Responsible for making sure we have points to plot.

Parameters

savedInstanceState	- Information holding last previous state.

```
                                                    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // Obtain the SupportMapFragment and get notified when the map is ready to be used.
        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
                .findFragmentById(R.id.map);

        getJSONObjects();

        mapFragment.getMapAsync(this);
    }
```

5.11.2.2 getJSONObjects()

`boolean AndroidApp.MapsActivity.getJSONObjects ( )  [inline], [private]`

Gets point data and convert to array of JSON objects.

Gets an arraylist of strings passed via a bundle to this activity. These strings are there converted back to JSON objects which will be used for plotting. The reason for not passing straight JSON objects is because they are not serializable and passable between activities.

Returns

boolean - Whether all objects were able to be created.

```
    {
        boolean result = true;

        /*
         * Get our serialized arrayList of jsonStrings
         * then convert them back to jsonObjects
         */
        ArrayList<String> jsonStrings = (ArrayList<String>)getIntent().getSerializableExtra("JSONList");
        for (int i = 0; i < jsonStrings.size(); i++)
        {
            try
            {
                JSONObject jsonObject = new JSONObject(jsonStrings.get(i));
                jsonList.add(jsonObject);
            }
            catch (JSONException e)
            {
                result = false;
            }
        }

         return result;
     }
```

5.11.2.3 findJSONByLatLng()

```
JSONObject AndroidApp.MapsActivity.findJSONByLatLng (
               LatLng position )  [inline], [private]
```

Finds JSONObject from ArrayList via LAT/LNG coordinates.

Parameters

position	- Latitude and Longitude position.

Returns

JSONObject - The found JSON object.

References gpsJSON.

```
                                                        {
        JSONObject result = null;

        for (int i = 0; i < jsonList.size(); i++) {
            JSONObject tmpJSON = jsonList.get(i);

            try {
                JSONObject gpsJSON = tmpJSON.getJSONObject("gps");

                Double latitude = gpsJSON.getDouble("lat");
                Double longitude = gpsJSON.getDouble("lng");

                /* Check to see if latitude and logitudes match */
                if ((latitude == position.latitude) && (longitude == position.longitude)) {
                    result = tmpJSON;
                    break;
                }

            } catch (JSONException e) {
                /* Do nothing */
            }
        }

        return result;
    }
```

5.11.2.4 calcDistance()

```
float AndroidApp.MapsActivity.calcDistance (
              LatLng start,
              LatLng end )  [inline], [private]
```

Calculates the absolute distance between two points.

Distance is as the crow flys and not via streets etc.

Parameters

start	- Start position.
end	- End position.

Returns

flaot - Distance between points in metres.

```
    {
        float[] results = new float[1];

        Location.distanceBetween(start.latitude, start.longitude, end.latitude, end.longitude, results);
        return results[0];
    }
```

5.11.2.5 onMapReady()

```
void AndroidApp.MapsActivity.onMapReady (
            GoogleMap googleMap )  [inline]
```

Manipulates the map once available.

This callback is triggered when the map is ready to be used. This is where we can add markers or lines.

If Google Play services is not installed on the device, the user will be prompted to install it inside the SupportMap↩ Fragment. This method will only be triggered once the user has installed Google Play services and returned to the app.

Parameters

googleMap	- Our map object ready to manipulate.

References gpsJSON.

```
                                                {
        mMap = googleMap;

        mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);

        /* Set our info window adapter class that is shown when marker clicked */
        mMap.setInfoWindowAdapter(new StatisticWindowAdapter());

        /* If we have no data don't bother plotting points */
        if (jsonList.size() != 0)
        {
            /* lineOpts will store our route */
            PolylineOptions lineOpts = new PolylineOptions();
            lineOpts.color(Color.parseColor( "#CC0000FF"));
            lineOpts.width(5);
            lineOpts.visible(true);

            try
            {
                LatLng lastMarker = null;

                /* Plot every point in the our JSONObject array */
                for (int i = 0; i < jsonList.size(); i++)
                {
                    JSONObject rootJSON = jsonList.get(i);
                    JSONObject gpsJSON = rootJSON.getJSONObject("gps");

                    Double lat = gpsJSON.getDouble("lat");
                    Double lng = gpsJSON.getDouble("lng");
                    LatLng location = new LatLng(lat, lng);

                    /* Add this location to our trip line */
                    lineOpts.add(location);

                    /*
                     * Check if distance between this point and
                     * last marker is greater than 5m otherwise don't add marker.
                     * Adding markers every 5 metres prevents the map being spammed with
                     * thousands of readings.
                     */
                    if ((lastMarker == null) || (calcDistance(location, lastMarker) > 5))
                    {
                        /* Only add a marker if the gps data is valid */
                        if (gpsJSON.getBoolean("gps_valid") == true) {
                            MarkerOptions markerOptions = new MarkerOptions();
                            markerOptions.position(location);
                            markerOptions.title("Reading: " + i);

                            mMap.addMarker(markerOptions);

                            lastMarker = location;

                            /* Changes camera to point to newest marker */
                            mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(location, 12));
                        }
                    }

                }
                mMap.addPolyline(lineOpts);
            }
            catch (JSONException e)
            {
                /* Do nothing */
            }
        }
    }
```

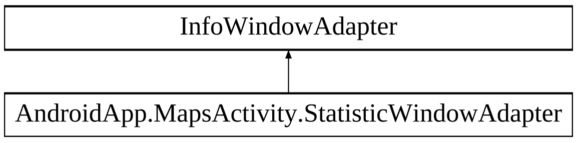
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/MapsActivity.java

5.12 AndroidApp.MapsActivity.StatisticWindowAdapter Class Reference

Adapter used for displaying statistics at a certain marker that user has clicked on.

Inheritance diagram for AndroidApp.MapsActivity.StatisticWindowAdapter:



Public Member Functions

- View getInfoWindow (Marker marker)

  We don't want to use default information window.
- View getInfoContents (Marker marker)

  Displays statistics at a marker that the user has clicked on.

5.12.1 Detailed Description

Adapter used for displaying statistics at a certain marker that user has clicked on.

5.12.2 Member Function Documentation

5.12.2.1 getInfoContents()

```
View AndroidApp.MapsActivity.StatisticWindowAdapter.getInfoContents (
               Marker marker )  [inline]
```

Displays statistics at a marker that the user has clicked on.

Parameters

marker	- The marker the user has clicked on.

Returns

View - Updated view showing information.

References gpsJSON, orientJSON, and timeJSON.

```
                                                    {

                View v =  getLayoutInflater().inflate(R.layout.map_marker_info, null);

                /* Get latitude and longitude from marker */
                LatLng latlng = marker.getPosition();

                /* Find the JSONObject relating to this location */
                JSONObject rootJSON = findJSONByLatLng(latlng);
                if (rootJSON != null) {
                    try {
                        JSONObject gpsJSON = rootJSON.getJSONObject("gps");
                        JSONObject orientJSON = rootJSON.getJSONObject("orientation");
                        JSONObject timeJSON = rootJSON.getJSONObject("time");

                        /* Set latitude and longitude in info window */
                        TextView tvLatLng = (TextView)v.findViewById(R.id.map_latlng);
                        tvLatLng.setText("Lat/Lng: " + Double.toString(latlng.latitude) + "/"
                                + Double.toString(latlng.longitude));

                        /* Set time */
                        TextView tvTime = (TextView)v.findViewById(R.id.map_time);
                        Calendar cal = Calendar.getInstance();
                        cal.clear();
                        cal.set(Calendar.YEAR, timeJSON.getInt("year"));
                        cal.set(Calendar.MONTH, timeJSON.getInt("month"));
                        cal.set(Calendar.DATE, timeJSON.getInt("day"));

                        cal.set(Calendar.HOUR, timeJSON.getInt("hour"));
                        cal.set(Calendar.MINUTE, timeJSON.getInt("minute"));
                        cal.set(Calendar.SECOND, timeJSON.getInt("second"));
                        cal.set(Calendar.MILLISECOND, timeJSON.getInt("centiseconds") * 10);

                        /* Create format for date and times then add to view */
                        DateFormat dateFormat = new SimpleDateFormat("dd/MM/yy HH:mm:ss.SS");
                        tvTime.setText("Time: " + dateFormat.format(cal.getTime()));

                        /* Velocity & Altitude */
                        TextView tvVelocity = (TextView)v.findViewById(R.id.map_velocity);
                        tvVelocity.setText("Velocity: " + gpsJSON.getDouble("vel_mph") + "mph");

                        TextView tvAltitude = (TextView)v.findViewById(R.id.map_altitude);
                        tvAltitude.setText("Altitude: " + gpsJSON.getDouble("alt_ft") + "ft");

                        /* Orientation */
                        TextView tvPitch = (TextView)v.findViewById(R.id.map_pitch);
                        tvPitch.setText("Pitch Angle: " + orientJSON.getDouble("pitch") + "\u00b0");

                        TextView tvRoll = (TextView)v.findViewById(R.id.map_roll);
                        tvRoll.setText("Roll/Lean Angle: " + orientJSON.getDouble("roll") + "\u00b0");

                    } catch (JSONException e) {
                        marker.hideInfoWindow();
                    }
                } else {
                    /* If unable to find relating we hide the info window */
                    marker.hideInfoWindow();
                }

                return v;
            }
```

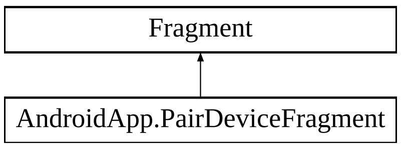
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/MapsActivity.java

5.13 AndroidApp.PairDeviceFragment Class Reference

UI Class for discovering, pairing and connecting to the logging device.

Inheritance diagram for AndroidApp.PairDeviceFragment:



Classes

- class DeviceItemListener

  Listener for when a ListView item is pressed (to connect).
- class DiscoverButtonListener

  Listener for when discovery button is pressed.
- class DiscoverReceiver

  Receiver for when a new device is discovered.

Public Member Functions

- PairDeviceFragment ()

  Constructor for UI fragment.
- View onCreateView (LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)

  Function called when fragment is shown on UI.
- BTConnection getBTConnection ()

  Getter for getting current connected device.

Private Member Functions

- void getNeededPrivileges ()

  Prompts user for needed permissions of this application.

Private Attributes

- boolean firstRun = true

  Check variable used to stop ListView from being re-populated.
- ToggleButton btnScan

  Scan button, used for toggling discovery.
- BluetoothAdapter btAdapter = null

  Mobile's bluetooth adapter.
- ArrayList< BTDeviceItem > btDeviceList

  List of all devices, unpaired, paired & connected.
- ArrayList< BTDeviceItem > btPairedList

  List of only paired devices.
- ArrayAdapter< BTDeviceItem > lvAdapter

  UI adapter for ListView that displays bluetooth devices.
- BTDeviceItem btConnectedDevice = null

  Applications connected logging device.
- DiscoverReceiver btReceiver = null

  Receiver class for when new device discovered.

Static Private Attributes

- static final int REQUEST_BLUETOOTH = 1

    Request code for activating bluetooth.
- static final String CONNECTED_STATUS = "connected"

    Status to change BTDeviceItem to when connected.
- static final int BT_DISABLED_ICON = R.drawable.ic_bluetooth_disabled_black_24px

    Icon ID to use when device is not connected.

5.13.1 Detailed Description

UI Class for discovering, pairing and connecting to the logging device.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 PairDeviceFragment()

```
AndroidApp.PairDeviceFragment.PairDeviceFragment ( )  [inline]
```

Constructor for UI fragment.

Get's the mobile's bluetooth adapter and sets up our lists of used for holding devices.

```
    {
        /* Get bluetooth adapter for device & create device arrays */
        btAdapter = BluetoothAdapter.getDefaultAdapter();
        btDeviceList = new ArrayList<BTDeviceItem>();
        btPairedList = new ArrayList<BTDeviceItem>();
        btReceiver = new DiscoverReceiver();
    }
```

5.13.3 Member Function Documentation

5.13.3.1 onCreateView()

```
View AndroidApp.PairDeviceFragment.onCreateView (
             LayoutInflater inflater,
             ViewGroup container,
             Bundle savedInstanceState )  [inline]
```

Function called when fragment is shown on UI.

Sets up the UI ListView and Buttons. Add all paired devices for the bluetooth adapter to the ListView.

Parameters

inflater	- Inflater used for displaying view.
container	- Container that the view will be displayed on.
savedInstanceState	- Last known state of this fragment.

Returns

View - The UI view of this fragment.

References AndroidApp.BTDeviceItem.getConnection(), AndroidApp.BTConnection.isConnected(), and Android←App.BTConnection.isRunning().

```
{

    View myView = inflater.inflate(R.layout.pairdevice_layout, container, false);

    /* Request needed privileges for bluetooth to work */
    getNeededPrivileges();

    /* Set our variables for UI buttons */
    btnScan = (ToggleButton)myView.findViewById(R.id.pairdevice_search);
    btnScan.setOnCheckedChangeListener(new DiscoverButtonListener());

    ListView lvDevices = (ListView)myView.findViewById(R.id.pairdevice_deviceList);
    lvDevices.setOnItemClickListener(new DeviceItemListener());

    lvAdapter = new BTDeviceListAdapter(getActivity(), R.layout.device_list_item,
btDeviceList);
    lvDevices.setAdapter(lvAdapter);

    /* Check and set up bluetooth adapter */
    if (btAdapter == null)
    {
        Toast.makeText(getActivity().getApplicationContext(),
                "This device has no bluetooth adapter", Toast.LENGTH_LONG).show();
    }
    else
    {
        /* Check to see if connected device still is connected */
        if (btConnectedDevice != null)
        {
            if (!btConnectedDevice.getConnection().
isConnected() ||
                    !btConnectedDevice.getConnection().
isRunning())
            {
                btConnectedDevice = null;
            }
        }

        /* firstRun check to list from being re-populated */
        if (firstRun)
        {
            firstRun = false;

            /* Enable bluetooth adapter if disabled */
            if (!btAdapter.isEnabled())
            {
                Intent enableBT = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBT, REQUEST_BLUETOOTH);
            }

            while (!btAdapter.isEnabled())
            {
                /* Wait for BT to be enabled */
            }

            /* Add all paired devices to list */
            Set<BluetoothDevice> pairedDevices = btAdapter.getBondedDevices();
            if (pairedDevices.size() > 0)
            {
                for (BluetoothDevice device : pairedDevices)
                {
                    BTDeviceItem newDevice =
                        new BTDeviceItem(device, "paired", BT_DISABLED_ICON);
                    btPairedList.add(newDevice);
                }
            }
            btDeviceList.addAll(btPairedList);
        }

    }

    return myView;
}
```

5.13.3.2 getBTConnection()

`BTConnection AndroidApp.PairDeviceFragment.getBTConnection ( )  [inline]`

Getter for getting current connected device.

Returns

BTConnection - Bluetooth device (logging device).

References AndroidApp.BTDeviceItem.getConnection().

```
    {
        if (btConnectedDevice != null) {
            return btConnectedDevice.getConnection();
        } else {
            return null;
        }
    }
```

5.13.3.3 getNeededPrivileges()

`void AndroidApp.PairDeviceFragment.getNeededPrivileges ( )  [inline], [private]`

Prompts user for needed permissions of this application.

Due to android using a permissions/access method this method parses through each permission needed and prompts the user to accept.

```
    {
        final int REQUEST_CODE = 5;

        boolean permsGranted = true;
        String[] permsToRequest =
                {
                        Manifest.permission.BLUETOOTH_ADMIN,
                        Manifest.permission.BLUETOOTH,
                        Manifest.permission.ACCESS_FINE_LOCATION,
                        Manifest.permission.ACCESS_COARSE_LOCATION
                };

        for (String permission: permsToRequest)
        {
            permsGranted &= (ContextCompat.checkSelfPermission(getActivity(), permission) == PackageManager
      .PERMISSION_GRANTED);
        }

        if (!permsGranted)
        {
            ActivityCompat.requestPermissions(getActivity(), permsToRequest, REQUEST_CODE);
        }
    }
```

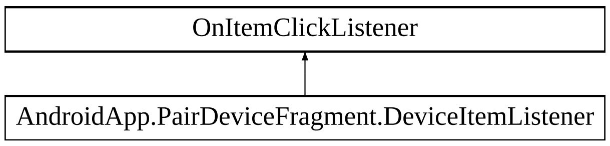
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/PairDeviceFragment.java

5.14 AndroidApp.PairDeviceFragment.DeviceItemListener Class Reference

Listener for when a ListView item is pressed (to connect).

Inheritance diagram for AndroidApp.PairDeviceFragment.DeviceItemListener:



Public Member Functions

- void onItemClick (AdapterView<?> parent, View view, int position, long id)
  Function called when user wants to connect to a device.

5.14.1 Detailed Description

Listener for when a ListView item is pressed (to connect).

5.14.2 Member Function Documentation

5.14.2.1 onItemClick()

```
void AndroidApp.PairDeviceFragment.DeviceItemListener.onItemClick (
             AdapterView<?> parent,
             View view,
             int position,
             long id )  [inline]
```

Function called when user wants to connect to a device.

Discovery is turned off to stop power wastage. A new connection thread is then created which is responsible for parsing receive, and transmission requests from other fragments.

Parameters

parent	- The parent ListView.
view	- Current view of the ListItem.
position	- Index of item pressed in ListView.
id	- ID of the ListItem.

References AndroidApp.BTDeviceItem.getConnection(), AndroidApp.BTDeviceItem.getDevice(), AndroidApp.B↩
TConnection.isConnected(), AndroidApp.BTDeviceItem.setConnection(), AndroidApp.BTDeviceItem.setIconID(),

and AndroidApp.BTDeviceItem.setStatus().

```
                                                                                {

            BTDeviceItem deviceItem = (BTDeviceItem)parent.getItemAtPosition(position);

            /* Check if there is already a connection between devices */
            if ((deviceItem.getConnection() == null) ||
                    (!deviceItem.getConnection().isConnected()))
            {
                if (btAdapter.isDiscovering())
                {
                    /* Cancel discovery is still enabled */
                    btnScan.setChecked(false);
                    btAdapter.cancelDiscovery();
                }

                try
                {
                    Toast.makeText(parent.getContext(), "Connecting to: " +
                            deviceItem.getDevice().getName(), Toast.LENGTH_SHORT).show();

                    /* Create a new BTConnection item with no RX handler */
                    BTConnection newConn = new BTConnection(deviceItem.getDevice());

                    /* Execute the 'run' procedure in object in new thread */
                    Thread tmpThread = new Thread(newConn);
                    tmpThread.start();

                    /* Add set connection and add item to listview */
                    deviceItem.setConnection(newConn);
                    btConnectedDevice = deviceItem;

                    /* Update status and icon in list view */
                    deviceItem.setIconID(R.drawable.ic_bluetooth_connected_black_24px);
                    deviceItem.setStatus(CONNECTED_STATUS);
                    lvAdapter.notifyDataSetChanged();
                }
                catch (IOException e)
                {
                    Toast.makeText(parent.getContext(), "Unable to connect: " +
                            e.toString(), Toast.LENGTH_SHORT).show();
                }
            }
        }
```

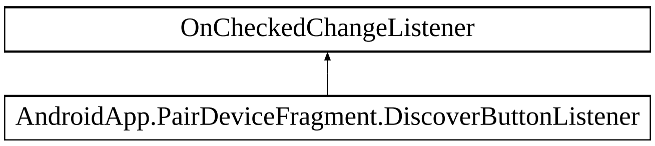
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/PairDeviceFragment.java

5.15 AndroidApp.PairDeviceFragment.DiscoverButtonListener Class Reference

Listener for when discovery button is pressed.

Inheritance diagram for AndroidApp.PairDeviceFragment.DiscoverButtonListener:



Public Member Functions

- void onCheckedChanged (CompoundButton buttonView, boolean isChecked)

    Function for handling when discover toggle button pressed.

5.15.1 Detailed Description

Listener for when discovery button is pressed.

5.15.2 Member Function Documentation

5.15.2.1 onCheckedChanged()

```
void AndroidApp.PairDeviceFragment.DiscoverButtonListener.onCheckedChanged (
            CompoundButton buttonView,
            boolean isChecked )  [inline]
```

Function for handling when discover toggle button pressed.

If toggled on it bluetooth adapter is turned to discover mode. If toggled off bluetooth adapter is turn off of disover mode.

Parameters

buttonView	- Current view of the toggle button.
isChecked	- The new state of the toggle button.

```
                                                                                            {

                IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
                if (isChecked)
                {
                    /* Clear listview, add previous paired items, start discovery */
                    lvAdapter.clear();
                    lvAdapter.addAll(btPairedList);

                    if (btConnectedDevice != null)
                        lvAdapter.add(btConnectedDevice);

                    getActivity().registerReceiver(btReceiver, filter);
                    btAdapter.startDiscovery();
                }
                else
                {
                    /* Stop searching for new devices */
                    getActivity().unregisterReceiver(btReceiver);
                    btAdapter.cancelDiscovery();
                }
            }
```

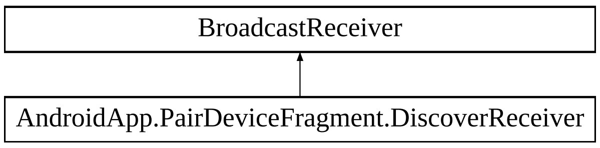
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/PairDeviceFragment.java

5.16 AndroidApp.PairDeviceFragment.DiscoverReceiver Class Reference

Receiver for when a new device is discovered.

Inheritance diagram for AndroidApp.PairDeviceFragment.DiscoverReceiver:



Public Member Functions

- void onReceive (Context context, Intent intent)

  When a BT device is found, adds the device to the ListView.

5.16.1 Detailed Description

Receiver for when a new device is discovered.

5.16.2 Member Function Documentation

5.16.2.1 onReceive()

```
void AndroidApp.PairDeviceFragment.DiscoverReceiver.onReceive (
             Context context,
             Intent intent )  [inline]
```

When a BT device is found, adds the device to the ListView.

Parameters

context	- Context that the application is running in.
intent	- Intent holding the device object.

```
                                                              {
            String action = intent.getAction();

            /* Check to see if found device */
            if (BluetoothDevice.ACTION_FOUND.equals(action))
            {
                BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

                /* Create new device item and add to list */
                BTDeviceItem newDevice = new BTDeviceItem(device, "unpaired",
      BT_DISABLED_ICON);
                lvAdapter.add(newDevice);
                lvAdapter.notifyDataSetChanged();
            }
        }
```

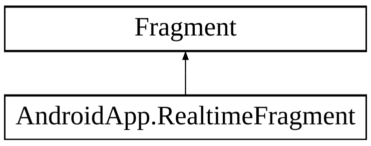
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/PairDeviceFragment.java

5.17 AndroidApp.RealtimeFragment Class Reference

UI Class for viewing data sent from the logging device.

Inheritance diagram for AndroidApp.RealtimeFragment:



Classes

- class MapButtonListener

  Listener for starting a map activity when button pressed.

Public Member Functions

- RealtimeFragment ()

  Constructor for UI fragment.
- View onCreateView (LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)

  Function called when fragment is shown on UI.

Public Attributes

- final Handler RXHandler

  Handler used for receiving statistics via bluetooth.

Private Member Functions

- final void newData (JSONObject jsonData)

  Function for adding new statistics when received via bluetooth.

Private Attributes

- TextView textStatus

  TextView to show amount of logs received.
- ArrayList< String > jsonList

  Array that holds serialised trip data to pass to map.
- SetOfDataItems dataList

  Array holding each statistic that device is measuring.
- ArrayAdapter< DataItem > lvAdapter

  Adapter used for displaying statistics in the ListView.

Static Private Attributes

- static final String NEW_LINE = "\r\n"

  String for new line parsing.

5.17.1 Detailed Description

UI Class for viewing data sent from the logging device.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 RealtimeFragment()

```
AndroidApp.RealtimeFragment.RealtimeFragment ( )  [inline]
```

Constructor for UI fragment.

Creates our initial data items that we are going to log. Setting whether extended functionality is needed for each data item.

```
                         {
        jsonList = new ArrayList<String>();

        dataList = new SetOfDataItems();

        /* Set up our data items that we will want to log */
        dataList.add(new DataItem<Double>("Yaw", true));
        dataList.add(new DataItem<Double>("Pitch", true));
        dataList.add(new DataItem<Double>("Roll", true));
        dataList.add(new DataItem<Boolean>("GPS Valid", false));
        dataList.add(new DataItem<Integer>("Satellites", false));
        dataList.add(new DataItem<Double>("Latitude", false));
        dataList.add(new DataItem<Double>("Longitude", false));
        dataList.add(new DataItem<Double>("Velocity (MPH)", true));
        dataList.add(new DataItem<Double>("Altitude (FT)", true));
        dataList.add(new DataItem<Boolean>("Date Valid", false));
        dataList.add(new DataItem<String>("Date", false));
        dataList.add(new DataItem<String>("Time", false));
    }
```

5.17.3 Member Function Documentation

5.17.3.1 onCreateView()

```
View AndroidApp.RealtimeFragment.onCreateView (
            LayoutInflater inflater,
            ViewGroup container,
            Bundle savedInstanceState )  [inline]
```

Function called when fragment is shown on UI.

Sets up the UI ListView and Buttons.

Parameters

inflater	- Inflater used for displaying view.
container	- Container that the view will be displayed on.
savedInstanceState	- Last known state of this fragment.

Returns

View - The UI view of this fragment.

```
                                                                                                   {
        View myView = inflater.inflate(R.layout.realtime_layout, container, false);

        textStatus = (TextView)myView.findViewById(R.id.realtime_status);

        /* Get the ListView via ID */
        ListView lvDataItems = (ListView) myView.findViewById(R.id.realtime_data_list);

        /* Inflate the header view for ListView */
        ViewGroup headerView = (ViewGroup) inflater.inflate(R.layout.data_list_header, lvDataItems, false);
        lvDataItems.addHeaderView(headerView);

        /* Create our new list adapter for our data list view */
        lvAdapter = new DataListAdapter(getActivity(), R.layout.data_list_item,
 dataList);
        lvDataItems.setAdapter(lvAdapter);

        /* Set our listeners for buttons */
        FloatingActionButton mapButton = (FloatingActionButton) myView.findViewById(R.id.realtime_show_map)
;
        mapButton.setOnClickListener(new MapButtonListener());

        return myView;
    }
```

5.17.3.2 newData()

```
final void AndroidApp.RealtimeFragment.newData (
              JSONObject jsonData )   [inline], [private]
```

Function for adding new statistics when received via bluetooth.

Attempts to break the initial JSON object into it's child objects and then retreive the data from these child nodes.

Parameters

jsonData	- Received JSONObject from receive handler.

References AndroidApp.SetOfDataItems.getItemByName(), and AndroidApp.DataItem< T >.setCurrent().

```
                                                 {

        try {
            JSONObject orientObject = jsonData.getJSONObject("orientation");
            JSONObject gpsObject = jsonData.getJSONObject("gps");
            JSONObject timeObject = jsonData.getJSONObject("time");

            /* Update our data items */
            dataList.getItemByName("Yaw").setCurrent(orientObject.getDouble(
 "yaw"));
```

```
            dataList.getItemByName("Pitch").setCurrent(orientObject.getDouble("pitch"));
            dataList.getItemByName("Roll").setCurrent(orientObject.getDouble("roll"));

            /* Add GPS based data to */
            dataList.getItemByName("GPS Valid").setCurrent(gpsObject.getBoolean("gps_valid"));
            dataList.getItemByName("Satellites").setCurrent(gpsObject.getInt("available"));
            dataList.getItemByName("Latitude").setCurrent(gpsObject.getDouble("lat"));
            dataList.getItemByName("Longitude").setCurrent(gpsObject.getDouble("lng"));
            dataList.getItemByName("Velocity (MPH)").setCurrent(gpsObject.getDouble("vel_mph"));
            dataList.getItemByName("Altitude (FT)").setCurrent(gpsObject.getDouble("alt_ft"));

            /* DateTime based data */
            dataList.getItemByName("Date Valid").setCurrent(timeObject.getBoolean("time_valid"));

            Calendar cal = Calendar.getInstance();
            cal.clear();
            cal.set(Calendar.YEAR, timeObject.getInt("year"));
            cal.set(Calendar.MONTH, timeObject.getInt("month"));
            cal.set(Calendar.DATE, timeObject.getInt("day"));

            cal.set(Calendar.HOUR, timeObject.getInt("hour"));
            cal.set(Calendar.MINUTE, timeObject.getInt("minute"));
            cal.set(Calendar.SECOND, timeObject.getInt("second"));
            cal.set(Calendar.MILLISECOND, timeObject.getInt("centiseconds") * 10);

            /* Create format for date and times then add to list */
            DateFormat dateFormat = new SimpleDateFormat("dd/MM/yy");
            dataList.getItemByName("Date").setCurrent(dateFormat.format(cal.getTime()));

            DateFormat timeFormat = new SimpleDateFormat("HH:mm:ss.SS");
            dataList.getItemByName("Time").setCurrent(timeFormat.format(cal.getTime()));

            lvAdapter.notifyDataSetChanged();

            /*
             * Add json object to our list
             * so we can send it to other activities/fragments later
             */
            jsonList.add(jsonData.toString());
            textStatus.setText("Reading count: " + Integer.toString(jsonList.size()));
        } catch (JSONException e) {
            /* Do nothing */
        }
    }
```

5.17.4 Member Data Documentation

5.17.4.1 RXHandler

```
final Handler AndroidApp.RealtimeFragment.RXHandler
```

Initial value:

```
= new Handler(Looper.getMainLooper()) {

        @Override
        public void handleMessage(Message msg) {

            Bundle msgData = msg.getData();
            String jsonString = msgData.getString("JSON");
            if (jsonString != null) {

                try {
                    JSONObject tmpJSON = new JSONObject(jsonString);
                    newData(tmpJSON);

                } catch (JSONException e) {

                }
            }
        }
    }
```

Handler used for receiving statistics via bluetooth.

Receives data in a bundle passed from the bluetooth connection thread. This is due to multithreading as safe data exchange between threads has to be done via messages. Attempts to parse the data into a JSON object, if successful this data is then passed to our JSON adding procedure.

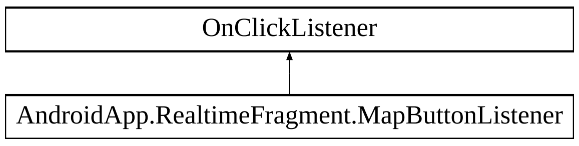
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/RealtimeFragment.java

5.18 AndroidApp.RealtimeFragment.MapButtonListener Class Reference

Listener for starting a map activity when button pressed.

Inheritance diagram for AndroidApp.RealtimeFragment.MapButtonListener:



Public Member Functions

- void onClick (View v)

  Function for handling when map button pressed.

5.18.1 Detailed Description

Listener for starting a map activity when button pressed.

5.18.2 Member Function Documentation

5.18.2.1 onClick()

```
void AndroidApp.RealtimeFragment.MapButtonListener.onClick (
             View v )  [inline]
```

Function for handling when map button pressed.

Created a new intent to start our map activity. Serialised statistics are then added as a bundle in the intent.

Parameters

v	- Current view of the button.

```
                        {
        Intent intent = new Intent(getActivity(), MapsActivity.class);
        intent.putExtra("JSONList", jsonList);
        startActivity(intent);
    }
```

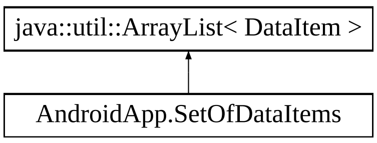
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/RealtimeFragment.java

5.19 AndroidApp.SetOfDataItems Class Reference

ArrayList extension to allow searching via item name.

Inheritance diagram for AndroidApp.SetOfDataItems:



Public Member Functions

- SetOfDataItems ()

  Constructor, just calls inhereted constructor method.
- DataItem getItemByName (String name)

  Function to allow searching of ArrayList<DataItem> via name.

5.19.1 Detailed Description

ArrayList extension to allow searching via item name.

5.19.2 Member Function Documentation

5.19.2.1 getItemByName()

```
DataItem AndroidApp.SetOfDataItems.getItemByName (
            String name )   [inline]
```

Function to allow searching of ArrayList<DataItem> via name.

Loops through all items in array until one item with matching name is found. This is then returned by the function.

Parameters

name	- Name to match.

Returns

DataItem - The item with matching name.

References AndroidApp.DataItem< T >.getName().

```
                                           {
        DataItem result = null;

        for (DataItem item: this) {
            if (item.getName().equals(name)) {
                result = item;
                break;
            }
        }

        return result;
    }
```

The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/SetOfDataItems.java

5.20 AndroidApp.TripItem Class Reference

Class used for holding name and size information relating to a trip.

Public Member Functions

- TripItem (String name, int size)

  Constructor for creating of a TripItem.
- String getTripName ()

  Getter for trip name.
- void setTripName (String tripName)

  Setter for trip name.
- int getFileSize ()

  Getter for trip filesize.
- void setFileSize (int fileSize)

  Setter for trip filesize.

Private Attributes

- String tripName = null

  The trips name on the uSD card.
- int fileSize = 0

  The trips file size on the uSD card.

5.20.1 Detailed Description

Class used for holding name and size information relating to a trip.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 TripItem()

```
AndroidApp.TripItem.TripItem (
            String name,
            int size )  [inline]
```

Constructor for creating of a TripItem.

Sets the original file name and size.

Parameters

name	- Trip name.
size	- Size of the file.

```
                                        {
        this.tripName = name;
        this.fileSize = size;
    }
```

5.20.3 Member Function Documentation

5.20.3.1 getTripName()

```
String AndroidApp.TripItem.getTripName ( )  [inline]
```

Getter for trip name.

Returns

String - Trip name.

```
                              {
        return tripName;
    }
```

5.20.3.2 setTripName()

```
void AndroidApp.TripItem.setTripName (
            String tripName )  [inline]
```

Setter for trip name.

Parameters

tripName	- New trip name.

```
                                        {
        this.tripName = tripName;
    }
```

5.20.3.3 getFileSize()

`int AndroidApp.TripItem.getFileSize ( ) [inline]`

Getter for trip filesize.

Returns

int - Filesize in bytes.

```
                        {
        return fileSize;
    }
```

5.20.3.4 setFileSize()

```
void AndroidApp.TripItem.setFileSize (
             int fileSize ) [inline]
```

Setter for trip filesize.

Parameters

fileSize	- New trip filesize.

```
                                    {
        this.fileSize = fileSize;
    }
```

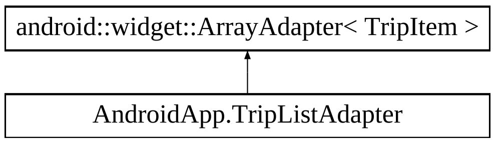
The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/TripItem.java

5.21 AndroidApp.TripListAdapter Class Reference

Adapter class used for displaying all trips.

Inheritance diagram for AndroidApp.TripListAdapter:


android::widget::ArrayAdapter< TripItem >
AndroidApp.TripListAdapter

Classes

- class ViewHolder

  Class that holds all UI data to be displayed for each ListItem.

Public Member Functions

- TripListAdapter (Context cnt, int layoutResourceId, ArrayList< TripItem > data)

  Constructor for the ListView adapter.
- View getView (int position, View convertView, ViewGroup parent)

  Function for returning the view of each list item (TripItem).

Private Attributes

- Context context

  Context that the ListView is operating in.
- int layoutResourceId

  Resource ID for current layout.
- ArrayList< TripItem > data

  ArrayList of all trip items to display.

5.21.1 Detailed Description

Adapter class used for displaying all trips.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 TripListAdapter()

```
AndroidApp.TripListAdapter.TripListAdapter (
              Context cnt,
              int layoutResourceId,
              ArrayList< TripItem > data )  [inline]
```

Constructor for the ListView adapter.

Calls the constructor of the superclass as well as setting other relevant information needed.

Parameters

cnt	- Context of the adapter to be operating in.
layout↩ResourceId	- Resource ID for current layout.
data	- ArrayList of statistics to display in ListView.

```
                                                                          {
        super(cnt, layoutResourceId, data);

        this.context = cnt;
        this.layoutResourceId = layoutResourceId;
        this.data = data;
    }
```

5.21.3 Member Function Documentation

5.21.3.1 getView()

```
View AndroidApp.TripListAdapter.getView (
            int position,
            View convertView,
            ViewGroup parent )  [inline]
```

Function for returning the view of each list item (TripItem).

If a view for selected item has not been created inflater initialises it. A holder is then used to hold all the information that will be displayed on the UI to the user.

Parameters

position	- Index of item in array to use/reference to.
convertView	- View to be used for specified item.
parent	- Object where the created view will be placed on.

Returns

View - The result view of item with updated/current information.

References AndroidApp.TripItem.getFileSize(), and AndroidApp.TripItem.getTripName().

```
                                                              {
        ViewHolder holder;

        if (convertView == null)
        {
            /* If view does not already exist. */
            LayoutInflater inflater = (LayoutInflater)context.getSystemService(Context.
    LAYOUT_INFLATER_SERVICE);
            convertView = inflater.inflate(layoutResourceId, parent, false);

            holder = new ViewHolder();
            holder.name = (TextView)convertView.findViewById(R.id.triplist_name);
            holder.fileSize = (TextView)convertView.findViewById(R.id.triplist_size);
            convertView.setTag(holder);
        }
        else
        {
            /* If view already exists. */
            holder = (ViewHolder)convertView.getTag();
        }

        TripItem tripItem = getItem(position);

        /* Set our holder with current data of item */
         holder.name.setText(tripItem.getTripName());
         holder.fileSize.setText(Integer.toString(tripItem.getFileSize()));

         return convertView;
    }
```

The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/TripListAdapter.java

5.22 AndroidApp.TripListAdapter.ViewHolder Class Reference

Class that holds all UI data to be displayed for each ListItem.

5.22.1 Detailed Description

Class that holds all UI data to be displayed for each ListItem.

The documentation for this class was generated from the following file:

- android-app/app/src/main/java/com/jack/motorbikestatistics/TripListAdapter.java

5.23 LoggingDevice::Orientation Class Reference

Class for dealing with Orientation functionality on logging device.

`#include <Orientation.h>`

Public Member Functions

- void init ()

  Initialisation function for orientation module.
- bool pollIMU ()

  Updates the IMU with newest values at 25Hz frequency.
- float getYaw ()

  Returns the Yaw orientation of the device.
- float getPitch ()

  Returns the Pitch orientation of the device.
- float getRoll ()

  Returns the Roll orientation of the device.

Private Member Functions

- float convertRawAccel (int aRaw)

  Converts a raw reading from accelerometer to a value in G.
- float convertRawGyro (int aRaw)

  Converts a raw reading from gyro to a value in deg/sec.

Private Attributes

- Madgwick IMUfilter

  Madgwick filter object uses to steady orientation readings.

5.23.1 Detailed Description

Class for dealing with Orientation functionality on logging device.

5.23.2 Member Function Documentation

5.23.2.1 convertRawAccel()

```
float Orientation::convertRawAccel (
              int aRaw )  [private]
```

Converts a raw reading from accelerometer to a value in G.

Parameters

aRaw	- Raw accelerometer axis value.

Returns

float - Processed acceleration axis in G.

References ACCEL_RANGE.

```
{
  /*
   * Since using 2G range.
   * -2G maps to raw value of -32768
   * +2G maps to raw value of +32767
   */
   float a = (aRaw * (float)ACCEL_RANGE) / 32768.0;
   return a;
}
```

5.23.2.2 convertRawGyro()

```
float Orientation::convertRawGyro (
            int gRaw )  [private]
```

Converts a raw reading from gyro to a value in deg/sec.

Parameters

gRaw	- Raw gyroscope axis value.

Returns

float - Processed rotation axis in deg/sec.

References GYRO_RANGE.

```
{
  /*
   * since we are using 250 degrees/seconds range
   * -250 maps to a raw value of -32768
   * +250 maps to a raw value of 32767
   */
  float g = (gRaw * (float)GYRO_RANGE) / 32768.0;
  return g;
}
```

5.23.2.3 init()

```
void Orientation::init ( )
```

Initialisation function for orientation module.

Initialises the CurieIMU module with set ranges and rates, our Madgwick filter is also initialised with this information.

References ACCEL_RANGE, GYRO_RANGE, IMU_FREQUENCY, and IMUfilter.

```
{
  /* Set up the Gyroscope + Accelerometer */
  CurieIMU.begin();
  CurieIMU.setGyroRate(IMU_FREQUENCY);
  CurieIMU.setAccelerometerRate(IMU_FREQUENCY);
  CurieIMU.setAccelerometerRange(ACCEL_RANGE);
  CurieIMU.setGyroRange(GYRO_RANGE);

  IMUfilter.begin(IMU_FREQUENCY);
}
```

5.23.2.4 pollIMU()

`bool Orientation::pollIMU ( )`

Updates the IMU with newest values at 25Hz frequency.

Function reads raw values from accelerometer and gyroscope and sends them to our Madgwick filter (IMUfilter).
This function needs to be called by the system as often as possible.
To ensure correct frequency of 25Hz if kept to a micros counter is in place.
Function will return true or false as of whether that call actually updated the IMU (depending on micros count check).

Returns

bool - Whether the IMU was actually updated.

References AXIS_X, AXIS_Y, AXIS_Z, convertRawAccel(), convertRawGyro(), IMU_FREQUENCY, IMUfilter, and NUMBER_AXIS.

```
{
  static const unsigned long US_PER_READING = 1000000 / IMU_FREQUENCY;
  static unsigned long usPrevious = micros();

  bool result = false;
  int accel_raw[NUMBER_AXIS];
  int gyro_raw[NUMBER_AXIS];
  float accel_g[NUMBER_AXIS];
  float gyro_ds[NUMBER_AXIS];
  unsigned long usNow;

  /* Ensures we stick to the sample rate (by not sampling too early) */
  usNow = micros();
  if ((usNow - usPrevious) >= US_PER_READING)
  {
    /* Read raw data from the IMU */
    CurieIMU.readMotionSensor(accel_raw[AXIS_X], accel_raw[AXIS_Y], accel_raw[
   AXIS_Z],
                                 gyro_raw[AXIS_X], gyro_raw[AXIS_Y], gyro_raw[AXIS_Z]);

    /* Convert raw readings from IMU to accel (G) and rotation vel (deg/s) */
    accel_g[AXIS_X] = convertRawAccel(accel_raw[AXIS_X]);
    accel_g[AXIS_Y] = convertRawAccel(accel_raw[AXIS_Y]);
    accel_g[AXIS_Z] = convertRawAccel(accel_raw[AXIS_Z]);
    gyro_ds[AXIS_X] = convertRawGyro(gyro_raw[AXIS_X]);
    gyro_ds[AXIS_Y] = convertRawGyro(gyro_raw[AXIS_Y]);
    gyro_ds[AXIS_Z] = convertRawGyro(gyro_raw[AXIS_Z]);

    /* Update the filter. Orientation is calculated here */
    IMUfilter.updateIMU(gyro_ds[AXIS_X], gyro_ds[AXIS_Y], gyro_ds[AXIS_Z],
                         accel_g[AXIS_X], accel_g[AXIS_Y], accel_g[AXIS_Z]);

     /* Increment previous counter */
     usPrevious += US_PER_READING;

     result = true;
   }

   return result;
 }
```

5.23.2.5 getYaw()

`float Orientation::getYaw ( )`

Returns the Yaw orientation of the device.

Returns

float - Yaw orientation.

References IMUfilter.

```
{
  return IMUfilter.getYaw();
}
```

5.23.2.6 getPitch()

```
float Orientation::getPitch ( )
```

Returns the Pitch orientation of the device.

Returns

float - Pitch orientation.

References IMUfilter.

```
{
  return IMUfilter.getPitch();
}
```

5.23.2.7 getRoll()

```
float Orientation::getRoll ( )
```

Returns the Roll orientation of the device.

Returns

float - Roll orientation.

References IMUfilter.

```
{
  return IMUfilter.getRoll();
}
```

The documentation for this class was generated from the following files:

- logging-device/Orientation.h
- logging-device/Orientation.cpp

5.24 LoggingDevice::Storage Class Reference

Class for storing & retrieving data on the logging device.

`#include <Storage.h>`

Public Member Functions

- void init ()

  Initialisation function for storage module.
- bool saveToFile (char data[ ], bool newLine)

  Saves a single line of data to a file.
- bool generateFileName ()

  Generates a new filename to use for saving.
- void loadTripNames ()

  Loads the information of all trips and sends them over bluetooth.
- void loadSavedTrip ()

  Loads a saved trip and sends data to client via Serial.

Private Attributes

- char fileName [30]

  File name to use when saving data.
- StaticJsonBuffer< 200 > jsonFileBuffer

  Allocated space for holding JSON objects within.
- JsonObject & fileJSON = jsonFileBuffer.createObject()

  JSON object that holds file information (size + name)

5.24.1 Detailed Description

Class for storing & retrieving data on the logging device.

5.24.2 Member Function Documentation

5.24.2.1 init()

```
void Storage::init ( )
```

Initialisation function for storage module.

Responsible for starting the uSD library.

References USD_CS.

```
{
  SD.begin(USD_CS);
}
```

5.24.2.2 saveToFile()

```
bool Storage::saveToFile (
            char data[],
            bool newLine )
```

Saves a single line of data to a file.

Opens a handle to the current fileName. If the file exists data is appended, if not the file is created first.

Parameters

data	- Character array of data to save.
newLine	- Whether to add new line character at end of line.

Returns

bool - Whether saving was a success.

References fileName.

```
{
  bool result = false;

  /* Create handle to log file */
  File logHandle = SD.open(fileName, FILE_WRITE);

  /* If handle exists print line to file */
  if (logHandle)
  {

    /* Print line, option to add newline characters */
    logHandle.print(data);
    if (newLine)
    {
      logHandle.println();
    }

    logHandle.close();
    result = true;
  }
  return result;
}
```

5.24.2.3 generateFileName()

```
bool Storage::generateFileName ( )
```

Generates a new filename to use for saving.

Searches through existing files using pattern PREFIX_ID.SUFFIX
Existing files are skipped, once non-existant is found that is used.

Returns

bool - Whether a valid file name was able to be found.

References fileName, LOG_EXTENSION, LOG_NAME, and MAX_LOG_FILES.

```
{
  bool result = false;
  int i = 0;

  for (i = 0; i < MAX_LOG_FILES; i++)
  {
    /* Clear name of log file */
    memset(fileName, 0, strlen(fileName));

    /* Set the new log file name to: trip_XXXXX.json */
    sprintf(fileName, "%s%d.%s", LOG_NAME, i, LOG_EXTENSION);

    if (!SD.exists(fileName))
     {
       /* If a file doesn't exist */
       result = true;
       break;
     }
   }

   return result;
 }
```

5.24.2.4 loadTripNames()

`void Storage::loadTripNames ( )`

Loads the information of all trips and sends them over bluetooth.

Searches directory for trips, then sends trip's name & size over serial.

References BT_SERIAL, and fileJSON.

```
{
  bool filesRemaining = true;

  File root = SD.open("/");

  /* Try to open directory for logs */
  if (root)
  {
    /* Ensure starting from start of directory */
    root.rewindDirectory();

    while (filesRemaining == true)
    {
      /* Try open handle for next file */
      File entry = root.openNextFile();
      if (entry)
      {
        if (entry.isDirectory() == false)
        {
          /* Print out file name & size */
          fileJSON["name"] = entry.name();
          fileJSON["size"] = entry.size();

          fileJSON.printTo(BT_SERIAL);
          BT_SERIAL.println();
        }
        entry.close();
      }
      else
      {
        /* No more files remaining in directory */
        filesRemaining = false;
      }
    }

    root.close();
  }
}
```

5.24.2.5 loadSavedTrip()

```
void Storage::loadSavedTrip ( )
```

Loads a saved trip and sends data to client via Serial.

Waits for the filename to be received via serial. Once file name is received, procedure attempts to open the file. If the file exists it then sends all bytes in the file via Serial.

References BT_SERIAL, and LOG_EXTENSION.

```
{
  bool nameComplete = false;
  String fileToOpen = "";

  while (nameComplete == false)
  {
    /* Keep reading input in serial until file name is found */
    if (BT_SERIAL.available() > 0)
    {
      char recvByte = BT_SERIAL.read();
      fileToOpen += recvByte;

      /* Wait until extension is found, then we know full file name */
      if (fileToOpen.endsWith(LOG_EXTENSION))
      {
        nameComplete = true;
      }
    }
  }

  /* Check if file exists */
  if (SD.exists(fileToOpen))
  {
    /* Open file, then read out data byte by byte */
    File handle = SD.open(fileToOpen);
    if (handle)
    {

      while (handle.available())
      {
        char readByte = handle.read();

        BT_SERIAL.write(readByte);
      }

      handle.close();
    }
  }
}
```

The documentation for this class was generated from the following files:

- logging-device/Storage.h
- logging-device/Storage.cpp

Chapter 6

File Documentation

6.1 android-app/app/src/main/java/com/jack/motorbikestatistics/BTConnection.java File Reference

Class for holding containing bluetooth connection on app.

Classes

- class AndroidApp.BTConnection

  Thread class for a new bluetooth connection to a device.

6.1.1 Detailed Description

Class for holding containing bluetooth connection on app.

Class runs in a seperate thread to main UI allowing for concurrent transmission and receiving of data to/from the logging device.

Author

Jack Allister - 23042098

Date

2016-2017

6.2 android-app/app/src/main/java/com/jack/motorbikestatistics/BTDeviceItem.java File Reference

UI class for holding information regarding a bluetooth device.

Classes

- class AndroidApp.BTDeviceItem

  Class used for holding core UI information of a bluetooth devices.

6.2.1 Detailed Description

UI class for holding information regarding a bluetooth device.

Implemented for the ListView that shows unpaired/paired & connected bluetooth devices.

Author

Jack Allister - 23042098

Date

2016-2017

6.3 android-app/app/src/main/java/com/jack/motorbikestatistics/BTDeviceListAdapter.java File Reference

UI ListView adapter to display bluetooth devices.

Classes

- class AndroidApp.BTDeviceListAdapter

  Adapter class used for displaying bluetooth devices.
- class AndroidApp.BTDeviceListAdapter.ViewHolder

  Class that holds all data displayed for each ListItem.

6.3.1 Detailed Description

UI ListView adapter to display bluetooth devices.

Implemented so that the device ListView can display relevant information relating to the BluetoothDevice's that are available to pair, connect.

Author

Jack Allister - 23042098

Date

2016-2017

6.4 android-app/app/src/main/java/com/jack/motorbikestatistics/DataItem.java File Reference

UI class for holding information regarding a specific statistic.

Classes

- class AndroidApp.DataItem< T >

    Class used for holding and displaying a piece of data within the statistic ListView UI.

6.4.1 Detailed Description

UI class for holding information regarding a specific statistic.

Implementation of generic class to allow multiple data types android added functionality such as averaging, minimum and maximum.

Author

Jack Allister - 23042098

Date

2016-2017

6.5 android-app/app/src/main/java/com/jack/motorbikestatistics/DataListAdapter.java File Reference

UI ListView adapter to display statistics.

Classes

- class AndroidApp.DataListAdapter

    Adapter class used for displaying statistics.
- class AndroidApp.DataListAdapter.ViewHolder

    Class that holds all data displayed for each ListItem.

6.5.1 Detailed Description

UI ListView adapter to display statistics.

Implemented so that the statistics ListView can display relevant information relating to the statistic such as name, value, average, min & max.

Author

Jack Allister - 23042098

Date

2016-2017

6.6 android-app/app/src/main/java/com/jack/motorbikestatistics/LoadDeviceFragment.java File Reference

Fragment/Tab for providing UI for loading from device.

Classes

- class AndroidApp.LoadDeviceFragment

  UI Class for loading saved trips from device.
- class AndroidApp.LoadDeviceFragment.TripItemListener

  Listener used to identify when a trip has been pressed.

6.6.1 Detailed Description

Fragment/Tab for providing UI for loading from device.

UI to allow the user to load saved trips stored on the uSD of the logging device.

Author

Jack Allister - 23042098

Date

2016-2017

6.7 android-app/app/src/main/java/com/jack/motorbikestatistics/MainActivity.java File Reference

Main activity class responsible for tabbing.

Classes

- class AndroidApp.MainActivity

  Main activity class for fragment navigation.

6.7.1 Detailed Description

Main activity class responsible for tabbing.

Responsible for navigation between each fragment/tab. Sends relevant commands to switch system modes on the logging device as well.

Author

Jack Allister - 23042098

Date

2016-2017

6.8 android-app/app/src/main/java/com/jack/motorbikestatistics/MapsActivity.java File Reference

Maps activity class reponsible for showing data on Google Maps.

Classes

- class AndroidApp.MapsActivity

  Maps activity class for displaying map data.
- class AndroidApp.MapsActivity.StatisticWindowAdapter

  Adapter used for displaying statistics at a certain marker that user has clicked on.

6.8.1 Detailed Description

Maps activity class reponsible for showing data on Google Maps.

Responsible for showing trip data on google maps. Places clickable points 5m away from each other showing stats at that point.

Author

Jack Allister - 23042098

Date

2016-2017

6.9 android-app/app/src/main/java/com/jack/motorbikestatistics/PairDeviceFragment.java File Reference

Fragment/Tab for connecting to the logging device.

Classes

- class AndroidApp.PairDeviceFragment

  UI Class for discovering, pairing and connecting to the logging device.
- class AndroidApp.PairDeviceFragment.DiscoverReceiver

  Receiver for when a new device is discovered.
- class AndroidApp.PairDeviceFragment.DiscoverButtonListener

  Listener for when discovery button is pressed.
- class AndroidApp.PairDeviceFragment.DeviceItemListener

  Listener for when a ListView item is pressed (to connect).

6.9.1 Detailed Description

Fragment/Tab for connecting to the logging device.

Implements Android's bluetooth API to discover, pair and connecting to the logging device.

Communication to the logging device is done via using Serial data mode.

Author

Jack Allister - 23042098

Date

2016-2017

6.10 android-app/app/src/main/java/com/jack/motorbikestatistics/RealtimeFragment.java File Reference

Fragment/Tab for viewing streamed statistics.

Classes

- class AndroidApp.RealtimeFragment

  UI Class for viewing data sent from the logging device.
- class AndroidApp.RealtimeFragment.MapButtonListener

  Listener for starting a map activity when button pressed.

6.10.1 Detailed Description

Fragment/Tab for viewing streamed statistics.

Implements RXHandler from bluetooth device to receive statistics. Data is then displayed in a ListView as well as option to view via Google Maps.

Author

Jack Allister - 23042098

Date

2016-2017

6.11 android-app/app/src/main/java/com/jack/motorbikestatistics/SetOfDataItems.java File Reference

Extension of ArrayList allows for searching via name.

Classes

- class AndroidApp.SetOfDataItems

  ArrayList extension to allow searching via item name.

6.11.1 Detailed Description

Extension of ArrayList allows for searching via name.

This class is created to allow RealtimeFragment to search items by name. Simple searches through all items for a matching name.

Author

Jack Allister - 23042098

Date

2016-2017

6.12 android-app/app/src/main/java/com/jack/motorbikestatistics/TripItem.java File Reference

Class for holding information relating to a specific trip.

Classes

- class AndroidApp.TripItem

  Class used for holding name and size information relating to a trip.

6.12.1 Detailed Description

Class for holding information relating to a specific trip.

Holds the trips name and file size. This information is used when loading a previous trip.

Author

Jack Allister - 23042098

Date

2016-2017

6.13 android-app/app/src/main/java/com/jack/motorbikestatistics/TripListAdapter.java File Reference

UI ListView adapter to display all saved trips.

Classes

- class AndroidApp.TripListAdapter

  Adapter class used for displaying all trips.
- class AndroidApp.TripListAdapter.ViewHolder

  Class that holds all UI data to be displayed for each ListItem.

6.13.1 Detailed Description

UI ListView adapter to display all saved trips.

Implemented so that the trip list ListView can display relevant information relating to the statistic such as name and file size.

Author

Jack Allister - 23042098

Date

2016-2017

6.14 logging-device/logging-device.ino File Reference

Arduino sketch for the logging device.

```
#include <SoftwareSerial.h>
#include <TinyGPS++.h>
#include <ArduinoJson.h>
#include "Orientation.h"
#include "Storage.h"
```

Macros

- #define IDLE_CHAR '0'

  Command to set system to idle mode.
- #define REALTIME_CHAR '1'

  Command to set system to realtime logging mode.
- #define LIST_SAVED_CHAR '2'

  Command to list all saved trip names from uSD.
- #define LOAD_TRIP_CHAR '3'

  Command to load a trip stored on uSD.
- #define BT_SERIAL Serial1

  Mapping for which HW-Serial port BT module is on.
- #define BT_BAUD 115200

  BAUD rate of BT device.
- #define GPS_TX_PIN 9

  GPS serial transmit pin.
- #define GPS_RX_PIN 8

  GPS serial receive pin.
- #define GPS_BAUD 9600

  GPS serial baud rate.
- #define LED_PIN 13

  LED pin to indicate read.

Enumerations

- enum OPERATING_MODE { **IDLE**, **REALTIME** }

  Typedef holding two possible states for device.

Functions

- SoftwareSerial serGPS (GPS_RX_PIN, GPS_TX_PIN)

  Serial object for communicating with GPS module.
- void setup ()

  Runs once at boot of arduino.
- void loop ()

  Main system loop for arduino.
- bool parseNewMode (char modeChar, OPERATING_MODE &newMode)

  Returns whether system should change operating mode.
- void realTimeMode ()

  Responsible for completing work needed in relatime mode.
- void addOrientationToJSON ()

  Responsible for updating orientation JSON object with newest information.
- void addGPSToJSON ()

  Responsible for updating GPS JSON object with newest information.
- void addTimeToJSON ()

  Responsible for updating time JSON object with newest information.

Variables

- OPERATING_MODE systemMode = IDLE

  State machine for system state of device.
- Orientation orientation

  Orientation object, used for receiving device orientation.
- Storage storage

  Storage object, responsible for saving & loading from uSD.
- TinyGPSPlus gps

  Our GPS object, responsible for parsing NMEA codes.
- StaticJsonBuffer< 500 > jsonBuffer

  Allocated space for holding all JSON objects within.
- JsonObject & mainJSON = jsonBuffer.createObject()

  Parent JSON object, holds orientation, time & gps children.
- JsonObject & orientJSON = mainJSON.createNestedObject("orientation")

  Holds all orientation related information.
- JsonObject & gpsJSON = mainJSON.createNestedObject("gps")

  Holds all location related information.
- JsonObject & timeJSON = mainJSON.createNestedObject("time")

  Holds all time related inforamtion.

6.14.1 Detailed Description

Arduino sketch for the logging device.

Author

Jack Allister - 23042098

Date

2016-2017

- Arduino 101
- Sparkfun GPS Logger shield
- Onboard gyroscope + accelerometer
- HC-06 Serial Bluetooth Module

6.14.2 Function Documentation

6.14.2.1 setup()

`void setup ( )`

Runs once at boot of arduino.

Responsible for setting up the peripherals.
Initialises modules such as storage, bluetooth & gps.

References BT_BAUD, BT_SERIAL, GPS_BAUD, LoggingDevice::Storage::init(), LoggingDevice::Orientation::init(), LED_PIN, and serGPS().

```
{
  pinMode(LED_PIN, OUTPUT);

  /* Initialise our created modules */
  storage.init();
  orientation.init();

  /* Set up serial for wireless data transmission */
  BT_SERIAL.begin(BT_BAUD);

  /* Set up serial for GPS module */
  serGPS.begin(GPS_BAUD);
}
```

6.14.2.2 loop()

`void loop ( )`

Main system loop for arduino.

Checks serial to see if any commands are available.
If available reads the byte and changes system mode relating to it.

System state machine is also iterated through each loop.
Relevant procedure depending on system state is then called.

References BT_SERIAL, parseNewMode(), and systemMode.

```
{

  /* Check if mode change character received from front-end */
  if (BT_SERIAL.available() > 0)
  {
    char modeChar = BT_SERIAL.read();

    OPERATING_MODE newMode;

    /* If valid new mode character found change system state */
    if (parseNewMode(modeChar, newMode) == true)
    {
      systemMode = newMode;
    }
  }

  /* State machine for choosing what option takes place */
  switch (systemMode)
  {
    case IDLE:
    {
      /*
       * In IDLE mode MCU does nothing.
       * System waits and still parses incoming commands.
       */
      break;
    }

    case REALTIME:
    {
      realTimeMode();
      break;
    }
  }
}
```

6.14.2.3 parseNewMode()

```
bool parseNewMode (
            char modeChar,
            OPERATING_MODE & newMode )
```

Returns whether system should change operating mode.

Parameters

modeChar	- The received command byte
&newMode	- Reference to new operating mode calculated via command.

Returns

bool - Whether a valid command was found.

References IDLE_CHAR.

```
{
  bool result = true;

  switch (modeChar)
  {
    case IDLE_CHAR:
    {
      newMode = IDLE;
      break;
    }

    case REALTIME_CHAR:
    {
      /* Change mode and then generate new file name for new log */
      if (systemMode != REALTIME)
      {
        /* Generate new name if not already in this mode */
        storage.generateFileName();
      }

      newMode = REALTIME;
      break;
    }

    case LIST_SAVED_CHAR:
    {
      /*
       * Load all trips and send to application.
       * Once we have finished sending trips we can go back to idle mode.
       */
      storage.loadTripNames();
      newMode = IDLE;
      break;
    }

    case LOAD_TRIP_CHAR:
    {
      /* Load a specific trip by file name */
      storage.loadSavedTrip();
      newMode = IDLE;
      break;
    }

    default:
    {
      /*
       * If not a valid operating mode character
       * then return that parsing failed.
       */
      result = false;
    }
  }

  return result;
}
```

6.14.2.4 realTimeMode()

```
void realTimeMode ( )
```

Responsible for completing work needed in relatime mode.

Every time called this procedure will poll the IMU to update our orientation class with newest information.
If available NMEA sentences received from GPS serial are sent to our GPS parsing object.
Every 1000ms all current information is transmitted via bluetooth, this information is also stored to the uSD so it can be retrieved at a later point.

References addGPSToJSON(), addOrientationToJSON(), addTimeToJSON(), BT_SERIAL, gps, LED_PIN, main↩
JSON, LoggingDevice::Orientation::pollIMU(), LoggingDevice::Storage::saveToFile(), and serGPS().

```
{
  static const unsigned int MAX_STRING_SIZE = 512;
  static const unsigned long PRINT_DELAY = 1000;
  static unsigned long lastMillis = 0;
  char jsonString[MAX_STRING_SIZE];

  /* Poll our IMU to update XYZ */
  orientation.pollIMU();

  /* Parse NMEA codes into GPS object */
  while (serGPS.available() > 0)
  {
    gps.encode(serGPS.read());
  }

  /* Print orientation and location information */
  if ((millis() - lastMillis) > PRINT_DELAY)
  {
    digitalWrite(LED_PIN, HIGH);

    addOrientationToJSON();
    addGPSToJSON();
    addTimeToJSON();

    /* Print our json object into a string */
    mainJSON.printTo(jsonString, MAX_STRING_SIZE);

    /* Log JSON to the microSD */
    storage.saveToFile(jsonString, true);

    /* Print to our bluetooth module */
    BT_SERIAL.println(jsonString);

    lastMillis = millis();
    digitalWrite(LED_PIN, LOW);
  }
}
```

6.14.2.5 addOrientationToJSON()

```
void addOrientationToJSON ( )
```

Responsible for updating orientation JSON object with newest information.

Interacts with devices Orientation object to get Yaw, Pitch & Roll.

References LoggingDevice::Orientation::getPitch(), LoggingDevice::Orientation::getRoll(), and LoggingDevice::↩
Orientation::getYaw().

```
{
  orientJSON["yaw"] = orientation.getYaw();
  orientJSON["pitch"] = orientation.getPitch();
  orientJSON["roll"] = orientation.getRoll();
}
```

6.14.2.6 addGPSToJSON()

```
void addGPSToJSON ( )
```

Responsible for updating GPS JSON object with newest information.

Interacts with devices TinyGPSPlus object to get all locational/gps related information. Floats are cat'd to 6 digits max.

References gps.

```
{
  /* Add location information */
  gpsJSON["gps_valid"] = gps.location.isUpdated();
  gpsJSON["lat"] = double_with_n_digits(gps.location.lat(), 6);
  gpsJSON["lng"] = double_with_n_digits(gps.location.lng(), 6);

  /* Other crucial GPS information */
  gpsJSON["available"] = gps.satellites.value();
  gpsJSON["vel_mph"] = gps.speed.mph();
  gpsJSON["alt_ft"] = gps.altitude.feet();
}
```

6.14.2.7 addTimeToJSON()

```
void addTimeToJSON ( )
```

Responsible for updating time JSON object with newest information.

Interacts with devices TinyGPSPlus object to get time related information. This is because GPS module has a RTC (Realtime-Clock) kept via NMEA sentences.

References gps.

```
{
  /* Add time information to JSON */
  timeJSON["time_valid"] = gps.date.isValid() && gps.time.isValid();
  timeJSON["day"] = gps.date.day();
  timeJSON["month"] = gps.date.month();
  timeJSON["year"] = gps.date.year();

  timeJSON["hour"] = gps.time.hour();
  timeJSON["minute"] = gps.time.minute();
  timeJSON["second"] = gps.time.second();
  timeJSON["centiseconds"] = gps.time.centisecond();
}
```

6.15 logging-device/Orientation.cpp File Reference

Module created to deal with all orientation related functionality.

```
#include <BMI160.h>
#include <CurieIMU.h>
#include "Orientation.h"
```

Macros

- #define IMU_FREQUENCY 25

  Frequency of update rate for IMU (25Hz)
- #define ACCEL_RANGE 2

  Range of acelerometer +-2G.
- #define GYRO_RANGE 250

  Range of gyroscope +-250 deg/sec.
- #define NUMBER_AXIS 3

  Number of axis for our IMU.
- #define AXIS_X 0

  Reference to X axis in array.
- #define AXIS_Y 1

  Reference to Y axis in array.
- #define AXIS_Z 2

  Reference to Z axis in array.

6.15.1 Detailed Description

Module created to deal with all orientation related functionality.

Author

Jack Allister - 23042098

Date

2016-2017 Uses the built in Gyroscope & Accelerometer of the Arduino 101 to create an Inertial Measurement Unit (IMU).

6.16 logging-device/Storage.cpp File Reference

Module created to handle all storage related functionality.

```
#include <SD.h>
#include <ArduinoJson.h>
#include "Storage.h"
```

Macros

- #define BT_SERIAL Serial1

  Mapping for which HW-Serial port BT module is on.
- #define USD_CS 10

  Chip select pin for MicroSD card (SPI)
- #define MAX_LOG_FILES 5000

  Maximum amount of log files that can be stored on the device.
- #define LOG_NAME "TRIP_"

  The prefix of the name for logs.
- #define LOG_EXTENSION "TXT"

  The suffix of the name for logs (file extension)

6.16.1 Detailed Description

Module created to handle all storage related functionality.

Author

Jack Allister - 23042098

Date

2016-2017 Handles saving, listing & loading of trips. Uses MicroSD available on the Sparkfun GPS logging shield.

Index