



Activity Logger

1.0

Generated by Doxygen 1.8.14

Contents

1	COMP-2005 Activity Logger Documentation	1
2	Namespace Index	3
2.1	Packages	3
3	Hierarchical Index	5
3.1	Class Hierarchy	5
4	Class Index	7
4.1	Class List	7
5	File Index	9
5.1	File List	9
6	Namespace Documentation	11
6.1	Package com	11
6.2	Package com.activitytracker	11

7 Class Documentation	13
7.1 com.activitytracker.ActivityTracker Class Reference	13
7.1.1 Detailed Description	13
7.1.2 Member Function Documentation	14
7.1.2.1 main()	14
7.2 com.activitytracker.CreateUserWindow Class Reference	15
7.2.1 Detailed Description	17
7.2.2 Constructor & Destructor Documentation	17
7.2.2.1 CreateUserWindow()	18
7.2.3 Member Function Documentation	18
7.2.3.1 rootPanel()	18
7.2.3.2 setupActionListeners()	19
7.2.3.3 setupUI()	19
7.2.4 Member Data Documentation	20
7.2.4.1 buttonCancel	20
7.2.4.2 buttonOk	20
7.2.4.3 comboBoxBirthDay	20
7.2.4.4 comboBoxBirthMonth	20
7.2.4.5 comboBoxSex	21
7.2.4.6 labelInfo	21
7.2.4.7 m_closeWindowHandler	21
7.2.4.8 m_dbmanager	21
7.2.4.9 m_rootPanel	21
7.2.4.10 passwordField	22
7.2.4.11 textFieldBirthYear	22

7.2.4.12 textFieldEmail	22
7.2.4.13 textFieldHeight	22
7.2.4.14 textFieldName	22
7.2.4.15 textFieldWeight	23
7.3 com.activitytracker.DBManager Class Reference	23
7.3.1 Detailed Description	25
7.3.2 Constructor & Destructor Documentation	25
7.3.2.1 DBManager()	25
7.3.3 Member Function Documentation	26
7.3.3.1 createUser()	26
7.3.3.2 executeQuery()	27
7.3.3.3 executeUpdate()	28
7.3.3.4 getDateOfBirth()	29
7.3.3.5 getRunDate()	30
7.3.3.6 getRunFloatAttribute()	30
7.3.3.7 getRuns()	32
7.3.3.8 getUserFloatAttribute()	33
7.3.3.9 getUserIDByEmail()	34
7.3.3.10 getUserLastRID()	35
7.3.3.11 getUserPassSalt()	36
7.3.3.12 getUserSex()	37
7.3.3.13 getUserStringAttribute()	38
7.3.3.14 init()	40
7.3.3.15 isEmpty()	41
7.3.3.16 newRun()	42

7.3.3.17	runExists()	44
7.3.3.18	setRun()	45
7.3.3.19	setUserLastRID()	46
7.3.3.20	userExists()	47
7.3.4	Member Data Documentation	47
7.3.4.1	m_conn	48
7.4	com.activitytracker.Iteration3Test Class Reference	48
7.4.1	Detailed Description	48
7.4.2	Member Function Documentation	49
7.4.2.1	main()	49
7.5	com.activitytracker.LoginWindow Class Reference	52
7.5.1	Detailed Description	54
7.5.2	Constructor & Destructor Documentation	54
7.5.2.1	LoginWindow()	55
7.5.3	Member Function Documentation	55
7.5.3.1	rootPanel()	55
7.5.3.2	setupActionListeners()	56
7.5.3.3	setupCreateUserDialog()	56
7.5.3.4	setupUI()	57
7.5.4	Member Data Documentation	57
7.5.4.1	buttonCreateUser	57
7.5.4.2	buttonLogin	58
7.5.4.3	labelLoginMsg	58
7.5.4.4	labelPassword	58
7.5.4.5	labelTitle	58

7.5.4.6	labelUsername	58
7.5.4.7	m_createUserDialog	59
7.5.4.8	m_dbmanager	59
7.5.4.9	m_loginHandler	59
7.5.4.10	m_rootPanel	59
7.5.4.11	passwordField	59
7.5.4.12	textFieldUsername	60
7.6	com.activitytracker.MainWindow Class Reference	61
7.6.1	Detailed Description	62
7.6.2	Constructor & Destructor Documentation	62
7.6.2.1	MainWindow()	63
7.6.3	Member Function Documentation	63
7.6.3.1	populateTable() [1/2]	63
7.6.3.2	populateTable() [2/2]	64
7.6.3.3	rootPanel()	64
7.6.3.4	setupActionListeners()	65
7.6.3.5	setupUI()	66
7.6.4	Member Data Documentation	66
7.6.4.1	buttonGo	67
7.6.4.2	buttonImportData	67
7.6.4.3	contentPanel	67
7.6.4.4	labelHeyThere	67
7.6.4.5	labelProfileIcon	67
7.6.4.6	m_dbManager	68
7.6.4.7	m_rootPanel	68

7.6.4.8	m_tableModel	68
7.6.4.9	m_user	68
7.6.4.10	panelMyActivity	68
7.6.4.11	tableMyActivity	69
7.6.4.12	textAreaStats	69
7.6.4.13	textFieldEndDate	69
7.6.4.14	textFieldStartDate	69
7.6.4.15	topPanel	69
7.7	com.activitytracker.Run Class Reference	70
7.7.1	Detailed Description	71
7.7.2	Constructor & Destructor Documentation	72
7.7.2.1	Run()	72
7.7.3	Member Function Documentation	72
7.7.3.1	bulkImport()	72
7.7.3.2	getAltitudeAscended()	73
7.7.3.3	getAltitudeDescended()	74
7.7.3.4	getDistance()	74
7.7.3.5	getDuration()	75
7.7.3.6	getID()	75
7.7.3.7	getRunDate()	76
7.7.3.8	getRuns()	76
7.7.3.9	getSpeed()	77
7.7.3.10	newRunDataPoint()	78
7.7.4	Member Data Documentation	79
7.7.4.1	altitudeAscended	79

7.7.4.2	altitudeDescended	80
7.7.4.3	caloriesBurned	80
7.7.4.4	dbManager	80
7.7.4.5	distance	81
7.7.4.6	duration	81
7.7.4.7	id	81
7.7.4.8	runDate	81
7.7.4.9	speed	82
7.8	com.activitytracker.RunAttribute Enum Reference	82
7.8.1	Detailed Description	83
7.8.2	Member Data Documentation	83
7.8.2.1	ALTITUDE_ASCENDED	83
7.8.2.2	ALTITUDE_DESCENDED	83
7.8.2.3	DISTANCE	84
7.8.2.4	DURATION	84
7.8.2.5	SPEED	84
7.9	com.activitytracker.RunStats Class Reference	85
7.9.1	Detailed Description	86
7.9.2	Constructor & Destructor Documentation	87
7.9.2.1	RunStats() [1/2]	87
7.9.2.2	RunStats() [2/2]	87
7.9.3	Member Function Documentation	88
7.9.3.1	addRun()	88
7.9.3.2	compute()	88
7.9.3.3	computeAll()	90

7.9.3.4	getMeanAltitudeAscended()	90
7.9.3.5	getMeanAltitudeDescended()	90
7.9.3.6	getMeanDistance()	91
7.9.3.7	getMeanDuration()	91
7.9.3.8	getMeanSpeed()	91
7.9.3.9	getTotalAltitudeAscended()	92
7.9.3.10	getTotalAltitudeDescended()	92
7.9.3.11	getTotalDistance()	92
7.9.3.12	isEmpty()	93
7.9.4	Member Data Documentation	93
7.9.4.1	meanAltitudeAscended	93
7.9.4.2	meanAltitudeDescended	93
7.9.4.3	meanDistance	94
7.9.4.4	meanDuration	94
7.9.4.5	meanSpeed	94
7.9.4.6	runs	94
7.9.4.7	totalAltitudeAscended	95
7.9.4.8	totalAltitudeDescended	95
7.9.4.9	totalDistance	95
7.10	com.activitytracker.SecureString Class Reference	96
7.10.1	Detailed Description	97
7.10.2	Constructor & Destructor Documentation	97
7.10.2.1	SecureString() [1/2]	97
7.10.2.2	SecureString() [2/2]	98
7.10.3	Member Function Documentation	98

7.10.3.1 equalString()	98
7.10.3.2 generateSalt()	99
7.10.3.3 generateSecureString()	100
7.10.3.4 getSalt()	100
7.10.3.5 toString()	101
7.10.4 Member Data Documentation	101
7.10.4.1 salt	101
7.10.4.2 secureString	102
7.11 com.activitytracker.User.Sex Enum Reference	102
7.11.1 Detailed Description	102
7.11.2 Member Data Documentation	103
7.11.2.1 FEMALE	103
7.11.2.2 MALE	103
7.12 com.activitytracker.User Class Reference	104
7.12.1 Detailed Description	105
7.12.2 Constructor & Destructor Documentation	105
7.12.2.1 User()	106
7.12.3 Member Function Documentation	106
7.12.3.1 createUser()	107
7.12.3.2 getDateOfBirth()	108
7.12.3.3 getEmailAddress()	108
7.12.3.4 getHeight()	108
7.12.3.5 getID()	109
7.12.3.6 getLastRID()	109
7.12.3.7 getName()	109

7.12.3.8 getSex()	109
7.12.3.9 getWeight()	110
7.12.3.10 getLastRID()	110
7.12.4 Member Data Documentation	110
7.12.4.1 dateOfBirth	110
7.12.4.2 dbManager	111
7.12.4.3 emailAddress	111
7.12.4.4 height	111
7.12.4.5 id	111
7.12.4.6 name	112
7.12.4.7 sex	112
7.12.4.8 weight	112
7.13 com.activitytracker.UserAttribute Enum Reference	113
7.13.1 Detailed Description	113
7.13.2 Member Data Documentation	114
7.13.2.1 DATE_OF_BIRTH	114
7.13.2.2 EMAIL_ADDRESS	114
7.13.2.3 HEIGHT	114
7.13.2.4 ID	115
7.13.2.5 NAME	115
7.13.2.6 PASSWORD	115
7.13.2.7 SALT	115
7.13.2.8 SEX	116
7.13.2.9 WEIGHT	116

8 File Documentation	117
8.1 app/src/com/activitytracker/ActivityTracker.java File Reference	117
8.2 app/src/com/activitytracker/CreateUserWindow.java File Reference	117
8.3 app/src/com/activitytracker/DBManager.java File Reference	118
8.4 app/src/com/activitytracker/Iteration3Test.java File Reference	118
8.5 app/src/com/activitytracker/LoginWindow.java File Reference	118
8.6 app/src/com/activitytracker/MainWindow.java File Reference	118
8.7 app/src/com/activitytracker/Run.java File Reference	119
8.8 app/src/com/activitytracker/RunAttribute.java File Reference	119
8.9 app/src/com/activitytracker/RunStats.java File Reference	119
8.10 app/src/com/activitytracker/SecureString.java File Reference	120
8.11 app/src/com/activitytracker/User.java File Reference	120
8.12 app/src/com/activitytracker/UserAttribute.java File Reference	120
Index	121

Chapter 1

COMP-2005 Activity Logger Documentation

This website contains documentation for all source code contained in the *Activity Logger* application. Class and method documentation may be accessed in HTML format using the left-hand side navigation bar, or the search box at the top right-hand side of the page.

For offline viewing, a precompiled PDF of this documentation has been made available [here](#) Note, however, that this document does *not* contain the full source code which is included in formatted HTML on this website.

More detailed information about contributions, repository branches, and commit history is available by browsing the [GitHub repository](#) for this project.

Chapter 2

Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

com	11
com.activitytracker	11

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

com.activitytracker.ActivityTracker	13
com.activitytracker.DBManager	23
com.activitytracker.Iteration3Test	48
JDialog	
com.activitytracker.CreateUserWindow	15
JFrame	
com.activitytracker.LoginWindow	52
com.activitytracker.MainWindow	61
com.activitytracker.Run	70
com.activitytracker.RunAttribute	82
com.activitytracker.RunStats	85
com.activitytracker.SecureString	96
com.activitytracker.User.Sex	102
com.activitytracker.User	104
com.activitytracker.UserAttribute	113

Chapter 4

Class Index

4.1 Class List

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

com.activitytracker.ActivityTracker	13
com.activitytracker.CreateUserWindow	15
com.activitytracker.DBManager	23
com.activitytracker.Iteration3Test	48
com.activitytracker.LoginWindow	52
com.activitytracker.MainWindow	61
com.activitytracker.Run	70
com.activitytracker.RunAttribute	82
com.activitytracker.RunStats	85
com.activitytracker.SecureString	96
com.activitytracker.User.Sex	102
com.activitytracker.User	104
com.activitytracker.UserAttribute	113

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

app/src/com/activitytracker/ActivityTracker.java	117
app/src/com/activitytracker/CreateUserWindow.java	117
app/src/com/activitytracker/DBManager.java	118
app/src/com/activitytracker/Iteration3Test.java	118
app/src/com/activitytracker/LoginWindow.java	118
app/src/com/activitytracker/MainWindow.java	118
app/src/com/activitytracker/Run.java	119
app/src/com/activitytracker/RunAttribute.java	119
app/src/com/activitytracker/RunStats.java	119
app/src/com/activitytracker/SecureString.java	120
app/src/com/activitytracker/User.java	120
app/src/com/activitytracker/UserAttribute.java	120

Chapter 6

Namespace Documentation

6.1 Package com

Packages

- package [activitytracker](#)

6.2 Package com.activitytracker

Classes

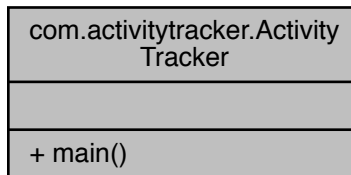
- class [ActivityTracker](#)
- class [CreateUserWindow](#)
- class [DBManager](#)
- class [Iteration3Test](#)
- class [LoginWindow](#)
- class [MainWindow](#)
- class [Run](#)
- enum [RunAttribute](#)
- class [RunStats](#)
- class [SecureString](#)
- class [User](#)
- enum [UserAttribute](#)

Chapter 7

Class Documentation

7.1 com.activitytracker.ActivityTracker Class Reference

Collaboration diagram for com.activitytracker.ActivityTracker:



Static Public Member Functions

- static void `main` (final String[] args)

7.1.1 Detailed Description

The main program class.

Definition at line 28 of file `ActivityTracker.java`.

7.1.2 Member Function Documentation

7.1.2.1 main()

```
static void com.activitytracker.ActivityTracker.main (
    final String [] args ) [static]
```

The main program entry point.

Definition at line 33 of file ActivityTracker.java.

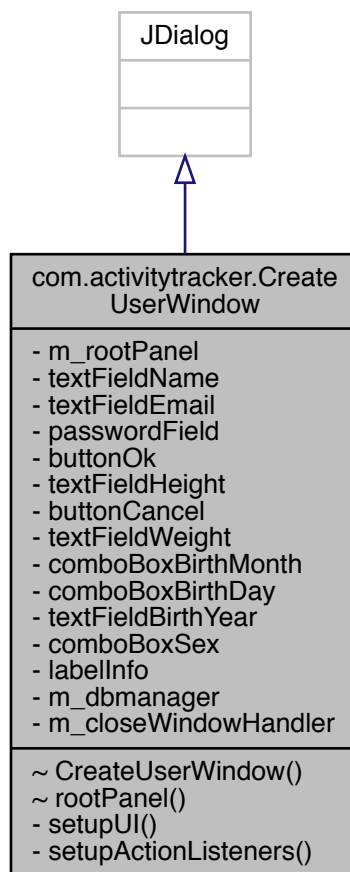
```
33         {
34
35         // Create singleton instance of DBManager
36         DBManager dbManager = new DBManager();
37         if (!dbManager.init("data.db")) {
38             System.err.println("Failed to initialize DBManager");
39             System.exit(1);
40         }
41
42         // Set Look and Feel
43         try {
44             UIManager.setLookAndFeel(new MaterialLookAndFeel());
45         }
46         catch (final UnsupportedLookAndFeelException e) {
47             e.printStackTrace();
48         }
49         // Get desktop resolution of default monitor (in case of multi-monitor setups)
50         final GraphicsDevice gd = GraphicsEnvironment.getLocalGraphicsEnvironment().getDefaultScreenDevice(
51     );
52         final JFrame frame = new JFrame("Activity Logger");
53
54         final String logoPath = "./assets/logo.png";
55         ImageIcon imgIcon = new ImageIcon(ActivityTracker.class.getResource(logoPath));
56         frame.setIconImage(imgIcon.getImage());
57         frame.setContentPane(new LoginWindow((final User user) -> {
58             frame.setContentPane(new MainWindow(dbManager, user).rootPanel());
59             frame.validate();
60             frame.repaint();
61         }, dbManager).rootPanel());
62         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
63         frame.pack();
64
65         // Set window size to be 1/2 of screen dimensions
66         frame.setSize(gd.getDisplayMode().getWidth() * 2/3 , gd.getDisplayMode().getHeight() * 2/3);
67         frame.setLocationRelativeTo(null); // Center window
68         frame.setVisible(true);
69     }
```

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

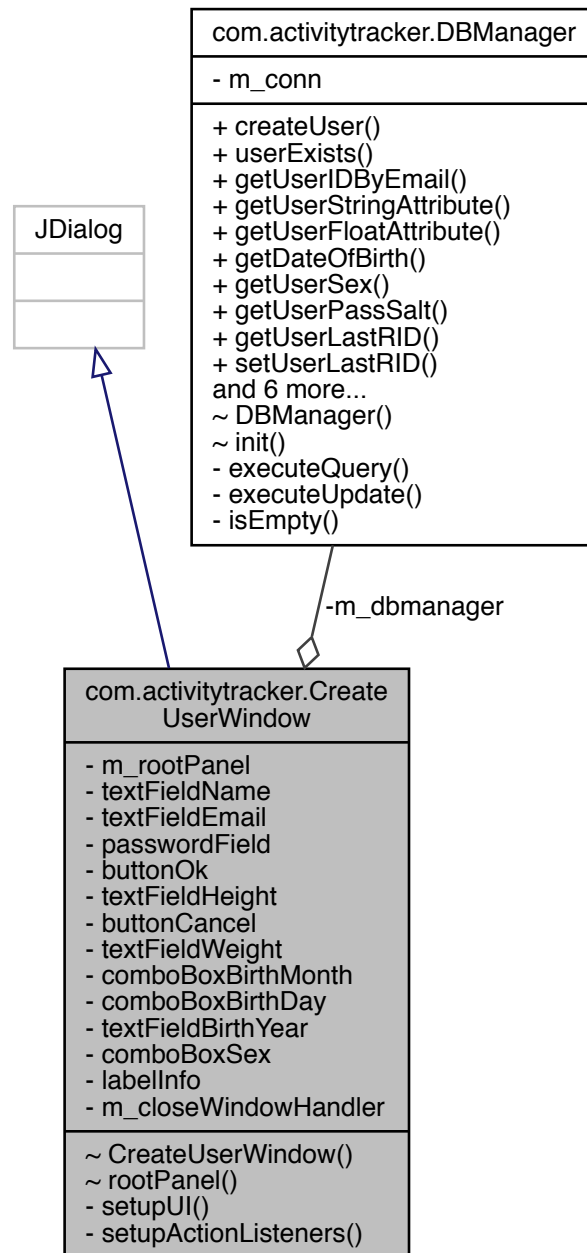
- [app/src/com/activitytracker/ActivityTracker.java](#)

7.2 com.activitytracker.CreateUserWindow Class Reference

Inheritance diagram for com.activitytracker.CreateUserWindow:



Collaboration diagram for com.activitytracker.CreateUserWindow:



Package Functions

- [CreateUserWindow](#) ([DBManager](#) dbmanager, java.util.function.Consumer< Void > closeWindowHandler)
- [JPanel](#) [rootPanel](#) ()

Private Member Functions

- void [setupUI](#) ()
- void [setupActionListeners](#) ()

Private Attributes

- [JPanel](#) [m_rootPanel](#)
- [JTextField](#) [textFieldName](#)
- [JTextField](#) [textFieldEmail](#)
- [JPasswordField](#) [passwordField](#)
- [JButton](#) [buttonOk](#)
- [JTextField](#) [textFieldHeight](#)
- [JButton](#) [buttonCancel](#)
- [JTextField](#) [textFieldWeight](#)
- [JComboBox](#) [comboBoxBirthMonth](#)
- [JComboBox](#) [comboBoxBirthDay](#)
- [JTextField](#) [textFieldBirthYear](#)
- [JComboBox](#) [comboBoxSex](#)
- [JLabel](#) [labelInfo](#)
- [DBManager](#) [m_dbmanager](#) = null
- java.util.function.Consumer< Void > [m_closeWindowHandler](#)

7.2.1 Detailed Description

Definition at line 12 of file CreateUserWindow.java.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 CreateUserWindow()

```
com.activitytracker.CreateUserWindow.CreateUserWindow (
    DBManager dbmanager,
    java.util.function.Consumer< Void > closeWindowHandler ) [package]
```

Definition at line 31 of file CreateUserWindow.java.

```
31                                     {
32     m_dbmanager = dbmanager;
33     m_closeWindowHandler = closeWindowHandler;
34
35     setupUI();
36     setupActionListeners();
37 }
```

7.2.3 Member Function Documentation

7.2.3.1 rootPanel()

```
JPanel com.activitytracker.CreateUserWindow.rootPanel ( ) [package]
```

Definition at line 88 of file CreateUserWindow.java.

```
88     {
89     return m_rootPanel;
90 }
```


7.2.3.2 setupActionListeners()

void com.activitytracker.CreateUserWindow.setupActionListeners () [private]

Definition at line 46 of file CreateUserWindow.java.

```

46         {
47             buttonOk.addActionListener(new ActionListener() {
48                 @Override
49                 public void actionPerformed(ActionEvent e) {
50
51                     if (textFieldName.getText().isEmpty() ||
52                         textFieldEmail.getText().isEmpty() ||
53                         passwordField.getPassword().length == 0 ||
54                         textFieldHeight.getText().isEmpty() ||
55                         textFieldWeight.getText().isEmpty() ||
56                         textFieldBirthYear.getText().isEmpty()) {
57
58                         m_closeWindowHandler.accept(null);
59                         return;
60                     }
61
62                     try {
63                         User.createUser(m_dbmanager,
64                                     textFieldName.getText(),
65                                     textFieldEmail.getText(),
66                                     new Date(Integer.parseInt(textFieldBirthYear.getText()),
67                                     comboBoxBirthMonth.getSelectedIndex() + 1, comboBoxBirthDay.
68                                     getSelectedIndex() + 1),
69                                     (comboBoxSex.getSelectedIndex() == 0 ? User.Sex.MALE : User.Sex.
70                                     FEMALE),
71                                     Float.parseFloat(textFieldHeight.getText()),
72                                     Float.parseFloat(textFieldWeight.getText()),
73                                     String.valueOf(passwordField.getPassword()));
74                     } catch (final AssertionError ex) {
75                         labelInfo.setVisible(true);
76                         labelInfo.setText("User already exists.");
77                     }
78                     m_closeWindowHandler.accept(null);
79                 }
80             });
81             buttonCancel.addActionListener(new ActionListener() {
82                 @Override
83                 public void actionPerformed(ActionEvent e) {
84                     m_closeWindowHandler.accept(null);
85                 }
86             });

```

7.2.3.3 setupUI()

void com.activitytracker.CreateUserWindow.setupUI () [private]

Definition at line 39 of file CreateUserWindow.java.

```
39         {
40             MaterialUIMovement.add(buttonCancel, MaterialColors.GRAY_100);
41             MaterialUIMovement.add(buttonOk, MaterialColors.GRAY_100);
42
43             labelInfo.setVisible(false);
44         }
```

7.2.4 Member Data Documentation

7.2.4.1 buttonCancel

JBButton com.activitytracker.CreateUserWindow.buttonCancel [private]

Definition at line 19 of file CreateUserWindow.java.

7.2.4.2 buttonOk

JBButton com.activitytracker.CreateUserWindow.buttonOk [private]

Definition at line 17 of file CreateUserWindow.java.

7.2.4.3 comboBoxBirthDay

JComboBox com.activitytracker.CreateUserWindow.comboBoxBirthDay [private]

Definition at line 22 of file CreateUserWindow.java.

7.2.4.4 comboBoxBirthMonth

JComboBox com.activitytracker.CreateUserWindow.comboBoxBirthMonth [private]

Definition at line 21 of file CreateUserWindow.java.

7.2.4.5 comboBoxSex

JComboBox com.activitytracker.CreateUserWindow.comboBoxSex [private]

Definition at line 24 of file CreateUserWindow.java.

7.2.4.6 labelInfo

JLabel com.activitytracker.CreateUserWindow.labelInfo [private]

Definition at line 25 of file CreateUserWindow.java.

7.2.4.7 m_closeWindowHandler

java.util.function.Consumer<Void> com.activitytracker.CreateUserWindow.m_↔
closeWindowHandler [private]

Definition at line 29 of file CreateUserWindow.java.

7.2.4.8 m_dbmanager

DBManager com.activitytracker.CreateUserWindow.m_dbmanager = null [private]

Definition at line 27 of file CreateUserWindow.java.

7.2.4.9 m_rootPanel

JPanel com.activitytracker.CreateUserWindow.m_rootPanel [private]

Definition at line 13 of file CreateUserWindow.java.

7.2.4.10 passwordField

`JPasswordField com.activitytracker.CreateUserWindow.passwordField [private]`

Definition at line 16 of file CreateUserWindow.java.

7.2.4.11 textFieldBirthYear

`TextField com.activitytracker.CreateUserWindow.textFieldBirthYear [private]`

Definition at line 23 of file CreateUserWindow.java.

7.2.4.12 textFieldEmail

`TextField com.activitytracker.CreateUserWindow.textFieldEmail [private]`

Definition at line 15 of file CreateUserWindow.java.

7.2.4.13 textFieldHeight

`TextField com.activitytracker.CreateUserWindow.textFieldHeight [private]`

Definition at line 18 of file CreateUserWindow.java.

7.2.4.14 textFieldName

`TextField com.activitytracker.CreateUserWindow.textFieldName [private]`

Definition at line 14 of file CreateUserWindow.java.

7.2.4.15 textFieldWeight

TextField com.activitytracker.CreateUserWindow.textFieldWeight [private]

Definition at line 20 of file CreateUserWindow.java.

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

- [app/src/com/activitytracker/CreateUserWindow.java](#)

7.3 com.activitytracker.DBManager Class Reference

Collaboration diagram for com.activitytracker.DBManager:

com.activitytracker.DBManager
- m_conn
+ createUser() + userExists() + getUserIDByEmail() + getUserStringAttribute() + getUserFloatAttribute() + getDateOfBirth() + getUserSex() + getUserPassSalt() + getUserLastRID() + setUserLastRID() and 6 more... ~ DBManager() ~ init() - executeQuery() - executeUpdate() - isEmpty()

Public Member Functions

- void [createUser](#) (final String name, final String emailAddress, final java.util.Date dateOfBirth, final User.Sex sex, final float height, final float weight, final [SecureString](#) securePassword) throws AssertionError

- boolean [userExists](#) (final String emailAddress)
- int [getUserIDByEmail](#) (final String emailAddress)
- String [getUserStringAttribute](#) (final [UserAttribute](#) attribute, final int id)
- float [getUserFloatAttribute](#) (final [UserAttribute](#) attribute, final int id)
- Date [getDateOfBirth](#) (final int id)
- User.Sex [getUserSex](#) (final int id)
- byte [] [getUserPassSalt](#) (final int id)
- int [getUserLastRID](#) (final int id)
- void [setUserLastRID](#) (final int id, final int lastRID)
- int [newRun](#) (final int userID, final java.util.Date date, final float duration, final float distance, final float altitudeAscended, final float altitudeDescended)
- void [setRun](#) (final int rID, final float duration, final float distance, final float altitudeAscended, final float altitudeDescended)
- float [getRunFloatAttribute](#) (final [RunAttribute](#) attribute, final int rID)
- java.util.Date [getRunDate](#) (final int runID)
- boolean [runExists](#) (final int rID)
- Vector< Integer > [getRuns](#) (final int userID, final java.util.Date startDate, final java.util.Date endDate)

Package Functions

- [DBManager](#) ()
- boolean [init](#) (final String dbURL)

Private Member Functions

- ResultSet [executeQuery](#) (final String sqlQuery)
- boolean [executeUpdate](#) (final String sqlQuery)
- boolean [isEmpty](#) ()

Private Attributes

- Connection [m_conn](#) = null

7.3.1 Detailed Description

Singleton class for the database. All classes and methods that interact with the database will use a method in this class.

Many times we are faced with the "chicken and egg" problem where we wish to create an object that is populated with information from the database. So the question one faces is, "does the object's constructor query the database (through the [DBManager](#) class, of course) for each attribute of the object that it wishes to retrieve, or do we directly interact with a [DBManager](#) method which will then return a [User](#) or [Run](#) object, for example?" We have decided to use the former methodology, with [DBManager](#) methods being as general as possible, and often accepting enum types which then are put into a switch to create the specific SQL query we wish to execute. This works best when all data returned is of the same data type (for example, the Workout class will have three float attributes at the time of writing so we use one method with return type of float for returning Workout attributes). This does not work as well when the object requires data of multiple types — for example, the [User](#) class. In this case, we have split the [DBManager](#) methods into a single method for each attribute being returned.

Polymorphism could theoretically be used here to simply have a return type of Object, however this is not flexible and requires casting *all* returned data to the correct type in the invoking method.

Definition at line 30 of file DBManager.java.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 DBManager()

com.activitytracker.DBManager.DBManager () [package]

Creates a new [DBManager](#) object.

This should only be called once, from the main program, as [DBManager](#) is meant to be a *singleton* class.

This constructor takes no parameters as verification of the SQLite database is done in the [init\(\)](#) method of this class, which returns information about whether the initialization was successful or not.

Definition at line 47 of file DBManager.java.

```
47         {  
48     }
```

7.3.3 Member Function Documentation

7.3.3.1 createUser()

```
void com.activitytracker.DBManager.createUser (
    final String name,
    final String emailAddress,
    final java.util.Date dateofBirth,
    final User.Sex sex,
    final float height,
    final float weight,
    final SecureString securePassword ) throws AssertionError
```

Adds a row for a user to the Users table in the SQLite database for the app.

Requires that the database tables exist and are in the correct format. If the user exists in the database this method raises an AssertionError exception.

Parameters

<i>name</i>	User's name.
<i>emailAddress</i>	User's email address; used to authenticate.
<i>dateofBirth</i>	The date the user was born.
<i>sex</i>	The user's sex; is either User.Sex.MALE or User.Sex.FEMALE .
<i>height</i>	Floating point number of the user's height in metres.
<i>weight</i>	Floating point number of the user's weight in kilograms.
<i>securePassword</i>	A SecureString object containing the user's password, encrypted.

Definition at line 64 of file DBManager.java.

```
66                                                                 {
67
68         if (!userExists(emailAddress)) {
69             String sqlQuery = "INSERT INTO Users (" +
70                 "email_address, " +
71                 "name, " +
72                 "date_of_birth, " +
73                 "sex, " +
74                 "height, " +
75                 "weight," +
76                 "password_hash," +
77                 "password_salt," +
78                 "created_at" +
79                 ") VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";
```



```

80         byte sexByte = sex.equals(User.Sex.MALE) ? (byte) 1 : (byte) 0;
81         java.sql.Date currentTime = new java.sql.Date(System.currentTimeMillis());
82
83         try {
84             PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
85             stmt.setString(1, emailAddress);
86             stmt.setString(2, name);
87             stmt.setLong(3, dateOfBirth.getTime());
88             stmt.setByte(4, sexByte);
89             stmt.setFloat(5, height);
90             stmt.setFloat(6, weight);
91             stmt.setString(7, securePassword.toString());
92             stmt.setBytes(8, securePassword.getSalt());
93             stmt.setDate(9, currentTime);
94
95             if (stmt.executeUpdate() != 1) {
96                 System.err.println("User not added to database.");
97             }
98
99             stmt.close();
100         }
101         catch (final SQLException e) {
102             System.err.println(e.getMessage());
103         }
104     }
105     else {
106         throw new AssertionError("User with email address '" + emailAddress + "' already exists.");
107     }
108 }

```

7.3.3.2 executeQuery()

```

ResultSet com.activitytracker.DBManager.executeQuery (
    final String sqlQuery ) [private]

```

A wrapper method for processing *safe* SQL queries.

By safe we mean that the SQL query string is entirely hard-coded in the program source code. In other words, no user input is added. This is an important distinction as the former may leave the application vulnerable to SQL injection.

In such cases, a SQL PreparedStatement should be used.

Parameters

<i>sqlQuery</i>	The SQL code to be executed. Must be a <i>SELECT</i> statement.
-----------------	---

Returns

This method returns a ResultSet containing the returned row(s) and/or column(s) of the SQL query that was executed.

Definition at line 732 of file DBManager.java.

```
732                                     {
733     ResultSet res = null;
734
735     try {
736         Statement stmt = m_conn.createStatement();
737         res = stmt.executeQuery(sqlQuery);
738         stmt.close();
739     }
740     catch (final SQLException e) {
741         System.err.println(e.getMessage());
742     }
743
744     return res;
745 }
```

7.3.3.3 executeUpdate()

```
boolean com.activitytracker.DBManager.executeUpdate (
    final String sqlQuery ) [private]
```

A wrapper method for processing *safe* SQL queries.

By safe we mean that the SQL query string is entirely hard-coded in the program source code. In other words, no user input is added. This is an important distinction as the former may leave the application vulnerable to SQL injection.

In such cases, a SQL PreparedStatement should be used.

Parameters

<i>sqlQuery</i>	The SQL code to be executed. Must be an <i>INSERT</i> or <i>UPDATE</i> statement.
-----------------	---

Returns

This method returns a boolean indicating if the query was successful.

Definition at line 760 of file DBManager.java.

```
760                                     {
761     try {
762         Statement stmt = m_conn.createStatement();
763         stmt.executeUpdate(sqlQuery);
764         stmt.close();
765     }
```

```
765     }
766     catch (final SQLException e) {
767         System.err.println(e.getMessage());
768         return false;
769     }
770
771     return true;
772 }
```

7.3.3.4 getDateOfBirth()

Date com.activitytracker.DBManager.getDateOfBirth (
 final int id)

Retrieves the user's date of birth (DOB) from the database.

At the time of writing, this method is only being used in the [User](#) constructor.

Parameters

<i>id</i>	Unique ID used to associate information in the database to this user.
-----------	---

Returns

This method returns a Date object containing the user's DOB (i.e., year, month, day).

Definition at line 294 of file DBManager.java.

```
294                                     {
295     Date DOB;
296     java.sql.Date DOBResult;
297     ResultSet res;
298     String sqlQuery = "SELECT date_of_birth FROM Users WHERE id=?";
299     try {
300         PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
301         stmt.setInt(1, id);
302         res = stmt.executeQuery();
303         DOBResult = res.getDate("date_of_birth");
304
305         stmt.close();
306     }
307     catch (final SQLException e) {
308         System.err.println(e.getMessage());
309         return null;
310     }
311     DOB = new Date(DOBResult.getYear(), DOBResult.getMonth(), DOBResult.getDay());
312
313     return DOB;
314 }
```

7.3.3.5 getRunDate()

```
java.util.Date com.activitytracker.DBManager.getRunDate (
    final int runID )
```

Retrieves a run's date from the database using its unique ID

Parameters

<i>runID</i>	Unique ID corresponding to the row in the Runs table that we wish to query.
--------------	---

Definition at line 627 of file DBManager.java.

```
627                                     {
628     ResultSet res;
629     long fromEpoch;
630     java.util.Date date = null;
631     try {
632         PreparedStatement stmt = m_conn.prepareStatement("SELECT date FROM Runs WHERE id=?");
633         stmt.setInt(1, runID);
634         res = stmt.executeQuery();
635         fromEpoch = res.getLong("date");
636         date = new java.util.Date(fromEpoch);
637     }
638     catch (final SQLException e) {
639         System.err.println(e.getMessage());
640     }
641
642     return date;
643 }
```

7.3.3.6 getRunFloatAttribute()

```
float com.activitytracker.DBManager.getRunFloatAttribute (
    final RunAttribute attribute,
    final int rID )
```

Retrieves a run's attribute as a floating point number, where applicable, from the database.

This method accepts a [RunAttribute](#) enumeration type to specify what attribute it is returning from the database. Only certain attributes are accepted by this method, namely those that are stored as real values. Attributes stored as other data types should use the appropriate accessor method.

Parameters

<i>attribute</i>	<p>The attribute that the method is supposed to query the DB for and return the value of. Note that only certain RunAttribute types are supported in this method.</p> <ul style="list-style-type: none"> • When <i>attribute</i> is RunAttribute.DURATION, the run's duration is returned. • When <i>attribute</i> is RunAttribute.DISTANCE, the run's cumulative distance is returned in metres. • When <i>attribute</i> is RunAttribute.ALTITUDE_ASCENDED, the run's cumulative altitude climbed is returned in metres • When <i>attribute</i> is RunAttribute.ALTITUDE_DESCENDED, the run's cumulative altitude descended is returned in metres
<i>rID</i>	<p>Unique ID corresponding to the row in the Runs table that we wish to query. If such an ID does not exist, <i>0.0f</i> will be returned.</p>

Returns

This method returns a float containing run attribute as specified by the *attribute* parameter.

Definition at line 578 of file DBManager.java.

```

578                                     {
579         ResultSet res;
580         PreparedStatement stmt;
581         String sqlQuery, columnLabel;
582         float attrVal = 0.0f;
583         switch (attribute) {
584             case DURATION:
585                 columnLabel = "duration";
586                 sqlQuery = "SELECT " + columnLabel + " FROM Runs WHERE id=?";
587                 break;
588             case DISTANCE:
589                 columnLabel = "distance";
590                 sqlQuery = "SELECT " + columnLabel + " FROM Runs WHERE id=?";
591                 break;
592             case ALTITUDE_ASCENDED:
593                 columnLabel = "altitude_ascended";
594                 sqlQuery = "SELECT " + columnLabel + " FROM Runs WHERE id=?";
595                 break;
596             case ALTITUDE_DESCENDED:
597                 columnLabel = "altitude_descended";
598                 sqlQuery = "SELECT " + columnLabel + " FROM Runs WHERE id=?";
599                 break;
600             default:
601                 return attrVal;
602         }
603         if (runExists(rID)) {
604             try {
605                 stmt = m_conn.prepareStatement(sqlQuery);
606                 stmt.setInt(1, rID);
607                 res = stmt.executeQuery();

```

```

608         attrVal = res.getFloat(columnLabel);
609     }
610     catch (final SQLException e) {
611         System.err.println(e.getMessage());
612     }
613 }
614 else {
615     System.err.println("Run " + Integer.toString(rID) + " does not exist. Cannot get " +
columnLabel + ".");
616 }
617
618     return attrVal;
619 }
620 }

```

7.3.3.7 getRuns()

```

Vector<Integer> com.activitytracker.DBManager.getRuns (
    final int userID,
    final java.util.Date startDate,
    final java.util.Date endDate )

```

Queries the database for all runs by a user with user ID *userID* between *startDate* and *endDate*.

Parameters

<i>userID</i>	The ID of the user whose runs we wish to retrieve.
<i>startDate</i>	The lower bound of the interval we wish to retrieve runs for.
<i>endDate</i>	The upper bound of the interval we wish to retrieve runs for.

Returns

Returns a vector containing run IDs for each run that meets the search criteria.

Definition at line 691 of file DBManager.java.

```

691     {
692     ResultSet res;
693     Vector<Integer> runs = new Vector<>();
694     try {
695         PreparedStatement stmt = m_conn.prepareStatement(
696             "SELECT id FROM Runs WHERE user_id=? AND date BETWEEN ? AND ?");
697         stmt.setInt(1, userID);
698         stmt.setLong(2, startDate.getTime());
699         stmt.setLong(3, endDate.getTime());
700     }

```

```

701         res = stmt.executeQuery();
702
703         if (res.isClosed())
704             System.err.println("Result set closed; cannot get any data?");
705
706         while (res.next()) {
707             runs.add(res.getInt("id"));
708         }
709         stmt.close();
710     }
711     catch (final SQLException e) {
712         System.err.println(e.getMessage());
713     }
714
715     return runs;
716 }

```

7.3.3.8 getUserFloatAttribute()

```
float com.activitytracker.DBManager.getUserFloatAttribute (
    final UserAttribute attribute,
    final int id )
```

Retrieves a user's attribute in floating point format, when applicable, from the database's Users table.

This method accepts a [UserAttribute](#) enumeration type to specify what attribute it is returning from the database. Only certain attributes are accepted by this method, namely those that are stored as real values. Attributes stored as other data types should use the appropriate accessor method.

Parameters

<i>attribute</i>	<p>The attribute that the method is supposed to query the DB for and return the value of. Note that only certain UserAttribute types are supported in this method.</p> <ul style="list-style-type: none"> • When <i>attribute</i> is UserAttribute.WEIGHT, this method retrieves the user's weight from the database. • When <i>attribute</i> is UserAttribute.HEIGHT, this method retrieves the user's height from the database.
<i>id</i>	Unique ID used to associate information in the database to this user.

Returns

Returns a floating point number corresponding to the [UserAttribute](#) passed to the method, for the user specified by *id*.

Definition at line 253 of file DBManager.java.

```

253                                     {
254         float attrVal;
255         ResultSet res;
256         String sqlQuery, columnLabel;
257         switch (attribute) {
258             case WEIGHT:
259                 columnLabel = "weight";
260                 sqlQuery = "SELECT " + columnLabel + " FROM Users WHERE id=?";
261                 break;
262             case HEIGHT:
263                 columnLabel = "height";
264                 sqlQuery = "SELECT " + columnLabel + " FROM Users WHERE id=?";
265                 break;
266             default:
267                 throw new AssertionError("Incorrect UserAttribute enumeration type passed to method.");
268         }
269         try {
270             PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
271             stmt.setInt(1, id);
272             res = stmt.executeQuery();
273             attrVal = res.getFloat(columnLabel);
274
275             stmt.close();
276         }
277         catch (final SQLException e) {
278             System.err.println(e.getMessage());
279             return 0.0f;
280         }
281         return attrVal;
282     }
283 }
```

7.3.3.9 getUserIDByEmail()

```
int com.activitytracker.DBManager.getUserIDByEmail (
    final String emailAddress )
```

As we are using the user's email address as their identifying attribute, they will supply this when they log in. Hence, as the database relates everything to the user's unique ID, we must retrieve this ID given the email address.

The logic behind this method relies on the database Users table structure making *email_address* a unique field.

Parameters

<i>emailAddress</i>	The user's email address with which they authenticate.
---------------------	--

Returns

This method returns a unique integer corresponding to the row in the database's Users table that stores user information for user with email address *emailAddress*.

Definition at line 151 of file DBManager.java.

```
151                                     {
152         int id = 0;
153         ResultSet res;
154         String sqlQuery = "SELECT id FROM Users WHERE 'email_address'=?";
155         try {
156             PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
157             stmt.setString(1, emailAddress);
158             res = stmt.executeQuery();
159             id = res.getInt("id");
160
161             stmt.close();
162         }
163         catch (final SQLException e) {
164             System.err.println(e.getMessage());
165         }
166         return id;
167     }
168 }
```

7.3.3.10 getUserLastRID()

```
int com.activitytracker.DBManager.getUserLastRID (
    final int id )
```

Retrieves the last workout ID that the user added as an integer from the database.

This is used because of the format in which the data is supplied. As the only way to denote a new workout is by receiving (0, 0, 0) in the input file, if the input is *not* (0, 0, 0), we need to update the previously added workout with the latest line. Hence we need some way of storing an identifier for this workout. As this is unique to each user, we have chosen to store this in the Users table of the database.

Parameters

<i>id</i>	Unique ID used to associate information in the database to this user.
-----------	---

Returns

An integer corresponding to the last row in the Workouts table that the user created.

Definition at line 391 of file DBManager.java.

```
391                                     {
392         int rID = 0;
393         ResultSet res;
394         String columnLabel = "last_run";
395         String sqlQuery = "SELECT " + columnLabel + " FROM Users WHERE id=?";
396         try {
397             PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
398             stmt.setInt(1, id);
399             res = stmt.executeQuery();
400             rID = res.getInt(columnLabel);
401             stmt.close();
402         }
403         catch (final SQLException e) {
404             System.err.println(e.getMessage());
405         }
406         return rID;
407     }
408 }
```

7.3.3.11 getUserPassSalt()

```
byte [] com.activitytracker.DBManager.getUserPassSalt (
    final int id )
```

Retrieves a byte array containing the salt used to encrypt the user's password from the database.

This is necessary because to compare a candidate password supplied by a user to a known (encrypted) password stored in the database, we must encrypt the new candidate password using the same salt as was originally used.

Parameters

<i>id</i>	Unique ID used to associate information in the database to this user.
-----------	---

Returns

This method returns a byte array containing the user's password encryption salt.

Definition at line 362 of file DBManager.java.

```
362                                     {
363         byte[] passSalt;
364         ResultSet res;
365         String sqlQuery = "SELECT password_salt FROM Users WHERE id=?";
```

```

366         try {
367             PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
368             stmt.setInt(1, id);
369             res = stmt.executeQuery();
370             passSalt = res.getBytes("password_salt");
371             stmt.close();
372         }
373         catch (final SQLException e) {
374             System.err.println(e.getMessage());
375             return null;
376         }
377         return passSalt;
378     }

```

7.3.3.12 getUserSex()

```
User.Sex com.activitytracker.DBManager.getUserSex (
    final int id )
```

Retrieves the user's gender from the database.

We have chosen to represent gender in the SQLite database with the data type BIT(1), where 1 denotes male and 0 denotes female. Hence, if the database contains 1 this method returns [User.Sex.MALE](#) and if the database contains 0 then this method returns [User.Sex.FEMALE](#).

At the time of writing, this method is only being used in the [User](#) constructor.

Parameters

<i>id</i>	Unique ID used to associate information in the database to this user.
-----------	---

Returns

This method returns a [User.Sex](#) enumeration type corresponding to the user's gender.

Definition at line 329 of file DBManager.java.

```

329         {
330             byte sex;
331             ResultSet res;
332             String sqlQuery = "SELECT sex FROM Users WHERE id=?";
333             try {
334                 PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
335                 stmt.setInt(1, id);
336                 res = stmt.executeQuery();
337                 sex = res.getBytes("sex");
338             }

```

```
339         stmt.close();
340     }
341     catch (final SQLException e) {
342         System.err.println(e.getMessage());
343         return null;
344     }
345
346     if (sex == (byte) 1)
347         return User.Sex.MALE;
348     else
349         return User.Sex.FEMALE;
350 }
```

7.3.3.13 getUserStringAttribute()

```
String com.activitytracker.DBManager.getUserStringAttribute (
    final UserAttribute attribute,
    final int id )
```

This method retrieves a string, varchar, text, or char field, when applicable, from the database's Users table.

This method accepts a [UserAttribute](#) enumeration type to specify what attribute it is returning from the database. Only certain attributes are accepted by this method, namely those that are stored as string-like values. Attributes stored as other data types should use the appropriate accessor method.

Parameters

<i>attribute</i>	<p>The attribute that the method is supposed to query the DB for and return the value of. Note that only certain UserAttribute types are supported in this method.</p> <ul style="list-style-type: none"> When <i>attribute</i> is UserAttribute.PASSWORD, this method retrieves the user's encrypted password from the database. Typically this will be used in the following sequence of calls: <ol style="list-style-type: none"> User attempts to authenticate with email and password Their unique ID is retrieved from the database using DBManager::getUserIDByEmail() Their ID is used to retrieve the hash of their password (i.e., this method is called) The returned string from this method is compared a SecureString generated from the candidate password supplied by the user when authenticating. When <i>attribute</i> is UserAttribute.NAME, this method retrieves the user's full name from the database (e.g., "John Doe"). When <i>attribute</i> is UserAttribute.EMAIL_ADDRESS, this method retrieves the user's email address from the database. Note that this is likely somewhat redundant as the user will always be required to authenticate by providing their email address and hence it will already be available to the User constructor, which is likely what is invoking this method.
<i>id</i>	Unique ID used to associate information in the database to this user.

Returns

This method returns a string containing attribute specified by the *attribute* parameter for the user specified by the *id* parameter.

Definition at line 199 of file DBManager.java.

```

199                                     {
200     String name;
201     ResultSet res;
202     String sqlQuery, columnLabel;
203     switch (attribute) {
204     case PASSWORD:
205         columnLabel = "password_hash";
206         sqlQuery = "SELECT " + columnLabel + " FROM Users WHERE id=?";
207         break;
208     case NAME:
209         columnLabel = "name";
210         sqlQuery = "SELECT " + columnLabel + " FROM Users WHERE id=?";

```

```

211         break;
212     case EMAIL_ADDRESS:
213         columnLabel = "email_address";
214         sqlQuery = "SELECT " + columnLabel + " FROM Users WHERE id=?";
215         break;
216     default:
217         throw new AssertionError("Incorrect UserAttribute enumeration type passed to method.");
218     }
219     try {
220         PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
221         stmt.setInt(1, id);
222         res = stmt.executeQuery();
223         name = res.getString(columnLabel);
224
225         stmt.close();
226     }
227     catch (final SQLException e) {
228         System.err.println(e.getMessage());
229         return null;
230     }
231
232     return name;
233 }

```

7.3.3.14 init()

```
boolean com.activitytracker.DBManager.init (
    final String dbURL ) [package]
```

Initializes a connection to the SQLite database.

As no work is done in the [DBManager\(\)](#) constructor, this method should be called immediately after creating the single instance of [DBManager](#) that the application is to use.

This method will attempt to connect to the database file specified by the *dbURL* parameter, creating the file and all required tables if it/they do not exist. You are encouraged to view the source code of this method for more information about the database schema used.

If all of the above is successful, the method returns True. Otherwise, False is returned.

Parameters

<i>dbURL</i>	A file system path to the SQLite database file.
--------------	---

Returns

This method returns True if the database can be initialized, or False otherwise.

Definition at line 814 of file DBManager.java.

```

814         {
815         try {
816             m_conn = DriverManager.getConnection("jdbc:sqlite:" + dbURL);
817         }
818         catch (final SQLException e) {
819             System.err.println(e.getMessage());
820             return false;
821         }
822         System.out.println("Opened database successfully.");
823
824         if (isEmpty()) {
825             System.out.println("Creating tables...");
826
827             // Create users table
828             String sqlQuery = "CREATE TABLE USERS (" +
829                 "    id                INTEGER PRIMARY KEY ASC AUTOINCREMENT NOT NULL," +
830                 "    email_address    STRING  NOT NULL UNIQUE ON CONFLICT FAIL," +
831                 "    name              STRING  NOT NULL," +
832                 "    date_of_birth     DATETIME NOT NULL," +
833                 "    sex               BIT(1)  NOT NULL," +
834                 "    height            REAL    NOT NULL," +
835                 "    weight            REAL    NOT NULL," +
836                 "    password_hash    STRING  NOT NULL," +
837                 "    password_salt    BLOB    NOT NULL," +
838                 "    last_run          INTEGER NOT NULL DEFAULT 0," +
839                 "    created_at        DATETIME NOT NULL" +
840                 ")";
841
842             if (!executeUpdate(sqlQuery)) {
843                 return false;
844             }
845
846             // Create workouts table
847             sqlQuery = "CREATE TABLE RUNS (" +
848                 "    id                INTEGER PRIMARY KEY ASC AUTOINCREMENT NOT NULL," +
849                 "    user_id           INTEGER NOT NULL REFERENCES USERS (id)," +
850                 "    date              DATETIME NOT NULL," +
851                 "    duration          REAL    NOT NULL," + // seconds
852                 "    distance          REAL    NOT NULL," + // metres
853                 "    altitude_ascended REAL    NOT NULL," + // metres
854                 "    altitude_descended REAL    NOT NULL" + // metres
855                 ")";
856
857             if (!executeUpdate(sqlQuery)) {
858                 return false;
859             }
860
861         }
862
863         return true;
864     }

```

7.3.3.15 isEmpty()

boolean com.activitytracker.DBManager.isEmpty () [private]

Returns a boolean value depending on whether or not the database is populated.

This is done by retrieving tables in the database and checking if this iterator has a next(). If not then there are no tables in the database and we consider it to be empty.

Returns

Returns True if there are tables in the database, False otherwise.

Definition at line 782 of file DBManager.java.

```

782             {
783
784         try {
785             final DatabaseMetaData dbmd = m_conn.getMetaData();
786             final String[] types = {"TABLE"};
787             final ResultSet rs = dbmd.getTables(null, null, "%", types);
788
789             return !rs.next();
790         }
791         catch (final SQLException e) {
792             System.err.println(e.getMessage());
793             return true;
794         }
795     }
796 }
```

7.3.3.16 newRun()

```

int com.activitytracker.DBManager.newRun (
    final int userID,
    final java.util.Date date,
    final float duration,
    final float distance,
    final float altitudeAscended,
    final float altitudeDescended )
```

Creates a new row in the Runs table with the attributes provided as parameters.

In particular, this method will be called when [Run::newRunDataPoint\(\)](#) receives (0, 0, 0) for (*duration*, *distance*, *altitude*).

Parameters

<i>userID</i>	Unique ID used to associate information in the database to this user.
<i>date</i>	Date that the run was completed.
<i>duration</i>	Duration of the run in seconds.
<i>distance</i>	Distance ran in metres.
<i>altitudeAscended</i>	Cumulative altitude climbed in metres.
<i>altitudeDescended</i>	Cumulative altitude descended in metres.

Returns

Returns a unique integer corresponding to the new row in the SQLite Workouts table by which the new entry can be identified.

Definition at line 453 of file DBManager.java.

```
454                                     {
455
456         int rID = 0;
457         ResultSet res;
458         String sqlInsertQuery = "INSERT INTO Runs (" +
459             "user_id," +
460             "date," +
461             "duration," +
462             "distance," +
463             "altitude_ascended," +
464             "altitude_descended" +
465             ") VALUES (?, ?, ?, ?, ?, ?, ?)";
466         String sqlSelectQuery = "SELECT id FROM Runs WHERE " +
467             "user_id=? AND " +
468             "date=? AND " +
469             "duration=? AND " +
470             "distance=? AND " +
471             "altitude_ascended=? AND " +
472             "altitude_descended=?";
473
474         try {
475             PreparedStatement stmt = m_conn.prepareStatement(sqlInsertQuery);
476             stmt.setInt(1, userID);
477             stmt.setLong(2, date.getTime());
478             stmt.setFloat(3, duration);
479             stmt.setFloat(4, distance);
480             stmt.setFloat(5, altitudeAscended);
481             stmt.setFloat(6, altitudeDescended);
482
483             if (stmt.executeUpdate() != 1) {
484                 System.err.println("Run not added to database.");
485             }
486
487             stmt.close();
488
489             // Pass back in the stuff we just created to get the right row ID
490             // Look at a better way of doing this with OUTPUT clause of INPUT statement
491             stmt = m_conn.prepareStatement(sqlSelectQuery);
492             stmt.setInt(1, userID);
493             stmt.setLong(2, date.getTime());
494             stmt.setFloat(3, duration);
495             stmt.setFloat(4, distance);
496             stmt.setFloat(5, altitudeAscended);
497             stmt.setFloat(6, altitudeDescended);
498
499             res = stmt.executeQuery();
500             rID = res.getInt("id");
501         }
502         catch (final SQLException e) {
503             System.err.println(e.getMessage());
504         }
505
506         return rID;
507     }
```

7.3.3.17 runExists()

```
boolean com.activitytracker.DBManager.runExists (
    final int rID )
```

Determines if a given run ID exists in the database.

Parameters

<i>rID</i>	Unique ID corresponding to the row in the Runs table that we wish to check exists.
------------	--

Returns

This method returns True if the run row with ID *WOID* exists in the database, or False otherwise.

Definition at line 652 of file DBManager.java.

```
652                                     {
653     ResultSet res;
654     String sqlQuery = "SELECT COUNT(*) as count FROM Runs WHERE id=?";
655     boolean exists = false;
656     try {
657         PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
658         stmt.setInt(1, rID);
659         res = stmt.executeQuery();
660         switch (res.getInt("count")) {
661             case 0:
662                 exists = false;
663                 break;
664             case 1:
665                 exists = true;
666                 break;
667             default:
668                 exists = true;
669                 System.err.println("More than one run for ID " +
670                                     Integer.toString(rID) + ". Something isn't right.");
671                 break;
672         }
673     }
674     catch (final SQLException e) {
675         System.err.println(e.getMessage());
676     }
677
678     return exists;
679 }
680 }
```

7.3.3.18 setRun()

```
void com.activitytracker.DBManager.setRun (
    final int rID,
    final float duration,
    final float distance,
    final float altitudeAscended,
    final float altitudeDescended )
```

Updates a run entry in the database as new information becomes available from the input file.

In particular, this method is called when [Run::newRunDataPoint\(\)](#) receives non-(0, 0, 0) input for (*duration*, *distance*, *altitude*).

This method will not be called directly by the application, rather it is called from [Run::newRunDataPoint\(\)](#). Hence that method will take care of adding/subtracting to/from the current stored values for *duration*, *distance*, and *altitude* — here we just take the input and put it in the database.

Parameters

<i>rID</i>	Unique ID used to identify a run in the database.
<i>duration</i>	The number of seconds the user's run lasted.
<i>distance</i>	The cumulative number of metres the user ran.
<i>altitudeAscended</i>	The cumulative number of metres the user climbed.
<i>altitudeDescended</i>	The cumulative number of metres the user descended.

Definition at line 526 of file DBManager.java.

```
527                                     {
528         String sqlQuery = "UPDATE Runs SET " +
529             "duration = ?, " +
530             "distance = ?, " +
531             "altitude_ascended=?, " +
532             "altitude_descended=? " +
533             "WHERE id=? ";
534         try {
535             PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
536             stmt.setFloat(1, duration);
537             stmt.setFloat(2, distance);
538             stmt.setFloat(3, altitudeAscended);
539             stmt.setFloat(4, altitudeDescended);
540             stmt.setInt(5, rID);
541
542             int result = stmt.executeUpdate();
543             System.err.println(Integer.toString(result) + " rows updated in setRun().");
544             if (result != 1) {
545                 System.err.println("Run not updated in database.");
546             }
547
548             stmt.close();
549         }
```

```

550         catch (final SQLException e) {
551             System.err.println(e.getMessage());
552         }
553     }
554 }

```

7.3.3.19 setUserLastRID()

```

void com.activitytracker.DBManager.setUserLastRID (
    final int id,
    final int lastRID )

```

Updates a user's last run ID in the database.

This method will be used to update the run that a particular user last created. This is used when creating new run as the format of the input file requires that we maintain a record of what run we must update if the next line in the file is *not* (0, 0, 0).

See [getUserLastRID\(\)](#) for more information on the user of the *last_run* field in the database.

Parameters

<i>id</i>	Unique ID used to associate information in the database to this user.
<i>lastRID</i>	Integer corresponding to the last row in the Workouts table that the user with ID <i>id</i> created.

Definition at line 422 of file DBManager.java.

```

422                                     {
423     String sqlQuery = "UPDATE Users SET last_run=? WHERE id=?";
424     try {
425         PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
426         stmt.setInt(1, lastRID);
427         stmt.setInt(2, id);
428         if (stmt.executeUpdate() != 1) {
429             System.err.println("User's last run was not updated correctly.");
430         }
431     }
432     catch (final SQLException e) {
433         System.err.println(e.getMessage());
434     }
435 }

```

7.3.3.20 userExists()

```
boolean com.activitytracker.DBManager.userExists (
    final String emailAddress )
```

The [DBManager::userExists\(\)](#) method is designed to facilitate the user experience (UX) design choice of users creating one account to the app and logging in with an existing account for future use. This maintains saved (persistent) data and helps enforce the unique constraint placed on the *email_address* field in the database (again, as users are authenticating using their email address as a user name to identify themselves).

Parameters

<i>emailAddress</i>	The user's email address for which we are checking existence. We use email address here because this is what the user uses to log in to the app.
---------------------	--

Returns

True if the user exists in the database, false otherwise.

Definition at line 120 of file DBManager.java.

```
120                                     {
121     String sqlQuery = "SELECT COUNT(*) AS count FROM Users WHERE 'email_address'=?";
122     boolean exists = false;
123
124     try {
125         PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
126         stmt.setString(1, emailAddress);
127         ResultSet res = stmt.executeQuery();
128         exists = res.getInt("count") > 0;
129
130         stmt.close();
131     }
132     catch (final SQLException e) {
133         System.err.println(e.getMessage());
134     }
135
136     return exists;
137 }
```

7.3.4 Member Data Documentation

7.3.4.1 m_conn

```
Connection com.activitytracker.DBManager.m_conn = null [private]
```

The *m_conn* variable in the [DBManager](#) class is initially assigned the value of *null*.

When [DBManager::init\(\)](#) is invoked, it is made to be the connection to the database and is subsequently used each time a new SQL statement is created.

Definition at line 37 of file DBManager.java.

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

- [app/src/com/activitytracker/DBManager.java](#)

7.4 com.activitytracker.Iteration3Test Class Reference

Collaboration diagram for com.activitytracker.Iteration3Test:



Static Public Member Functions

- static void [main](#) (String[] args)

7.4.1 Detailed Description

Definition at line 12 of file Iteration3Test.java.

7.4.2 Member Function Documentation

7.4.2.1 main()

```
static void com.activitytracker.Iteration3Test.main (  
    String [] args ) [static]
```

Definition at line 14 of file Iteration3Test.java.

```
14                                     {  
15  
16         // Iteration 1 begins here  
17  
18         User john = null;  
19  
20         DBManager dbManager = new DBManager();  
21         if (!dbManager.init("data.db")) {  
22             System.err.println("Failed to initialize DBManager");  
23             System.exit(1);  
24         }  
25  
26         System.out.println("Attempting to create user...");  
27  
28         if (!dbManager.userExists("jdoe@mac.com")) {  
29             Date dob = null;  
30             DateFormat sourceFormat = new SimpleDateFormat("dd-MM-yyyy");  
31             try {  
32                 dob = sourceFormat.parse("03-04-1978");  
33             } catch (final ParseException e) {  
34                 System.err.println(e.getMessage());  
35             }  
36  
37             User.createUser(  
38                 dbManager,  
39                 "John Doe",  
40                 "jdoe@mac.com",  
41                 dob,  
42                 User.Sex.MALE,  
43                 1.6764f,  
44                 54.4310844f,  
45                 "My Very Secure Password"  
46             );  
47         }  
48         else  
49             System.out.println("User already exists.");  
50  
51  
52  
53         if (dbManager.userExists("jdoe@mac.com"))  
54             System.out.println("John Doe was created!");  
55         else  
56             System.out.println("User was NOT created.");  
57  
58  
59         System.out.println("Testing incorrect password...");  
60  
61         try {
```

```

62         john = new User(dbManager,"jdoe@mac.com", "Some Incorrect Password");
63     }
64     catch (final AuthenticationException e) {
65         System.out.println("Incorrect password used; authentication failed.");
66     }
67
68     System.out.println("Authenticating user...");
69
70     try {
71         john = new User(dbManager,"jdoe@mac.com", "My Very Secure Password");
72     }
73     catch (final AuthenticationException e) {
74         System.out.println("Test failed; user could not be authenticated.");
75     }
76
77     // Iteration 1 ended here
78
79     // Iteration 2 begins here
80
81     if (john != null) {
82         Date today = new Date();
83         try {
84             Run.bulkImport(dbManager, john, "/Users/jacobhouse/Google Drive File Stream/My
Drive/Documents/Courses/Computer Science/COMP-2005 Software Engineering/Final
Project/comp2005-activity-tracker/app/InputW0.csv");
85         }
86         catch (final IOException e) {
87             System.err.println(e.getMessage());
88         }
89     }
90     else {
91         System.out.println("John is null. Cannot execute phase 2.");
92     }
93
94     // Iteration 2 ends here
95     // Iteration 3 begins here
96
97     Date date = null;
98     DateFormat sourceFormat = new SimpleDateFormat("dd-MM-yyyy");
99
100    try {
101        date = sourceFormat.parse("01-01-2018");
102    }
103    catch (final ParseException e) {
104        System.err.println(e.getMessage());
105    }
106
107    Vector<Run> runs = Run.getRuns(dbManager, john, date, new Date());
108
109    if (runs == null) {
110        System.out.println("Runs is null.");
111    } else if (runs.size() == 0)
112        System.err.println("No runs in vector.");
113    else {
114        for (Run run : runs) {
115            System.out.println("Retrieved run with ID " + Integer.toString(run.getID()));
116            System.out.println("Run duration: " + Float.toString(run.getDuration()));
117            System.out.println("Run distance: " + Float.toString(run.getDistance()));
118            System.out.println("Run speed: " + Float.toString(run.getSpeed()));
119            System.out.println("Run altitude ascended: " + run.getAltitudeAscended());
120            System.out.println("Run altitude descended: " + run.getAltitudeDescended());
121            System.out.println("Run date: " + run.getRunDate().toString());
122            System.out.println();
123        }
124    }
125
126    RunStats stats = new RunStats(runs);
127    if (!stats.isEmpty()) {

```



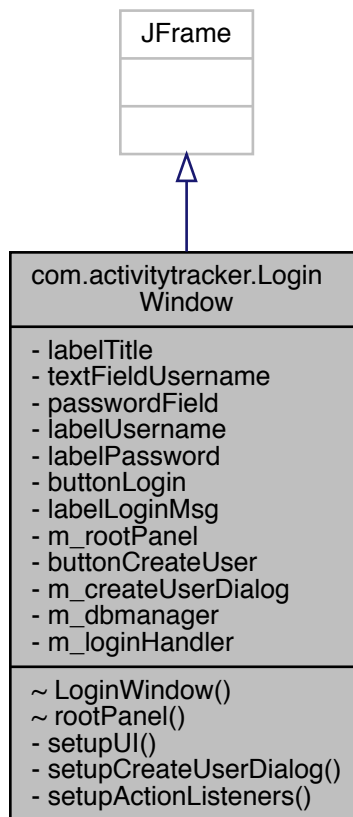
```
128         System.out.println("Average run speed: " + stats.getMeanSpeed());
129         System.out.println("Average run duration: " + stats.getMeanDuration());
130         System.out.println("Average run distance: " + stats.getMeanDistance());
131         System.out.println("Total run distance: " + stats.getTotalDistance());
132         System.out.println("Average altitude gained: " + stats.getMeanAltitudeAscended());
133         System.out.println("Total altitude gained: " + stats.getTotalAltitudeAscended());
134         System.out.println("Average altitude lost: " + stats.getMeanAltitudeDescended());
135         System.out.println("Total altitude lost: " + stats.getTotalAltitudeDescended());
136     }
137     else {
138         System.out.println("RunStats is empty. No stats to show.");
139     }
140
141     // Iteration 3 ends here
142
143 }
```

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

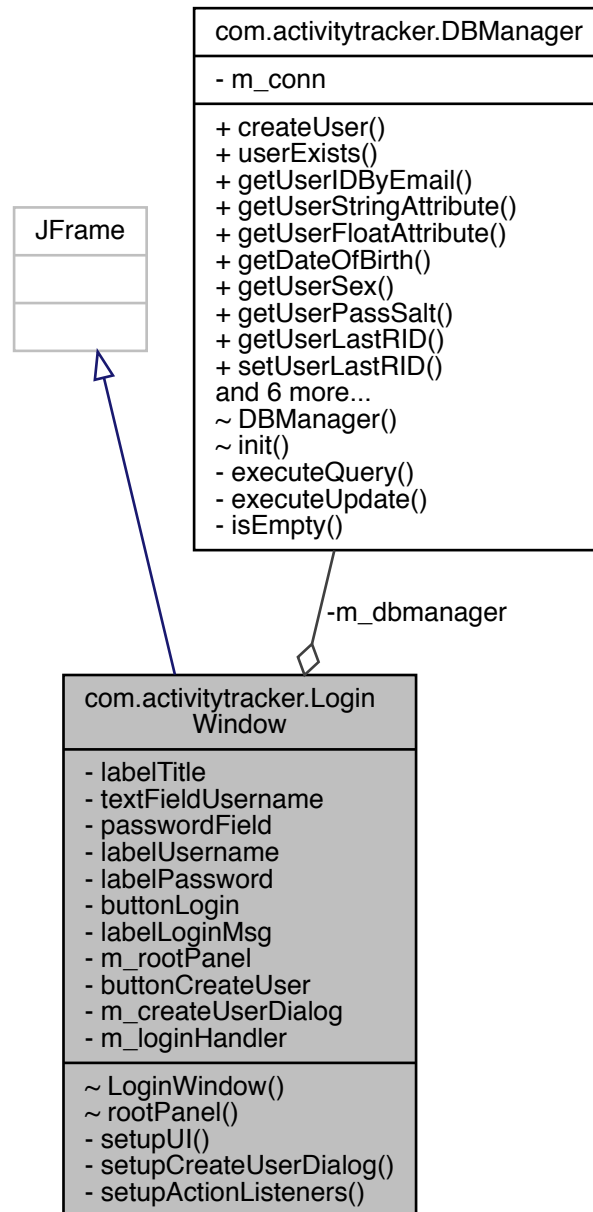
- [app/src/com/activitytracker/Iteration3Test.java](#)

7.5 com.activitytracker.LoginWindow Class Reference

Inheritance diagram for com.activitytracker.LoginWindow:



Collaboration diagram for com.activitytracker.LoginWindow:



Package Functions

- [LoginWindow](#) (java.util.function.Consumer< [User](#) > loginHandler, [DBManager](#) dbmanager)

- JPanel [rootPanel](#) ()

Private Member Functions

- void [setupUI](#) ()
- void [setupCreateUserDialog](#) ()
- void [setupActionListeners](#) ()

Private Attributes

- JLabel [labelTitle](#)
- JTextField [textFieldUsername](#)
- JPasswordField [passwordField](#)
- JLabel [labelUsername](#)
- JLabel [labelPassword](#)
- JButton [buttonLogin](#)
- JLabel [labelLoginMsg](#)
- JPanel [m_rootPanel](#)
- JButton [buttonCreateUser](#)
- JDialog [m_createUserDialog](#) = null
- DBManager [m_dbmanager](#) = null
- java.util.function.Consumer< [User](#) > [m_loginHandler](#)

7.5.1 Detailed Description

Definition at line 13 of file LoginWindow.java.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 LoginWindow()

```
com.activitytracker.LoginWindow.LoginWindow (
    java.util.function.Consumer< User > loginHandler,
    DBManager dbmanager ) [package]
```

Definition at line 30 of file LoginWindow.java.

```
30                                     {
31     m_loginHandler = loginHandler;
32     m_dbmanager = dbmanager;
33
34     setupUI();
35     setupCreateUserDialog();
36     setupActionListeners();
37 }
```

7.5.3 Member Function Documentation

7.5.3.1 rootPanel()

```
JPanel com.activitytracker.LoginWindow.rootPanel ( ) [package]
```

Definition at line 99 of file LoginWindow.java.

```
99                                     {
100     return m_rootPanel;
101 }
```

7.5.3.2 setupActionListeners()

```
void com.activitytracker.LoginWindow.setupActionListeners ( ) [private]
```

Definition at line 59 of file LoginWindow.java.

```

59                                     {
60
61         // Login button
62         buttonLogin.addActionListener(new ActionListener() {
63             @Override
64             public void actionPerformed(ActionEvent e) {
65
66                 // Do nothing if login fields are empty
67                 if (textFieldUsername.getText().isEmpty() ||
passwordField.getPassword().length == 0) {
68                     return;
69                 }
70
71                 User user = null;
72                 try {
73                     user = new User(m_dbmanager, textFieldUsername.getText(),
String.valueOf(passwordField.getPassword()));
74                 } catch (final AuthenticationException ex) {
75                     labelLoginMsg.setVisible(true);
76                     labelLoginMsg.setText("Incorrect login.");
77                     return;
78                 }
79                 catch (final NoSuchElementException ex) {
80                     labelLoginMsg.setVisible(true);
81                     labelLoginMsg.setText("User doesn't exist.");
82                     return;
83                 }
84
85                 m_loginHandler.accept(user);
86
87             }
88         });
89
90         // Create user button
91         buttonCreateUser.addActionListener(new ActionListener() {
92             @Override
93             public void actionPerformed(ActionEvent e) {
94                 m_createUserDialog.setVisible(true);
95             }
96         });
97     }

```

7.5.3.3 setupCreateUserDialog()

```
void com.activitytracker.LoginWindow.setupCreateUserDialog ( ) [private]
```

Definition at line 46 of file LoginWindow.java.

```
46         {
47
48         // Get desktop resolution of default monitor (in case of multi-monitor setups)
49         final GraphicsDevice gd = GraphicsEnvironment.getLocalGraphicsEnvironment().getDefaultScreenDevice(
50         );
51         m_createUserDialog = new JDialog(this, "Activity Logger | Create User", true);
52         m_createUserDialog.setContentPane(new CreateUserWindow(
53         m_dbmanager, (Void) -> m_createUserDialog.hide() ).
54         rootPanel());
55         m_createUserDialog.pack();
56         // Set window size to be 1/2 of screen dimensions
57         m_createUserDialog.setSize(gd.getDisplayMode().getWidth() / 2, gd.getDisplayMode(
58         ).getHeight() / 2);
59         m_createUserDialog.setLocationRelativeTo(this); // Center window
60     }
```

7.5.3.4 setupUI()

void com.activitytracker.LoginWindow.setupUI () [private]

Definition at line 39 of file LoginWindow.java.

```
39         {
40         MaterialUIMovement.add(buttonLogin, MaterialColors.GRAY_100);
41         MaterialUIMovement.add(buttonCreateUser, MaterialColors.GRAY_100);
42
43         labelLoginMsg.setVisible(false);
44     }
```

7.5.4 Member Data Documentation

7.5.4.1 buttonCreateUser

JBUTTON com.activitytracker.LoginWindow.buttonCreateUser [private]

Definition at line 22 of file LoginWindow.java.

7.5.4.2 buttonLogin

`JBUTTON com.activitytracker.LoginWindow.buttonLogin [private]`

Definition at line 19 of file LoginWindow.java.

7.5.4.3 labelLoginMsg

`JLabel com.activitytracker.LoginWindow.labelLoginMsg [private]`

Definition at line 20 of file LoginWindow.java.

7.5.4.4 labelPassword

`JLabel com.activitytracker.LoginWindow.labelPassword [private]`

Definition at line 18 of file LoginWindow.java.

7.5.4.5 labelTitle

`JLabel com.activitytracker.LoginWindow.labelTitle [private]`

Definition at line 14 of file LoginWindow.java.

7.5.4.6 labelUsername

`JLabel com.activitytracker.LoginWindow.labelUsername [private]`

Definition at line 17 of file LoginWindow.java.

7.5.4.7 m_createUserDialog

```
JDialog com.activitytracker.LoginWindow.m_createUserDialog = null [private]
```

Definition at line 24 of file LoginWindow.java.

7.5.4.8 m_dbmanager

```
DBManager com.activitytracker.LoginWindow.m_dbmanager = null [private]
```

Definition at line 26 of file LoginWindow.java.

7.5.4.9 m_loginHandler

```
java.util.function.Consumer<User> com.activitytracker.LoginWindow.m_loginHandler [private]
```

Definition at line 28 of file LoginWindow.java.

7.5.4.10 m_rootPanel

```
JPanel com.activitytracker.LoginWindow.m_rootPanel [private]
```

Definition at line 21 of file LoginWindow.java.

7.5.4.11 passwordField

```
JPasswordField com.activitytracker.LoginWindow.passwordField [private]
```

Definition at line 16 of file LoginWindow.java.

7.5.4.12 textFieldUsername

`TextField com.activitytracker.LoginWindow.textFieldUsername [private]`

Definition at line 15 of file LoginWindow.java.

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

- `app/src/com/activitytracker/LoginWindow.java`

Collaboration diagram for com.activitytracker.MainWindow:



- Generated by Doxygen

Private Member Functions

- void [setupUI](#) () throws HeadlessException
- void [populateTable](#) ()
- void [populateTable](#) (final Date start, final Date end)
- void [setupActionListeners](#) ()

Private Attributes

- final DefaultTableModel [m_tableModel](#) = new DefaultTableModel()
- JPanel [m_rootPanel](#)
- JPanel [topPanel](#)
- JButton [buttonImportData](#)
- JPanel [contentPanel](#)
- JLabel [labelProfileIcon](#)
- JPanel [panelMyActivity](#)
- JTable [tableMyActivity](#)
- JLabel [labelHeyThere](#)
- JTextField [textFieldEndDate](#)
- JTextField [textFieldStartDate](#)
- JButton [buttonGo](#)
- JTextArea [textAreaStats](#)
- DBManager [m_dbManager](#) = null
- User [m_user](#)

7.6.1 Detailed Description

Definition at line 19 of file MainWindow.java.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 MainWindow()

```
com.activitytracker.MainWindow.MainWindow (
    DBManager dbmanager,
    final User user ) [package]
```

Definition at line 38 of file MainWindow.java.

```
38                                     {
39     m_dbManager = dbmanager;
40     m_user = user;
41
42     setupUI();
43     setupActionListeners();
44 }
```

7.6.3 Member Function Documentation

7.6.3.1 populateTable() [1/2]

```
void com.activitytracker.MainWindow.populateTable ( ) [private]
```

Definition at line 74 of file MainWindow.java.

```
74                                     {
75     Date start = new Date(Long.MIN_VALUE);
76     Date end = new Date(Long.MAX_VALUE);
77     populateTable(start, end);
78 }
```

7.6.3.2 populateTable() [2/2]

```
void com.activitytracker.MainWindow.populateTable (
    final Date start,
    final Date end ) [private]
```

Definition at line 80 of file MainWindow.java.

```
80                                     {
81
82     // Clear table model
83     m_tableModel.setRowCount(0);
84
85     final Vector<String> columnNames = new Vector<>();
86     columnNames.add("Date");
87     columnNames.add("Duration (sec)");
88     columnNames.add("Distance (m)");
89     columnNames.add("Altitude + (m)");
90     columnNames.add("Altitude - (m)");
91
92     final Vector<Run> runs = Run.getRuns(m_dbManager, m_user, start, end);
93     final Vector<Vector<Object>> dataVector = new Vector<>();
94
95     if (runs == null) {
96         return;
97     }
98
99     if (!runs.isEmpty()) {
100         for (final Run run : runs) {
101             final Vector<Object> row = new Vector<>();
102
103             row.add(run.getRunDate());
104             row.add(run.getDuration());
105             row.add(run.getDistance());
106             row.add(run.getAltitudeAscended());
107             row.add(run.getAltitudeDescended());
108
109             dataVector.add(row);
110         }
111     }
112
113     m_tableModel.setDataVector(dataVector, columnNames);
114     m_tableModel.setColumnCount(5);
115 }
```

7.6.3.3 rootPanel()

```
JPanel com.activitytracker.MainWindow.rootPanel ( ) [package]
```

Definition at line 184 of file MainWindow.java.

```
184                                     {
185     return m_rootPanel;
186 }
```

7.6.3.4 setupActionListeners()

void com.activitytracker.MainWindow.setupActionListeners () [private]

Definition at line 117 of file MainWindow.java.

```

117                                     {
118         // My Activity button
119         buttonImportData.addActionListener(new ActionListener() {
120             @Override
121             public void actionPerformed(ActionEvent actionEvent) {
122                 final JFileChooser fc = new JFileChooser();
123                 FileNameExtensionFilter filter = new FileNameExtensionFilter("CSV Files", "csv");
124                 fc.setFileFilter(filter);
125                 final int res = fc.showOpenDialog(null);
126
127                 if (res == JFileChooser.APPROVE_OPTION) {
128                     final File file = fc.getSelectedFile();
129
130                     // Run in separate thread
131                     new SwingWorker<Void, Void>() {
132                         @Override
133                         protected Void doInBackground() {
134                             buttonImportData.setEnabled(false);
135                             try {
136                                 Run.bulkImport(m_dbManager, m_user, file.getAbsolutePath()
137                                     );
138                                 buttonImportData.setEnabled(true);
139                                 return null;
140                             }
141                             catch (final Exception e) {
142                                 buttonImportData.setEnabled(true);
143                                 return null;
144                             }
145                         }
146                     }.execute();
147                 }
148             }
149         });
150
151         // Go button
152         buttonGo.addActionListener(new ActionListener() {
153             @Override
154             public void actionPerformed(ActionEvent e) {
155                 final String start = textFieldStartDate.getText();
156                 final String end = textFieldEndDate.getText();
157                 Date startDate, endDate;
158
159                 DateFormat sourceFormat = new SimpleDateFormat("dd-MM-yyyy");
160                 try {
161                     startDate = sourceFormat.parse(start);
162                     endDate = sourceFormat.parse(end);
163                     Vector<Run> runs = Run.getRuns(m_dbManager,
164                         m_user, startDate, endDate);
165                     RunStats stats = new RunStats(runs);
166                     String statStr = "Run Stats beginning at " + start + " and ending at " + end + ":\n" +
167                         "Average Run Duration: " + stats.getMeanDuration() + "\n" +
168                         "Average Speed: " + stats.getMeanSpeed() + "\n" +
169                         "Average Distance: " + stats.getMeanDistance() + "\n" +
170                         "Total Distance: " + stats.getTotalDistance() + "\n" +
171                         "Average Altitude Ascended: " + stats.getMeanAltitudeAscended() + "\n" +
172                         "Total Altitude Ascended: " + stats.getTotalAltitudeAscended() + "\n" +

```

```

173         "Average Altitude Descended: " + stats.getMeanAltitudeDescended();
174         populateTable(startDate, endDate);
175         textAreaStats.setText(statStr);
176     }
177     catch (final ParseException ex) {
178         System.err.println(ex.getMessage());
179     }
180 }
181 });
182 }

```

7.6.3.5 setupUI()

`void com.activitytracker.MainWindow.setupUI () throws HeadlessException [private]`

Definition at line 46 of file MainWindow.java.

```

46                                     {
47
48         // Apply Material-defined hover effect to buttons
49         Color coolGrey11 = new Color(83, 86, 90);
50         MaterialUIMovement.add(buttonImportData, coolGrey11);
51         MaterialUIMovement.add(buttonImportData, coolGrey11);
52
53         // Load and scale logo into UI
54         String logoPath = "./assets/logo.png";
55         ImageIcon imageIcon = new ImageIcon(getClass().getResource(logoPath));
56         final Image image = imageIcon.getImage(); // transform it
57         final Image newimg = image.getScaledInstance(50, 50, java.awt.Image.SCALE_SMOOTH);
58         imageIcon = new ImageIcon(newimg); // transform it back
59         labelProfileIcon.setIcon(imageIcon);
60
61         textFieldStartDate.setText("DD-MM-YYYY");
62         textFieldEndDate.setText("DD-MM-YYYY");
63
64         // Show first name in greeting
65         labelHeyThere.setText(String.format("Hey %s!", m_user.
66         getName().split(" ", 2)[0]));
67
68         panelMyActivity.setVisible(true);
69
70         populateTable();
71
72         tableMyActivity.setModel(m_tableModel);
73     }

```

7.6.4 Member Data Documentation

7.6.4.1 buttonGo

`JButton com.activitytracker.MainWindow.buttonGo [private]`

Definition at line 32 of file MainWindow.java.

7.6.4.2 buttonImportData

`JButton com.activitytracker.MainWindow.buttonImportData [private]`

Definition at line 24 of file MainWindow.java.

7.6.4.3 contentPanel

`JPanel com.activitytracker.MainWindow.contentPanel [private]`

Definition at line 25 of file MainWindow.java.

7.6.4.4 labelHeyThere

`JLabel com.activitytracker.MainWindow.labelHeyThere [private]`

Definition at line 29 of file MainWindow.java.

7.6.4.5 labelProfileIcon

`JLabel com.activitytracker.MainWindow.labelProfileIcon [private]`

Definition at line 26 of file MainWindow.java.

7.6.4.6 m_dbManager

`DBManager` `com.activitytracker.MainWindow.m_dbManager = null` [private]

Definition at line 35 of file MainWindow.java.

7.6.4.7 m_rootPanel

`JPanel` `com.activitytracker.MainWindow.m_rootPanel` [private]

Definition at line 22 of file MainWindow.java.

7.6.4.8 m_tableModel

`final DefaultTableModel` `com.activitytracker.MainWindow.m_tableModel = new DefaultTableModel()` [private]

Definition at line 20 of file MainWindow.java.

7.6.4.9 m_user

`User` `com.activitytracker.MainWindow.m_user` [private]

Definition at line 36 of file MainWindow.java.

7.6.4.10 panelMyActivity

`JPanel` `com.activitytracker.MainWindow.panelMyActivity` [private]

Definition at line 27 of file MainWindow.java.

7.6.4.11 tableMyActivity

`JTable com.activitytracker.MainWindow.tableMyActivity [private]`

Definition at line 28 of file MainWindow.java.

7.6.4.12 textAreaStats

`JTextArea com.activitytracker.MainWindow.textAreaStats [private]`

Definition at line 33 of file MainWindow.java.

7.6.4.13 textFieldEndDate

`TextField com.activitytracker.MainWindow.textFieldEndDate [private]`

Definition at line 30 of file MainWindow.java.

7.6.4.14 textFieldStartDate

`TextField com.activitytracker.MainWindow.textFieldStartDate [private]`

Definition at line 31 of file MainWindow.java.

7.6.4.15 topPanel

`JPanel com.activitytracker.MainWindow.topPanel [private]`

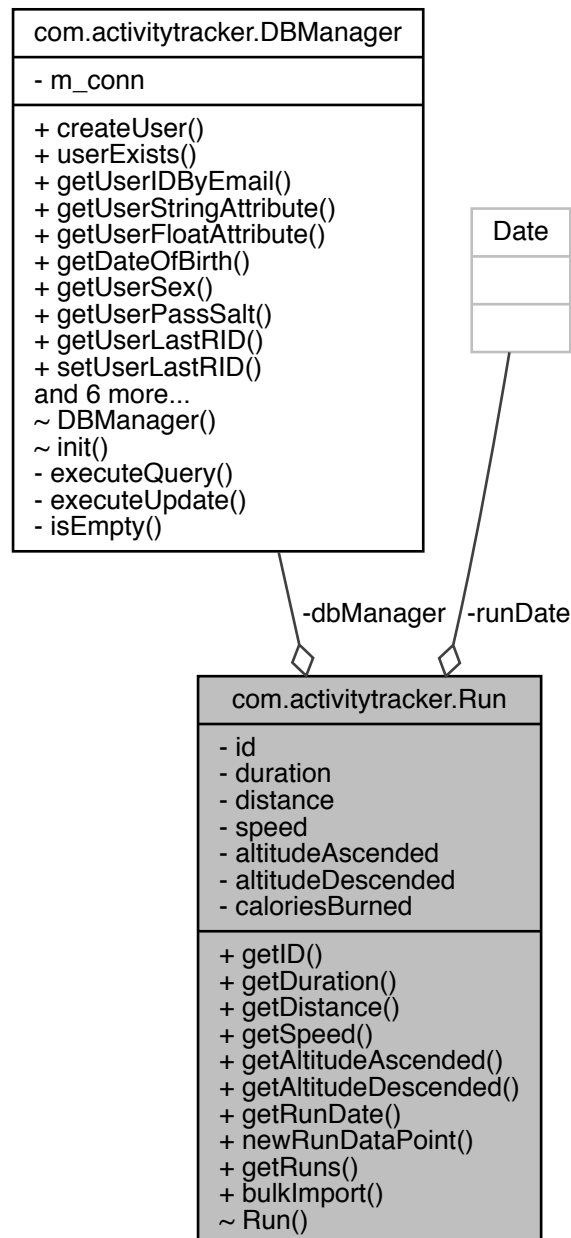
Definition at line 23 of file MainWindow.java.

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

- [app/src/com/activitytracker/MainWindow.java](#)

7.7 com.activitytracker.Run Class Reference

Collaboration diagram for com.activitytracker.Run:



Public Member Functions

- int [getID](#) ()
- float [getDuration](#) ()
- float [getDistance](#) ()
- float [getSpeed](#) ()
- float [getAltitudeAscended](#) ()
- float [getAltitudeDescended](#) ()
- Date [getRunDate](#) ()

Static Public Member Functions

- static void [newRunDataPoint](#) (final [DBManager](#) dbManager, final [User](#) user, final float [duration](#), final Date date, final float [distance](#), final float altitude)
- static Vector< [Run](#) > [getRuns](#) (final [DBManager](#) dbManager, final [User](#) user, final Date startDate, final Date endDate)
- static void [bulkImport](#) (final [DBManager](#) dbManager, final [User](#) user, final String filePath) throws FileNotFoundException, IOException

Package Functions

- [Run](#) (final [DBManager](#) dbManager, final int rID)

Private Attributes

- int [id](#)
- [DBManager](#) [dbManager](#)
- float [duration](#)
- float [distance](#)
- float [speed](#)
- float [altitudeAscended](#)
- float [altitudeDescended](#)
- Date [runDate](#)
- long [caloriesBurned](#)

7.7.1 Detailed Description

Used to logically instantiate a run.

Definition at line 19 of file Run.java.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 Run()

```
com.activitytracker.Run.Run (
    final DBManager dbManager,
    final int rID ) [package]
```

The [Run\(\)](#) constructor is used to retrieve workout information from the database and instantiate each row of the Runs table in a logical format.

Parameters

<i>dbManager</i>	The connection to the database.
<i>rID</i>	The run ID used to retrieve information from the database.

Definition at line 66 of file Run.java.

```
66                                     {
67     this.id = rID;
68     this.dbManager = dbManager;
69     this.runDate = this.dbManager.getRunDate(rID);
70     this.duration = this.dbManager.getRunFloatAttribute(
RunAttribute.DURATION, rID);
71     this.distance = this.dbManager.getRunFloatAttribute(
RunAttribute.DISTANCE, rID);
72     this.speed = this.distance / this.duration;
73     this.altitudeAscended = this.dbManager.
getRunFloatAttribute(RunAttribute.ALTITUDE_ASCENDED, rID);
74     this.altitudeDescended = this.dbManager.
getRunFloatAttribute(RunAttribute.ALTITUDE_DESCENDED, rID);
75     this.caloriesBurned = 0;
76 }
```

7.7.3 Member Function Documentation

7.7.3.1 bulkImport()

```
static void com.activitytracker.Run.bulkImport (
    final DBManager dbManager,
    final User user,
    final String filePath ) throws FileNotFoundException, IOException [static]
```

Opens and iterates through a file. The [Run::newRunDataPoint\(\)](#) method is called for each line.

Parameters

<i>dbManager</i>	Database connection with with the method interacts.
<i>user</i>	A User object corresponding to the use whose run(s) is/are being retrieved from the database.
<i>filePath</i>	The file to be iterated through

Exceptions

<i>FileNotFoundException</i>	Thrown if the file path given does not exist.
<i>IOException</i>	Thrown if there is an error reading or opening the file.

Definition at line 193 of file Run.java.

```

194                                     {
195     BufferedReader br = new BufferedReader(new FileReader(filePath));
196     String line = null;
197     Date date = null;
198     while ((line = br.readLine()) != null)
199     {
200         String[] attributes = line.split(",");
201         String buffTime = attributes[0];
202         String buffDistance = attributes[1];
203         String buffAltitude = attributes[2];
204         String buffDate = attributes[3];
205         DateFormat sourceFormat = new SimpleDateFormat("dd-MM-yyyy");
206         try {
207             date = sourceFormat.parse(buffDate);
208         }
209         catch (final ParseException e) {
210             System.err.println(e.getMessage());
211         }
212
213         // Convert strings to floats
214         float fDur = Float.parseFloat(buffTime);
215         float fDist = Float.parseFloat(buffDistance);
216         float fAlt = Float.parseFloat(buffAltitude);
217
218         newRunDataPoint(dbManager, user, fDur, date, fDist, fAlt);
219     }
220 }
```

7.7.3.2 getAltitudeAscended()

float com.activitytracker.Run.getAltitudeAscended ()

Retrieves a [Run](#) object's altitude ascended (in metres).

Returns

The [Run](#)'s altitude ascended as defined in the database.

Definition at line 263 of file Run.java.

```
263      {
264      BigDecimal a = new BigDecimal(altitudeAscended);
265      return a.setScale(3, RoundingMode.UP).floatValue();
266  }
```

7.7.3.3 getAltitudeDescended()

```
float com.activitytracker.Run.getAltitudeDescended ( )
```

Retrieves a [Run](#) object's altitude descended (in metres).

Returns

The [Run](#)'s altitude descended as defined in the database.

Definition at line 273 of file Run.java.

```
273      {
274      BigDecimal a = new BigDecimal(altitudeDescended);
275      return a.setScale(3, RoundingMode.UP).floatValue();
276  }
```

7.7.3.4 getDistance()

```
float com.activitytracker.Run.getDistance ( )
```

Retrieves a [Run](#) object's distance (in metres).

Returns

The [Run](#)'s distance as defined in the database.

Definition at line 245 of file Run.java.

```
245      {
246      return distance;
247  }
```


7.7.3.5 getDuration()

```
float com.activitytracker.Run.getDuration ( )
```

Retrieves a [Run](#) object's duration (in seconds).

Returns

The [Run](#)'s duration as defined in the database.

Definition at line 236 of file Run.java.

```
236                                     {  
237     return duration;  
238 }
```

7.7.3.6 getID()

```
int com.activitytracker.Run.getID ( )
```

Retrieves a [Run](#) object's ID.

Returns

The [Run](#)'s ID as defined in the database.

Definition at line 227 of file Run.java.

```
227                                     {  
228     return this.id;  
229 }
```

7.7.3.7 getRunDate()

```
Date com.activitytracker.Run.getRunDate ( )
```

Retrieves a [Run](#) object's date.

Returns

The [Run](#)'s date.

Definition at line 283 of file Run.java.

```
283                                     {  
284         return runDate;  
285     }
```

7.7.3.8 getRuns()

```
static Vector<Run> com.activitytracker.Run.getRuns (  
    final DBManager dbManager,  
    final User user,  
    final Date startDate,  
    final Date endDate ) [static]
```

Retrieves a set of runs from the database. Returns the result as a vector of [Run](#) objects.

Parameters

<i>dbManager</i>	Database connection with with the method interacts.
<i>user</i>	A User object corresponding to the use whose run(s) is/are being retrieved from the database.
<i>startDate</i>	The beginning of the interval for which we are retrieving workouts.
<i>endDate</i>	The end of the interval for which we are retrieving workouts.

Returns

A vector containing instances of [Run](#) corresponding to all entered workouts between the start and end dates specified.

Definition at line 162 of file Run.java.

```
163                                     {
164     Vector<Run> runs = new Vector<>();
165     int rID;
166     Vector<Integer> rIDs = dbManager.getRuns(user.getID(), startDate, endDate);
167
168     if (rIDs != null) {
169         Iterator<Integer> runIDIter = rIDs.iterator();
170         while (runIDIter.hasNext()) {
171             rID = runIDIter.next();
172             runs.add(new Run(dbManager, rID));
173         }
174         return runs;
175     }
176     else {
177         System.err.println("DBManager.getRuns() returned null.");
178         return null;
179     }
180 }
```

7.7.3.9 getSpeed()

float com.activitytracker.Run.getSpeed ()

Retrieves a [Run](#) object's average speed (in metres per second).

Returns

The [Run](#)'s average speed as computed in the [Run\(\)](#) constructor.

Definition at line 254 of file Run.java.

```
254                                     {
255     return speed;
256 }
```

7.7.3.10 newRunDataPoint()

```
static void com.activitytracker.Run.newRunDataPoint (
    final DBManager dbManager,
    final User user,
    final float duration,
    final Date date,
    final float distance,
    final float altitude ) [static]
```

Adds a new workout to the database or updates an existing workout with new information that the user imported from the log file.

If (*duration*, *distance*, *altitude*) passed to this method is (0, 0, 0) then the intended assumption is that this is the beginning of a new workout. As such, this input will cause a new row to be added to the Runs table in the database and the user's last run ID attribute will be updated accordingly. If the input is non-(0, 0, 0), then three things take place:

1. The *duration* in the database is overwritten by the *duration* provided as input;
2. The *distance* in the database is overwritten by the *distance* provided as input; and
3. Existing values for *altitude_ascended* and *altitude_descended* are retrieved from the database, their difference is compared to the current relative altitude, and depending whether this difference is positive or negative, the appropriate field in the database is updated to reflect the change.

Parameters

<i>dbManager</i>	Database connection with with the method interacts.
<i>user</i>	A User object corresponding to the use whose run is being added to the database.
<i>duration</i>	The length of time in seconds that the user's run lasted.
<i>date</i>	The date the run occurred.
<i>distance</i>	The cumulative distance (in metres) that the user ran as of the current time passed to the method.
<i>altitude</i>	The relative current altitude (in metres) of the user at the time point being entered. Used to compute cumulative altitude ascended and descended throughout the run.

Definition at line 102 of file Run.java.

```
103
104     int userID = user.getID();
105     int rID;
106     float altitude_ascended;
```

```
{
```

```

107         float altitude_descended;
108
109         if (duration == 0f && distance == 0f && altitude == 0f) {
110             altitude_ascended = 0f;
111             altitude_descended = 0f;
112             rID = dbManager.newRun(
113                 userID,
114                 date,
115                 duration,
116                 distance,
117                 altitude_ascended,
118                 altitude_descended
119             );
120             user.setLastRID(rID);
121             System.err.println("Run " + Integer.toString(rID) + " added to database.");
122         } else {
123             rID = user.getLastRID();
124             if (dbManager.runExists(rID)) {
125                 altitude_ascended = dbManager.getRunFloatAttribute(
RunAttribute.ALTITUDE_ASCENDED, rID);
126                 altitude_descended = dbManager.getRunFloatAttribute(
RunAttribute.ALTITUDE_DESCENDED, rID);
127
128                 System.err.println("Altitude: " + Float.toString(altitude) + ", +: " + Float.toString(
altitude_ascended) + ", -: " + Float.toString(altitude_descended));
129                 if (altitude < altitude_ascended - altitude_descended) {
130                     altitude_descended += altitude_ascended - altitude_descended - altitude;
131                     System.err.println("Went down " + Float.toString(altitude_ascended - altitude_descended
- altitude));
132                 }
133                 else {
134                     altitude_ascended += altitude - (altitude_ascended - altitude_descended);
135                     System.err.println("Went up " + Float.toString(altitude - altitude_ascended -
altitude_descended));
136                 }
137
138                 dbManager.setRun(rID, duration, distance, altitude_ascended,
altitude_descended);
139                 System.err.println("Run " + Integer.toString(rID) + " exists in the database; updating...");
140             } else {
141                 System.err.println("Run table and User table are inconsistent. No changes made.");
142             }
143         }
144
145         altitude_ascended = dbManager.getRunFloatAttribute(RunAttribute.
ALTITUDE_ASCENDED, rID);
146         altitude_descended = dbManager.getRunFloatAttribute(RunAttribute.
ALTITUDE_DESCENDED, rID);
147
148         System.err.println("+: " + Float.toString(altitude_ascended) + ", -: " + Float.toString(
altitude_descended));
149     }

```

7.7.4 Member Data Documentation

7.7.4.1 altitudeAscended

float com.activitytracker.Run.altitudeAscended [private]

The altitude (in metres) that the user climbed throughout the run.

Definition at line 43 of file Run.java.

7.7.4.2 altitudeDescended

```
float com.activitytracker.Run.altitudeDescended [private]
```

The altitude (in metres) that the user descended throughout their run.

Definition at line 47 of file Run.java.

7.7.4.3 caloriesBurned

```
long com.activitytracker.Run.caloriesBurned [private]
```

The number of calories that the user burned throughout their run.

Currently this is not being used; it is for future features.

Definition at line 57 of file Run.java.

7.7.4.4 dbManager

```
DBManager com.activitytracker.Run.dbManager [private]
```

The run's connection to the database. This is used to add data points and retrieve workout metadata.

Definition at line 27 of file Run.java.

7.7.4.5 distance

`float com.activitytracker.Run.distance [private]`

The distance (in metres) that the user ran.

Definition at line 35 of file Run.java.

7.7.4.6 duration

`float com.activitytracker.Run.duration [private]`

The length of the run in seconds.

Definition at line 31 of file Run.java.

7.7.4.7 id

`int com.activitytracker.Run.id [private]`

The run's unique ID.

Definition at line 23 of file Run.java.

7.7.4.8 runDate

`Date com.activitytracker.Run.runDate [private]`

The date the run took place.

Definition at line 51 of file Run.java.

7.7.4.9 speed

`float com.activitytracker.Run.speed [private]`

The average speed (in metres per second) that the user ran.

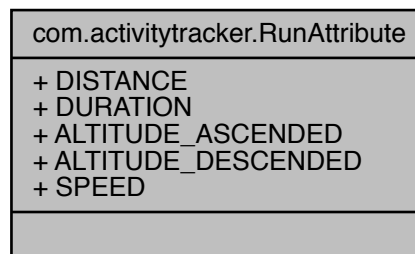
Definition at line 39 of file `Run.java`.

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

- `app/src/com/activitytracker/Run.java`

7.8 com.activitytracker.RunAttribute Enum Reference

Collaboration diagram for `com.activitytracker.RunAttribute`:



Public Attributes

- `DISTANCE`
- `DURATION`
- `ALTITUDE_ASCENDED`
- `ALTITUDE_DESCENDED`
- `SPEED`

7.8.1 Detailed Description

This enumeration type is used to specify the behaviour of generalized methods, particularly in the [DBManager](#) class.

Definition at line 6 of file RunAttribute.java.

7.8.2 Member Data Documentation

7.8.2.1 ALTITUDE_ASCENDED

`com.activitytracker.RunAttribute.ALTITUDE_ASCENDED`

The cumulative altitude (in metres) that the user has climbed throughout their run.

Used in [DBManager::getRunFloatAttribute](#) to specify that ascended altitude should be returned and [RunStats::computeMean\(\)](#).

Definition at line 27 of file RunAttribute.java.

7.8.2.2 ALTITUDE_DESCENDED

`com.activitytracker.RunAttribute.ALTITUDE_DESCENDED`

The cumulative altitude (in metres) that the user has descended throughout their run.

Used in [DBManager::getRunFloatAttribute](#) to specify that descended altitude should be returned and [RunStats::computeMean\(\)](#).

Definition at line 34 of file RunAttribute.java.

7.8.2.3 DISTANCE

`com.activitytracker.RunAttribute.DISTANCE`

The cumulative distance the user has run (in metres).

Used in [DBManager::getRunFloatAttribute](#) to specify that distance should be returned and `RunStats::computeMean()`.

Definition at line 13 of file `RunAttribute.java`.

7.8.2.4 DURATION

`com.activitytracker.RunAttribute.DURATION`

The duration of the user's run (in seconds).

Used in [DBManager::getRunFloatAttribute](#) to specify that duration should be returned and `RunStats::computeMean()`.

Definition at line 20 of file `RunAttribute.java`.

7.8.2.5 SPEED

`com.activitytracker.RunAttribute.SPEED`

The average speed the user ran.

Used in `RunStats::computeMean()`.

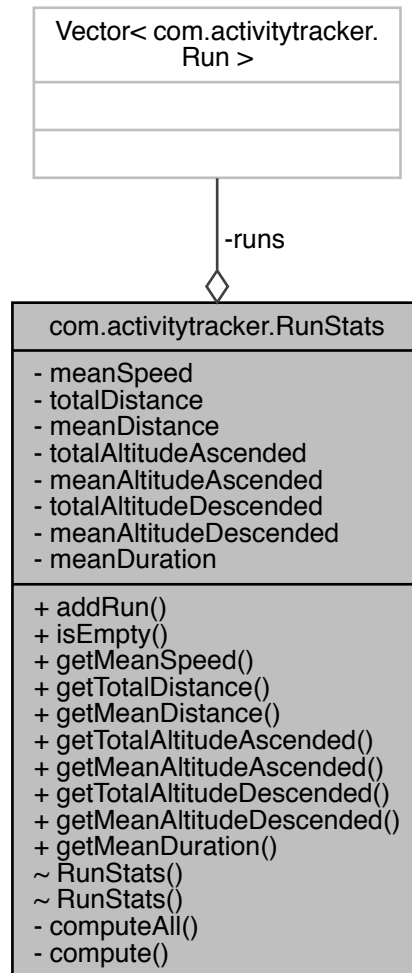
Definition at line 40 of file `RunAttribute.java`.

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

- [app/src/com/activitytracker/RunAttribute.java](#)

7.9 com.activitytracker.RunStats Class Reference

Collaboration diagram for com.activitytracker.RunStats:



Public Member Functions

- void [addRun](#) ([Run](#) run)
- boolean [isEmpty](#) ()
- float [getMeanSpeed](#) ()
- float [getTotalDistance](#) ()

- float [getMeanDistance](#) ()
- float [getTotalAltitudeAscended](#) ()
- float [getMeanAltitudeAscended](#) ()
- float [getTotalAltitudeDescended](#) ()
- float [getMeanAltitudeDescended](#) ()
- float [getMeanDuration](#) ()

Package Functions

- [RunStats](#) (final Vector< [Run](#) > runs)
- [RunStats](#) ()

Private Member Functions

- void [computeAll](#) ()
- void [compute](#) (final [RunAttribute](#) attribute)

Private Attributes

- Vector< [Run](#) > runs
- float [meanSpeed](#)
- float [totalDistance](#)
- float [meanDistance](#)
- float [totalAltitudeAscended](#)
- float [meanAltitudeAscended](#)
- float [totalAltitudeDescended](#)
- float [meanAltitudeDescended](#)
- float [meanDuration](#)

7.9.1 Detailed Description

The [RunStats](#) class is used to compute statistics for a selection of past runs.

It is intended to be integrated with the [Run::getRuns](#) method, as that returns a vector of [Run](#) objects that match search parameters, and the constructor of this class accepts a vector of [Run](#) objects, or nothing.

Definition at line 12 of file RunStats.java.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 RunStats() [1/2]

com.activitytracker.RunStats.RunStats (
 final Vector< Run > runs) [package]

Stores the *runs* parameter in [RunStats::runs](#) and computes all statistics based on the contents of this vector.

Parameters

<i>runs</i>	A vector of Run objects with which we compute statistics.
-------------	---

Definition at line 56 of file RunStats.java.

```

56                                     {
57         this.runs = runs;
58         this.computeAll();
59     }
```

7.9.2.2 RunStats() [2/2]

com.activitytracker.RunStats.RunStats () [package]

An overloaded constructor which takes no argument. Here we assign to [RunStats::runs](#) an empty vector. Computations are attempted but since the vector has size of zero, [RunStats::compute\(\)](#) assigns all stats the value *0.0f*.

You may want to use [RunStats::addRun\(\)](#) to add Runs to a (possibly empty) [RunStats](#) object.

Definition at line 68 of file RunStats.java.

```

68         {
69         this(new Vector<>());
70     }
```

7.9.3 Member Function Documentation

7.9.3.1 addRun()

```
void com.activitytracker.RunStats.addRun (
    Run run )
```

Adds a [Run](#) object to the [RunStats::runs](#) vector and re-computes statistics by invoking [RunStats::computeAll\(\)](#).

Parameters

<i>run</i>	A Run object to be appended to the set of runs that the RunStats statistics are based on.
------------	---

Definition at line 161 of file RunStats.java.

```
161         {
162         this.runs.addElement(run);
163         this.computeAll();
164     }
```

7.9.3.2 compute()

```
void com.activitytracker.RunStats.compute (
    final RunAttribute attribute ) [private]
```

Computes and stores averages for the [RunAttribute](#) passed as *attribute*, and stores a total sum where applicable.

Parameters

<i>attribute</i>	The RunAttribute for which we are computing statistics.
------------------	---

Definition at line 91 of file RunStats.java.

```
91         {
```

```
92     float sum = 0.0f;
93     int numRuns = runs.size();
94
95     // If there are no runs, set everything to 0
96     if (numRuns == 0) {
97         this.meanSpeed = 0.0f;
98         this.totalDistance = 0.0f;
99         this.meanDistance = 0.0f;
100         this.totalAltitudeAscended = 0.0f;
101         this.meanAltitudeAscended = 0.0f;
102         this.totalAltitudeDescended = 0.0f;
103         this.meanAltitudeDescended = 0.0f;
104         this.meanDuration = 0.0f;
105         return;
106     }
107
108     switch (attribute) {
109         case DISTANCE:
110             for (Run run : this.runs)
111                 sum += run.getDistance();
112             this.totalDistance = sum;
113             break;
114         case DURATION:
115             for (Run run : this.runs)
116                 sum += run.getDuration();
117             break;
118         case ALTITUDE_ASCENDED:
119             for (Run run : this.runs)
120                 sum += run.getAltitudeAscended();
121             this.totalAltitudeAscended = sum;
122             break;
123         case ALTITUDE_DESCENDED:
124             for (Run run : this.runs)
125                 sum += run.getAltitudeDescended();
126             this.totalAltitudeDescended = sum;
127             break;
128         case SPEED:
129             for (Run run : this.runs)
130                 sum += run.getSpeed();
131             break;
132     }
133
134     float mean = sum / numRuns;
135
136     switch (attribute) {
137         case DISTANCE:
138             this.meanDistance = mean;
139             break;
140         case DURATION:
141             this.meanDuration = mean;
142             break;
143         case ALTITUDE_ASCENDED:
144             this.meanAltitudeAscended = mean;
145             break;
146         case ALTITUDE_DESCENDED:
147             this.meanAltitudeDescended = mean;
148             break;
149         case SPEED:
150             this.meanSpeed = mean;
151     }
152 }
```

7.9.3.3 computeAll()

```
void com.activitytracker.RunStats.computeAll ( ) [private]
```

A wrapper for the [compute\(\)](#) method in this class that invokes all calculations.

Basically, it's the lazy man's way of calling the full computation.

Definition at line 77 of file RunStats.java.

```
77         {
78     this.compute(RunAttribute.SPEED);
79     this.compute(RunAttribute.DISTANCE);
80     this.compute(RunAttribute.ALTITUDE_ASCENDED);
81     this.compute(RunAttribute.ALTITUDE_DESCENDED);
82     this.compute(RunAttribute.DURATION);
83 }
```

7.9.3.4 getMeanAltitudeAscended()

```
float com.activitytracker.RunStats.getMeanAltitudeAscended ( )
```

Definition at line 194 of file RunStats.java.

```
194         {
195     return meanAltitudeAscended;
196 }
```

7.9.3.5 getMeanAltitudeDescended()

```
float com.activitytracker.RunStats.getMeanAltitudeDescended ( )
```

Definition at line 202 of file RunStats.java.

```
202         {
203     return meanAltitudeDescended;
204 }
```


7.9.3.6 getMeanDistance()

float com.activitytracker.RunStats.getMeanDistance ()

Definition at line 186 of file RunStats.java.

```
186                                     {  
187         return meanDistance;  
188     }
```

7.9.3.7 getMeanDuration()

float com.activitytracker.RunStats.getMeanDuration ()

Definition at line 206 of file RunStats.java.

```
206                                     {  
207         return meanDuration;  
208     }
```

7.9.3.8 getMeanSpeed()

float com.activitytracker.RunStats.getMeanSpeed ()

Definition at line 178 of file RunStats.java.

```
178                                     {  
179         return meanSpeed;  
180     }
```

7.9.3.9 getTotalAltitudeAscended()

float com.activitytracker.RunStats.getTotalAltitudeAscended ()

Definition at line 190 of file RunStats.java.

```
190                                     {
191     return totalAltitudeAscended;
192 }
```

7.9.3.10 getTotalAltitudeDescended()

float com.activitytracker.RunStats.getTotalAltitudeDescended ()

Definition at line 198 of file RunStats.java.

```
198                                     {
199     return totalAltitudeDescended;
200 }
```

7.9.3.11 getTotalDistance()

float com.activitytracker.RunStats.getTotalDistance ()

Definition at line 182 of file RunStats.java.

```
182                                     {
183     return totalDistance;
184 }
```

7.9.3.12 isEmpty()

```
boolean com.activitytracker.RunStats.isEmpty ( )
```

Checks if an instance of [RunStats](#) has any runs with statistics have been computed.

If this method returns *False* then all statistics are set to *0.0f* (*i.e.*, they are useless).

Returns

True if the [RunStats::runs](#) vector has a size of zero, false otherwise.

Definition at line 174 of file RunStats.java.

```
174         {  
175     return this.runs.size() == 0;  
176 }
```

7.9.4 Member Data Documentation

7.9.4.1 meanAltitudeAscended

```
float com.activitytracker.RunStats.meanAltitudeAscended [private]
```

Mean altitude climbed per run, computed using the runs in [RunStats::runs](#).

Definition at line 36 of file RunStats.java.

7.9.4.2 meanAltitudeDescended

```
float com.activitytracker.RunStats.meanAltitudeDescended [private]
```

Mean altitude descended per run, computed using the runs in [RunStats::runs](#).

Definition at line 44 of file RunStats.java.

7.9.4.3 meanDistance

`float com.activitytracker.RunStats.meanDistance [private]`

Mean distance per run, computed using the runs in [RunStats::runs](#).

Definition at line 28 of file RunStats.java.

7.9.4.4 meanDuration

`float com.activitytracker.RunStats.meanDuration [private]`

Mean duration per run, computed using the runs in [RunStats::runs](#).

Definition at line 48 of file RunStats.java.

7.9.4.5 meanSpeed

`float com.activitytracker.RunStats.meanSpeed [private]`

The average speed for all runs in [RunStats::runs](#).

Definition at line 20 of file RunStats.java.

7.9.4.6 runs

`Vector<Run> com.activitytracker.RunStats.runs [private]`

The runs for which the statistics in the [RunStats](#) object pertain.

Definition at line 16 of file RunStats.java.

7.9.4.7 totalAltitudeAscended

```
float com.activitytracker.RunStats.totalAltitudeAscended [private]
```

Cumulative altitude climbed for all runs in [RunStats::runs](#).

Definition at line 32 of file RunStats.java.

7.9.4.8 totalAltitudeDescended

```
float com.activitytracker.RunStats.totalAltitudeDescended [private]
```

Cumulative altitude descended for all runs in [RunStats::runs](#).

Definition at line 40 of file RunStats.java.

7.9.4.9 totalDistance

```
float com.activitytracker.RunStats.totalDistance [private]
```

Cumulative distance ran for all runs in [RunStats::runs](#).

Definition at line 24 of file RunStats.java.

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

- [app/src/com/activitytracker/RunStats.java](#)

7.10 com.activitytracker.SecureString Class Reference

Collaboration diagram for com.activitytracker.SecureString:

com.activitytracker.SecureString
<ul style="list-style-type: none">- secureString- salt
<ul style="list-style-type: none">+ equalString()+ getSalt()+ toString()~ SecureString()~ SecureString()- generateSecureString()- generateSalt()

Public Member Functions

- boolean [equalString](#) (final String other)
- byte [] [getSalt](#) ()
- String [toString](#) ()

Package Functions

- [SecureString](#) (final String plaintext)
- [SecureString](#) (final String plaintext, final byte[] [salt](#))

Private Member Functions

- String [generateSecureString](#) (final String strToSecure, final byte[] [salt](#))

Static Private Member Functions

- static byte [] [generateSalt](#) () throws NoSuchAlgorithmException

Private Attributes

- String [secureString](#)
- byte [] [salt](#)

7.10.1 Detailed Description

This class is used to securely store sensitive string-like information such as user passwords.

Definition at line 10 of file SecureString.java.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 SecureString() [1/2]

```
com.activitytracker.SecureString.SecureString (  
    final String plaintext ) [package]
```

The [SecureString\(\)](#) constructor takes as an argument a plain text string, encrypts it, and stores the encrypted string in the variable [SecureString::secureString](#).

Salt is generated using [SecureString::generateSalt\(\)](#).

Parameters

<i>plaintext</i>	The string to be encrypted. May contain sensitive information.
------------------	--

Definition at line 29 of file SecureString.java.

```
29                                     {  
30  
31         try {  
32             this.salt = generateSalt();  
33         }  
34         catch (final NoSuchAlgorithmException e) {  
35             System.err.println(e.getMessage());  
36         }  
37         this.secureString = generateSecureString(plaintext, this.  
38         salt);  
39     }
```

7.10.2.2 SecureString() [2/2]

```
com.activitytracker.SecureString.SecureString (
    final String plaintext,
    final byte [] salt ) [package]
```

The [SecureString\(\)](#) constructor takes as an argument a plain text string and a previously-generated salt, encrypts the plain text string with the provided salt, and stores the encrypted string in the variable [SecureString::secureString](#).

Parameters

<i>plaintext</i>	The string to be encrypted. May contain sensitive information.
<i>salt</i>	Salt that is used to encrypt <i>plaintext</i> . This parameter is used whenever we wish to encrypt using a previously-generated salt for the purpose of encrypted string comparison.

Definition at line 50 of file SecureString.java.

```
50                                     {
51
52         this.salt = salt;
53         this.secureString = generateSecureString(plaintext,
54         salt);
55     }
```

7.10.3 Member Function Documentation

7.10.3.1 equalString()

```
boolean com.activitytracker.SecureString.equalString (
    final String other )
```

Compares the secure string to the *other* parameter for equality.

This method will likely be used to authenticate a user from a password hash existing in the database.

Parameters

<i>other</i>	A (previously encrypted) string with which we compare SecureString::secureString .
--------------	--

Returns

This method returns True if the hashes of both strings are the same, and False otherwise.

Definition at line 66 of file SecureString.java.

```
66                                     {
67
68         return this.secureString.equals(other);
69
70     }
```

7.10.3.2 generateSalt()

static byte [] com.activitytracker.SecureString.generateSalt () throws No←
SuchAlgorithmException [static], [private]

This method generates salt for encryption of a plain text string.

Returns

Returns a byte array of length sixteen (16) containing the encryption salt.

Exceptions

<i>NoSuchAlgorithmException</i>	Required as <i>SecureRandom.getInstance()</i> may throw this exception and we would like the invoking method to decide how to handle it rather than catching and dismissing it here.
---------------------------------	--

Definition at line 83 of file SecureString.java.

```
83                                     {
84         SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
85         byte[] salt = new byte[16];
86         sr.nextBytes(salt);
87         return salt;
88     }
```

7.10.3.3 generateSecureString()

```
String com.activitytracker.SecureString.generateSecureString (
    final String strToSecure,
    final byte [] salt ) [private]
```

Encrypt string and return secure version.

Due to the importance of securely storing passwords, a "tried and true" method for encrypting passwords found at [this link](#) has been used.

Parameters

<i>strToSecure</i>	The plain text string we wish to encrypt.
<i>salt</i>	The salt with which we will encrypt <i>strToSecure</i> .

Returns

This private method returns the encrypted string to the [SecureString\(\)](#) constructor.

Definition at line 102 of file SecureString.java.

```
102                                     {
103     String generatedPassword = null;
104     try {
105         MessageDigest md = MessageDigest.getInstance("SHA-512");
106         md.update(salt);
107         byte[] strBytes = md.digest(strToSecure.getBytes());
108         StringBuilder sb = new StringBuilder();
109         for (int i = 0; i < strBytes.length; i++) {
110             sb.append(Integer.toString((strBytes[i] & 0xff) + 0x100, 16).substring(1));
111         }
112         generatedPassword = sb.toString();
113     }
114     catch (final NoSuchAlgorithmException e) {
115         System.err.println(e.getMessage());
116     }
117     return generatedPassword;
118 }
```

7.10.3.4 getSalt()

```
byte [] com.activitytracker.SecureString.getSalt ( )
```

Returns

Returns the byte array-type salt used to encrypt the text given to the object's constructor.

Definition at line 123 of file SecureString.java.

```
123         {  
124     return this.salt;  
125     }
```

7.10.3.5 toString()

String com.activitytracker.SecureString.toString ()

Overridden method to return the object as a Java String.

The encrypted string will be returned, though it should be noted for completeness that this is not a full representation of the object since the salt is crucial in arriving at [SecureString::secureString](#) being returned.

Returns

Returns the encrypted string.

Definition at line 136 of file SecureString.java.

```
136         {  
137     return this.secureString;  
138     }
```

7.10.4 Member Data Documentation

7.10.4.1 salt

byte [] com.activitytracker.SecureString.salt [private]

The salt that was used to encrypt the plain text string.

Definition at line 19 of file SecureString.java.

7.10.4.2 secureString

String com.activitytracker.SecureString.secureString [private]

The encrypted string.

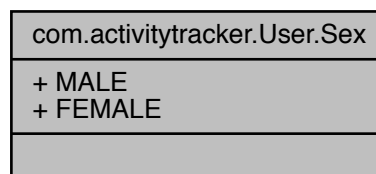
Definition at line 15 of file SecureString.java.

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

- [app/src/com/activitytracker/SecureString.java](#)

7.11 com.activitytracker.User.Sex Enum Reference

Collaboration diagram for com.activitytracker.User.Sex:



Public Attributes

- [MALE](#)
- [FEMALE](#)

7.11.1 Detailed Description

Used to represent whether the user is male or female.

Definition at line 12 of file User.java.

7.11.2 Member Data Documentation

7.11.2.1 FEMALE

`com.activitytracker.User.Sex.FEMALE`

Used to represent that the user is female.

Recall from the source code included in [DBManager::init\(\)](#) that sex is stored in the database using a data type of BIT(1). If the user is female, we store this in the database by populating this field with a *0*.

Definition at line 26 of file User.java.

7.11.2.2 MALE

`com.activitytracker.User.Sex.MALE`

Used to represent that the user is male.

Recall from the source code included in [DBManager::init\(\)](#) that sex is stored in the database using a data type of BIT(1). If the user is female, we store this in the database by populating this field with a *1*.

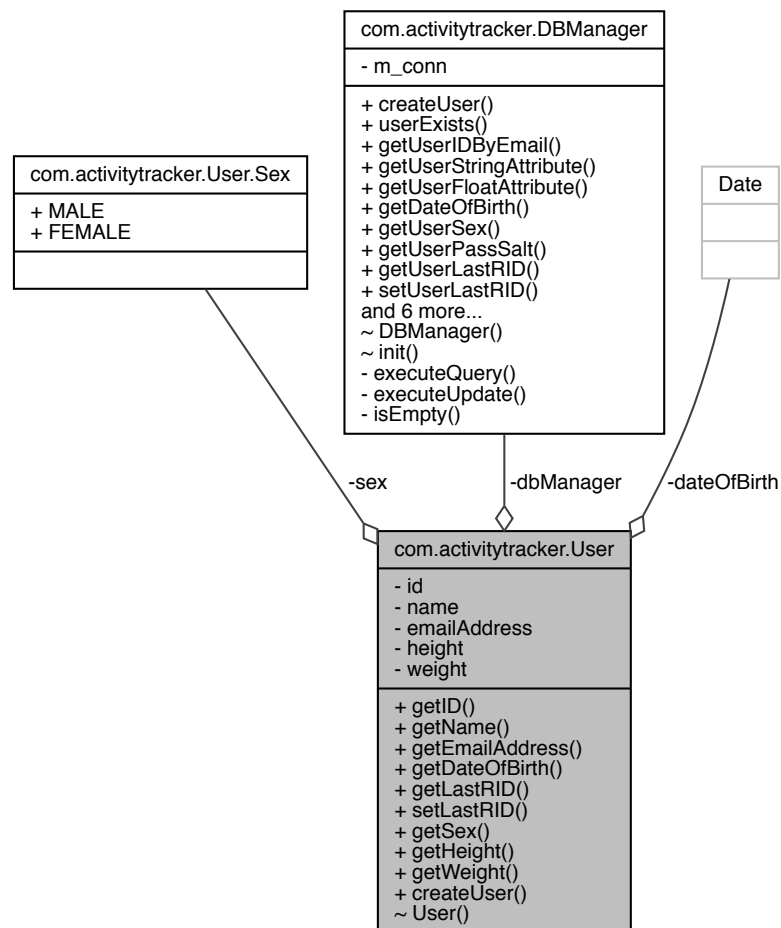
Definition at line 19 of file User.java.

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

- [app/src/com/activitytracker/User.java](#)

7.12 com.activitytracker.User Class Reference

Collaboration diagram for com.activitytracker.User:



Classes

- enum [Sex](#)

Public Member Functions

- int [getID\(\)](#)

- String [getName](#) ()
- String [getEmailAddress](#) ()
- Date [getDateOfBirth](#) ()
- int [getLastRID](#) ()
- void [setLastRID](#) (final int rID)
- [Sex](#) [getSex](#) ()
- float [getHeight](#) ()
- float [getWeight](#) ()

Static Public Member Functions

- static void [createUser](#) (final [DBManager](#) [dbManager](#), final String [name](#), final String [emailAddress](#), final Date [dateOfBirth](#), final User.Sex [sex](#), final float [height](#), final float [weight](#), final String plaintextPassword)

Package Functions

- [User](#) (final [DBManager](#) [dbManager](#), final String [emailAddress](#), final String plaintext←Password) throws AuthenticationException

Private Attributes

- int [id](#)
- String [name](#)
- String [emailAddress](#)
- Date [dateOfBirth](#)
- [Sex](#) [sex](#)
- float [height](#)
- float [weight](#)
- [DBManager](#) [dbManager](#) = null

7.12.1 Detailed Description

Definition at line 8 of file User.java.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 User()

```
com.activitytracker.User.User (
    final DBManager dbManager,
    final String emailAddress,
    final String plaintextPassword ) throws AuthenticationException [pack-
age]
```

Definition at line 64 of file User.java.

```
64
65         {
66             this.dbManager = dbManager;
67             if (this.dbManager.userExists(emailAddress)) {
68
69                 this.id = dbManager.getUserIDByEmail(
70                     emailAddress);
71                 String passHash = this.dbManager.getUserStringAttribute(
72                     UserAttribute.PASSWORD, this.id);
73                 byte[] passSalt = this.dbManager.getUserPassSalt(this.id);
74                 SecureString candidatePassword = new SecureString(plaintextPassword, passSalt);
75
76                 if (candidatePassword.equalString(passHash)) {
77
78
79                     this.name = this.dbManager.getUserStringAttribute(
80                         UserAttribute.NAME, this.id);
81                     // this.emailAddress = this.dbManager.getEmailAddress(this.id);
82                     this.emailAddress = emailAddress;
83                     this.dateOfBirth = this.dbManager.
84                         getDateOfBirth(this.id);
85                     this.sex = this.dbManager.getUserSex(this.id);
86                     this.height = this.dbManager.
87                         getUserFloatAttribute(UserAttribute.HEIGHT, this.id);
88                     this.weight = this.dbManager.
89                         getUserFloatAttribute(UserAttribute.WEIGHT, this.id);
90
91                     System.out.println("Authentication succeeded for " + this.name);
92                 }
93             }
94             else {
95
96                 throw new AuthenticationException("Incorrect password.");
97             }
98         }
99     }
100     else {
101         throw new NoSuchElementException("No such user exists.");
102     }
103 }
```

7.12.3 Member Function Documentation

7.12.3.1 createUser()

```
static void com.activitytracker.User.createUser (
    final DBManager dbManager,
    final String name,
    final String emailAddress,
    final Date dateOfBirth,
    final User.Sex sex,
    final float height,
    final float weight,
    final String plaintextPassword ) [static]
```

Generates a [SecureString](#) for the plain text password provided by the user and passes this, along with the rest of the user's information, to [DBManager](#) for entry in the database.

Parameters

<i>dbManager</i>	An instance of DBManager with which we access the database to store the new user.
<i>name</i>	The new user's full name (<i>e.g.</i> , Johnathan Doe).
<i>emailAddress</i>	The new user's email address (<i>e.g.</i> , jondoe@mac.com) used to register.
<i>dateOfBirth</i>	
<i>sex</i>	
<i>height</i>	
<i>weight</i>	
<i>plaintextPassword</i>	

Definition at line 117 of file User.java.

```
119                                     {
120
121         SecureString securePassword = new SecureString(plaintextPassword);
122
123
124         dbManager.createUser(
125             name,
126             emailAddress,
127             dateOfBirth,
128             sex,
129             height,
130             weight,
131             securePassword
132         );
133
134     }
```

7.12.3.2 getDateOfBirth()

Date com.activitytracker.User.getDateOfBirth ()

Definition at line 148 of file User.java.

```
148         {  
149     return this.dateOfBirth;  
150     }
```

7.12.3.3 getEmailAddress()

String com.activitytracker.User.getEmailAddress ()

Definition at line 144 of file User.java.

```
144         {  
145     return this.emailAddress;  
146     }
```

7.12.3.4 getHeight()

float com.activitytracker.User.getHeight ()

Definition at line 160 of file User.java.

```
160         {  
161     return this.height;  
162     }
```

7.12.3.5 getID()

int com.activitytracker.User.getID ()

Definition at line 136 of file User.java.

```
136      {  
137      return this.id;  
138      }
```

7.12.3.6 getLastRID()

int com.activitytracker.User.getLastRID ()

Definition at line 152 of file User.java.

```
152 { return this.dbManager.getUserLastRID(this.id); }
```

7.12.3.7 getName()

String com.activitytracker.User.getName ()

Definition at line 140 of file User.java.

```
140      {  
141      return this.name;  
142      }
```

7.12.3.8 getSex()

Sex com.activitytracker.User.getSex ()

Definition at line 156 of file User.java.

```
156      {  
157      return this.sex;  
158      }
```

7.12.3.9 `getWeight()`

`float com.activitytracker.User.getWeight ()`

Definition at line 164 of file User.java.

```
164         {  
165             return this.weight;  
166         }
```

7.12.3.10 `setLastRID()`

`void com.activitytracker.User.setLastRID (`
 `final int rID)`

Definition at line 154 of file User.java.

```
154 { this.dbManager.setUserLastRID(this.id, rID); }
```

7.12.4 Member Data Documentation

7.12.4.1 `dateOfBirth`

`Date com.activitytracker.User.dateOfBirth [private]`

The user's date of birth (*e.g.*, 12-12-1998).

Definition at line 45 of file User.java.

7.12.4.2 dbManager

```
DBManager com.activitytracker.User.dbManager = null [private]
```

An instance of [DBManager](#) with which we perform any DB accesses required to retrieve or set user attributes in the database.

Definition at line 62 of file User.java.

7.12.4.3 emailAddress

```
String com.activitytracker.User.emailAddress [private]
```

The email address (*e.g.*, [jondoe@mac.com](#)) that the user entered when registering for the app. This is used to authenticate the user at login.

Definition at line 41 of file User.java.

7.12.4.4 height

```
float com.activitytracker.User.height [private]
```

The user's height in metres.

Definition at line 53 of file User.java.

7.12.4.5 id

```
int com.activitytracker.User.id [private]
```

The user's ID, used by the database to associate workouts with a user.

Definition at line 32 of file User.java.

7.12.4.6 name

String com.activitytracker.User.name [private]

The user's full name (*e.g.*, Johnathan Doe).

Definition at line 36 of file User.java.

7.12.4.7 sex

[Sex](#) com.activitytracker.User.sex [private]

The user's sex. Can be one of [User.Sex.MALE](#) or [User.Sex.FEMALE](#).

Definition at line 49 of file User.java.

7.12.4.8 weight

float com.activitytracker.User.weight [private]

The user's weight in kilograms.

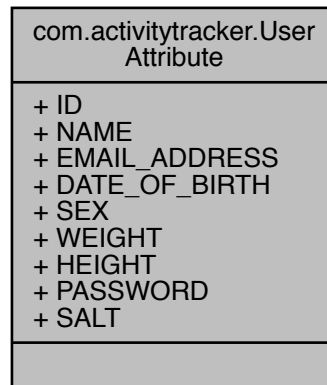
Definition at line 57 of file User.java.

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

- [app/src/com/activitytracker/User.java](#)

7.13 com.activitytracker.UserAttribute Enum Reference

Collaboration diagram for com.activitytracker.UserAttribute:



Public Attributes

- [ID](#)
- [NAME](#)
- [EMAIL_ADDRESS](#)
- [DATE_OF_BIRTH](#)
- [SEX](#)
- [WEIGHT](#)
- [HEIGHT](#)
- [PASSWORD](#)
- [SALT](#)

7.13.1 Detailed Description

This enumeration type is used to specify the behaviour of generalized methods, particularly in the [DBManager](#) class.

Definition at line 6 of file `UserAttribute.java`.

7.13.2 Member Data Documentation

7.13.2.1 DATE_OF_BIRTH

`com.activitytracker.UserAttribute.DATE_OF_BIRTH`

Currently not used as no generalized method retrieves the user's DOB.

Definition at line 26 of file `UserAttribute.java`.

7.13.2.2 EMAIL_ADDRESS

`com.activitytracker.UserAttribute.EMAIL_ADDRESS`

The user's email address.

Used in [DBManager::getUserStringAttribute](#) to specify that the user's email address should be returned.

Definition at line 22 of file `UserAttribute.java`.

7.13.2.3 HEIGHT

`com.activitytracker.UserAttribute.HEIGHT`

The user's height (in metres).

Used in [DBManager::getUserFloatAttribute](#) to specify that the user's email height should be returned.

Definition at line 42 of file `UserAttribute.java`.

7.13.2.4 ID

`com.activitytracker.UserAttribute.ID`

Currently not used as no generalized method retrieves the user's ID.

Definition at line 10 of file `UserAttribute.java`.

7.13.2.5 NAME

`com.activitytracker.UserAttribute.NAME`

The user's full name.

Used in [DBManager::getUserStringAttribute](#) to specify that the user's name should be returned.

Definition at line 16 of file `UserAttribute.java`.

7.13.2.6 PASSWORD

`com.activitytracker.UserAttribute.PASSWORD`

The user's encrypted password hash.

Used in [DBManager::getUserStringAttribute](#) to specify that the user's password hash should be returned.

Definition at line 48 of file `UserAttribute.java`.

7.13.2.7 SALT

`com.activitytracker.UserAttribute.SALT`

Currently not used as no generalized method retrieves the user's password encryption salt.

Definition at line 52 of file `UserAttribute.java`.

7.13.2.8 SEX

`com.activitytracker.UserAttribute.SEX`

Currently not used as no generalized method retrieves the user's sex.

Definition at line 30 of file `UserAttribute.java`.

7.13.2.9 WEIGHT

`com.activitytracker.UserAttribute.WEIGHT`

The user's weight (in kilograms).

Used in [DBManager::getUserFloatAttribute](#) to specify that the user's email weight should be returned.

Definition at line 36 of file `UserAttribute.java`.

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

- [app/src/com/activitytracker/UserAttribute.java](#)

Chapter 8

File Documentation

8.1 app/src/com/activitytracker/ActivityTracker.java File Reference

Classes

- class [com.activitytracker.ActivityTracker](#)

Packages

- package [com.activitytracker](#)

8.2 app/src/com/activitytracker/CreateUserWindow.java File Reference

Classes

- class [com.activitytracker.CreateUserWindow](#)

Packages

- package [com.activitytracker](#)

8.3 app/src/com/activitytracker/DBManager.java File Reference

Classes

- class [com.activitytracker.DBManager](#)

Packages

- package [com.activitytracker](#)

8.4 app/src/com/activitytracker/Iteration3Test.java File Reference

Classes

- class [com.activitytracker.Iteration3Test](#)

Packages

- package [com.activitytracker](#)

8.5 app/src/com/activitytracker/LoginWindow.java File Reference

Classes

- class [com.activitytracker.LoginWindow](#)

Packages

- package [com.activitytracker](#)

8.6 app/src/com/activitytracker/MainWindow.java File Reference

Classes

- class [com.activitytracker.MainWindow](#)

Packages

- package [com.activitytracker](#)

8.7 app/src/com/activitytracker/Run.java File Reference

Classes

- class [com.activitytracker.Run](#)

Packages

- package [com.activitytracker](#)

8.8 app/src/com/activitytracker/RunAttribute.java File Reference

Classes

- enum [com.activitytracker.RunAttribute](#)

Packages

- package [com.activitytracker](#)

8.9 app/src/com/activitytracker/RunStats.java File Reference

Classes

- class [com.activitytracker.RunStats](#)

Packages

- package [com.activitytracker](#)

8.10 app/src/com/activitytracker/SecureString.java File Reference

Classes

- class [com.activitytracker.SecureString](#)

Packages

- package [com.activitytracker](#)

8.11 app/src/com/activitytracker/User.java File Reference

Classes

- class [com.activitytracker.User](#)
- enum [com.activitytracker.User.Sex](#)

Packages

- package [com.activitytracker](#)

8.12 app/src/com/activitytracker/UserAttribute.java File Reference

Classes

- enum [com.activitytracker.UserAttribute](#)

Packages

- package [com.activitytracker](#)

Index

- ALTITUDE_ASCENDED
 - com::activitytracker::RunAttribute, [83](#)
- ALTITUDE_DESCENDED
 - com::activitytracker::RunAttribute, [83](#)
- addRun
 - com::activitytracker::RunStats, [88](#)
- altitudeAscended
 - com::activitytracker::Run, [79](#)
- altitudeDescended
 - com::activitytracker::Run, [80](#)
- app/src/com/activitytracker/ActivityTracker.↵
 - java, [117](#)
- app/src/com/activitytracker/CreateUserWindow.↵
 - java, [117](#)
- app/src/com/activitytracker/DBManager.java,
[118](#)
- app/src/com/activitytracker/Iteration3Test.java,
[118](#)
- app/src/com/activitytracker/LoginWindow.java,
[118](#)
- app/src/com/activitytracker/MainWindow.java,
[118](#)
- app/src/com/activitytracker/Run.java, [119](#)
- app/src/com/activitytracker/RunAttribute.java,
[119](#)
- app/src/com/activitytracker/RunStats.java, [119](#)
- app/src/com/activitytracker/SecureString.java,
[120](#)
- app/src/com/activitytracker/User.java, [120](#)
- app/src/com/activitytracker/UserAttribute.java,
[120](#)
- bulkImport
 - com::activitytracker::Run, [72](#)
- buttonCancel
 - com::activitytracker::CreateUserWindow,
[20](#)
- buttonCreateUser
 - com::activitytracker::LoginWindow, [57](#)
- buttonGo
 - com::activitytracker::MainWindow, [66](#)
- buttonImportData
 - com::activitytracker::MainWindow, [67](#)
- buttonLogin
 - com::activitytracker::LoginWindow, [57](#)
- buttonOk
 - com::activitytracker::CreateUserWindow,
[20](#)
- caloriesBurned
 - com::activitytracker::Run, [80](#)
- com, [11](#)
- com.activitytracker, [11](#)
- com.activitytracker.ActivityTracker, [13](#)
- com.activitytracker.CreateUserWindow, [15](#)
- com.activitytracker.DBManager, [23](#)
- com.activitytracker.Iteration3Test, [48](#)
- com.activitytracker.LoginWindow, [52](#)
- com.activitytracker.MainWindow, [61](#)
- com.activitytracker.Run, [70](#)
- com.activitytracker.RunAttribute, [82](#)
- com.activitytracker.RunStats, [85](#)
- com.activitytracker.SecureString, [96](#)
- com.activitytracker.User, [104](#)
- com.activitytracker.User.Sex, [102](#)
- com.activitytracker.UserAttribute, [113](#)
- com::activitytracker::ActivityTracker
 - main, [14](#)
- com::activitytracker::CreateUserWindow
 - buttonCancel, [20](#)
 - buttonOk, [20](#)
 - comboBoxBirthDay, [20](#)
 - comboBoxBirthMonth, [20](#)
 - comboBoxSex, [20](#)
 - CreateUserWindow, [17](#)
 - labelInfo, [21](#)

- m_closeWindowHandler, 21
- m_dbmanager, 21
- m_rootPanel, 21
- passwordField, 21
- rootPanel, 18
- setupActionListeners, 18
- setupUI, 19
- textFieldBirthYear, 22
- textFieldEmail, 22
- textFieldHeight, 22
- textFieldName, 22
- textFieldWeight, 22
- com::activitytracker::DBManager
 - createUser, 26
 - DBManager, 25
 - executeQuery, 27
 - executeUpdate, 28
 - getDateOfBirth, 29
 - getRunDate, 29
 - getRunFloatAttribute, 30
 - getRuns, 32
 - getUserFloatAttribute, 33
 - getUserIDByEmail, 34
 - getUserLastRID, 35
 - getUserPassSalt, 36
 - getUserSex, 37
 - getUserStringAttribute, 38
 - init, 40
 - isEmpty, 41
 - m_conn, 47
 - newRun, 42
 - runExists, 43
 - setRun, 44
 - setUserLastRID, 46
 - userExists, 46
- com::activitytracker::Iteration3Test
 - main, 49
- com::activitytracker::LoginWindow
 - buttonCreateUser, 57
 - buttonLogin, 57
 - labelLoginMsg, 58
 - labelPassword, 58
 - labelTitle, 58
 - labelUsername, 58
 - LoginWindow, 54
 - m_createUserDialog, 58
 - m_dbmanager, 59
 - m_loginHandler, 59
 - m_rootPanel, 59
 - passwordField, 59
 - rootPanel, 55
 - setupActionListeners, 55
 - setupCreateUserDialog, 56
 - setupUI, 57
 - textFieldUsername, 59
- com::activitytracker::MainWindow
 - buttonGo, 66
 - buttonImportData, 67
 - contentPanel, 67
 - labelHeyThere, 67
 - labelProfileIcon, 67
 - m_dbManager, 67
 - m_rootPanel, 68
 - m_tableModel, 68
 - m_user, 68
 - MainWindow, 62
 - panelMyActivity, 68
 - populateTable, 63
 - rootPanel, 64
 - setupActionListeners, 64
 - setupUI, 66
 - tableMyActivity, 68
 - textAreaStats, 69
 - textFieldEndDate, 69
 - textFieldStartDate, 69
 - topPanel, 69
- com::activitytracker::Run
 - altitudeAscended, 79
 - altitudeDescended, 80
 - bulkImport, 72
 - caloriesBurned, 80
 - dbManager, 80
 - distance, 80
 - duration, 81
 - getAltitudeAscended, 73
 - getAltitudeDescended, 74
 - getDistance, 74
 - getDuration, 74
 - getID, 75
 - getRunDate, 75

- getRuns, 76
- getSpeed, 77
- id, 81
- newRunDataPoint, 77
- Run, 72
- runDate, 81
- speed, 81
- com::activitytracker::RunAttribute
 - ALTITUDE_ASCENDED, 83
 - ALTITUDE_DESCENDED, 83
 - DISTANCE, 83
 - DURATION, 84
 - SPEED, 84
- com::activitytracker::RunStats
 - addRun, 88
 - compute, 88
 - computeAll, 89
 - getMeanAltitudeAscended, 90
 - getMeanAltitudeDescended, 90
 - getMeanDistance, 90
 - getMeanDuration, 91
 - getMeanSpeed, 91
 - getTotalAltitudeAscended, 91
 - getTotalAltitudeDescended, 92
 - getTotalDistance, 92
 - isEmpty, 92
 - meanAltitudeAscended, 93
 - meanAltitudeDescended, 93
 - meanDistance, 93
 - meanDuration, 94
 - meanSpeed, 94
 - RunStats, 87
 - runs, 94
 - totalAltitudeAscended, 94
 - totalAltitudeDescended, 95
 - totalDistance, 95
- com::activitytracker::SecureString
 - equalString, 98
 - generateSalt, 99
 - generateSecureString, 99
 - getSalt, 100
 - salt, 101
 - SecureString, 97
 - secureString, 101
 - toString, 101
- com::activitytracker::User
 - createUser, 106
 - dateOfBirth, 110
 - dbManager, 110
 - emailAddress, 111
 - getDateOfBirth, 107
 - getEmailAddress, 108
 - getHeight, 108
 - getID, 108
 - getLastRID, 109
 - getName, 109
 - getSex, 109
 - getWeight, 109
 - height, 111
 - id, 111
 - name, 111
 - setLastRID, 110
 - sex, 112
 - User, 105
 - weight, 112
- com::activitytracker::User::Sex
 - FEMALE, 103
 - MALE, 103
- com::activitytracker::UserAttribute
 - DATE_OF_BIRTH, 114
 - EMAIL_ADDRESS, 114
 - HEIGHT, 114
 - ID, 114
 - NAME, 115
 - PASSWORD, 115
 - SALT, 115
 - SEX, 115
 - WEIGHT, 116
- comboBoxBirthDay
 - com::activitytracker::CreateUserWindow, 20
- comboBoxBirthMonth
 - com::activitytracker::CreateUserWindow, 20
- comboBoxSex
 - com::activitytracker::CreateUserWindow, 20
- compute
 - com::activitytracker::RunStats, 88
- computeAll

- com::activitytracker::RunStats, 89
- contentPanel
 - com::activitytracker::MainWindow, 67
- createUser
 - com::activitytracker::DBManager, 26
 - com::activitytracker::User, 106
- CreateUserWindow
 - com::activitytracker::CreateUserWindow, 17
- DATE_OF_BIRTH
 - com::activitytracker::UserAttribute, 114
- DBManager
 - com::activitytracker::DBManager, 25
- DISTANCE
 - com::activitytracker::RunAttribute, 83
- DURATION
 - com::activitytracker::RunAttribute, 84
- dateOfBirth
 - com::activitytracker::User, 110
- dbManager
 - com::activitytracker::Run, 80
 - com::activitytracker::User, 110
- distance
 - com::activitytracker::Run, 80
- duration
 - com::activitytracker::Run, 81
- EMAIL_ADDRESS
 - com::activitytracker::UserAttribute, 114
- emailAddress
 - com::activitytracker::User, 111
- equalString
 - com::activitytracker::SecureString, 98
- executeQuery
 - com::activitytracker::DBManager, 27
- executeUpdate
 - com::activitytracker::DBManager, 28
- FEMALE
 - com::activitytracker::User::Sex, 103
- generateSalt
 - com::activitytracker::SecureString, 99
- generateSecureString
 - com::activitytracker::SecureString, 99
- getAltitudeAscended
 - com::activitytracker::Run, 73
- getAltitudeDescended
 - com::activitytracker::Run, 74
- getDateOfBirth
 - com::activitytracker::DBManager, 29
 - com::activitytracker::User, 107
- getDistance
 - com::activitytracker::Run, 74
- getDuration
 - com::activitytracker::Run, 74
- getEmailAddress
 - com::activitytracker::User, 108
- getHeight
 - com::activitytracker::User, 108
- getID
 - com::activitytracker::Run, 75
 - com::activitytracker::User, 108
- getLastRID
 - com::activitytracker::User, 109
- getMeanAltitudeAscended
 - com::activitytracker::RunStats, 90
- getMeanAltitudeDescended
 - com::activitytracker::RunStats, 90
- getMeanDistance
 - com::activitytracker::RunStats, 90
- getMeanDuration
 - com::activitytracker::RunStats, 91
- getMeanSpeed
 - com::activitytracker::RunStats, 91
- getName
 - com::activitytracker::User, 109
- getRunDate
 - com::activitytracker::DBManager, 29
 - com::activitytracker::Run, 75
- getRunFloatAttribute
 - com::activitytracker::DBManager, 30
- getRuns
 - com::activitytracker::DBManager, 32
 - com::activitytracker::Run, 76
- getSalt
 - com::activitytracker::SecureString, 100
- getSex
 - com::activitytracker::User, 109
- getSpeed

- com::activitytracker::Run, 77
- getTotalAltitudeAscended
 - com::activitytracker::RunStats, 91
- getTotalAltitudeDescended
 - com::activitytracker::RunStats, 92
- getTotalDistance
 - com::activitytracker::RunStats, 92
- getUserFloatAttribute
 - com::activitytracker::DBManager, 33
- getUserIDByEmail
 - com::activitytracker::DBManager, 34
- getUserLastRID
 - com::activitytracker::DBManager, 35
- getUserPassSalt
 - com::activitytracker::DBManager, 36
- getUserSex
 - com::activitytracker::DBManager, 37
- getUserStringAttribute
 - com::activitytracker::DBManager, 38
- getWeight
 - com::activitytracker::User, 109
- HEIGHT
 - com::activitytracker::UserAttribute, 114
- height
 - com::activitytracker::User, 111
- ID
 - com::activitytracker::UserAttribute, 114
- id
 - com::activitytracker::Run, 81
 - com::activitytracker::User, 111
- init
 - com::activitytracker::DBManager, 40
- isEmpty
 - com::activitytracker::DBManager, 41
 - com::activitytracker::RunStats, 92
- labelHeyThere
 - com::activitytracker::MainWindow, 67
- labelInfo
 - com::activitytracker::CreateUserWindow, 21
- labelLoginMsg
 - com::activitytracker::LoginWindow, 58
- labelPassword
 - com::activitytracker::LoginWindow, 58
- labelProfileIcon
 - com::activitytracker::MainWindow, 67
- labelTitle
 - com::activitytracker::LoginWindow, 58
- labelUsername
 - com::activitytracker::LoginWindow, 58
- LoginWindow
 - com::activitytracker::LoginWindow, 54
- m_closeWindowHandler
 - com::activitytracker::CreateUserWindow, 21
- m_conn
 - com::activitytracker::DBManager, 47
- m_createUserDialog
 - com::activitytracker::LoginWindow, 58
- m_dbManager
 - com::activitytracker::MainWindow, 67
- m_dbmanager
 - com::activitytracker::CreateUserWindow, 21
 - com::activitytracker::LoginWindow, 59
- m_loginHandler
 - com::activitytracker::LoginWindow, 59
- m_rootPanel
 - com::activitytracker::CreateUserWindow, 21
 - com::activitytracker::LoginWindow, 59
 - com::activitytracker::MainWindow, 68
- m_tableModel
 - com::activitytracker::MainWindow, 68
- m_user
 - com::activitytracker::MainWindow, 68
- MALE
 - com::activitytracker::User::Sex, 103
- main
 - com::activitytracker::ActivityTracker, 14
 - com::activitytracker::Iteration3Test, 49
- MainWindow
 - com::activitytracker::MainWindow, 62
- meanAltitudeAscended
 - com::activitytracker::RunStats, 93
- meanAltitudeDescended
 - com::activitytracker::RunStats, 93

- meanDistance
 - com::activitytracker::RunStats, 93
- meanDuration
 - com::activitytracker::RunStats, 94
- meanSpeed
 - com::activitytracker::RunStats, 94
- NAME
 - com::activitytracker::UserAttribute, 115
- name
 - com::activitytracker::User, 111
- newRun
 - com::activitytracker::DBManager, 42
- newRunDataPoint
 - com::activitytracker::Run, 77
- PASSWORD
 - com::activitytracker::UserAttribute, 115
- panelMyActivity
 - com::activitytracker::MainWindow, 68
- passwordField
 - com::activitytracker::CreateUserWindow, 21
 - com::activitytracker::LoginWindow, 59
- populateTable
 - com::activitytracker::MainWindow, 63
- rootPanel
 - com::activitytracker::CreateUserWindow, 18
 - com::activitytracker::LoginWindow, 55
 - com::activitytracker::MainWindow, 64
- Run
 - com::activitytracker::Run, 72
- runDate
 - com::activitytracker::Run, 81
- runExists
 - com::activitytracker::DBManager, 43
- RunStats
 - com::activitytracker::RunStats, 87
- runs
 - com::activitytracker::RunStats, 94
- SALT
 - com::activitytracker::UserAttribute, 115
- SEX
 - com::activitytracker::UserAttribute, 115
- SPEED
 - com::activitytracker::RunAttribute, 84
- salt
 - com::activitytracker::SecureString, 101
- SecureString
 - com::activitytracker::SecureString, 97
- secureString
 - com::activitytracker::SecureString, 101
- setLastRID
 - com::activitytracker::User, 110
- setRun
 - com::activitytracker::DBManager, 44
- setUserLastRID
 - com::activitytracker::DBManager, 46
- setupActionListeners
 - com::activitytracker::CreateUserWindow, 18
 - com::activitytracker::LoginWindow, 55
 - com::activitytracker::MainWindow, 64
- setupCreateUserDialog
 - com::activitytracker::LoginWindow, 56
- setupUI
 - com::activitytracker::CreateUserWindow, 19
 - com::activitytracker::LoginWindow, 57
 - com::activitytracker::MainWindow, 66
- sex
 - com::activitytracker::User, 112
- speed
 - com::activitytracker::Run, 81
- tableMyActivity
 - com::activitytracker::MainWindow, 68
- textAreaStats
 - com::activitytracker::MainWindow, 69
- textFieldBirthYear
 - com::activitytracker::CreateUserWindow, 22
- textFieldEmail
 - com::activitytracker::CreateUserWindow, 22
- textFieldEndDate
 - com::activitytracker::MainWindow, 69
- textFieldHeight

- com::activitytracker::CreateUserWindow,
[22](#)
- textFieldName
 - com::activitytracker::CreateUserWindow,
[22](#)
- textFieldStartDate
 - com::activitytracker::MainWindow, [69](#)
- textFieldUsername
 - com::activitytracker::LoginWindow, [59](#)
- textFieldWeight
 - com::activitytracker::CreateUserWindow,
[22](#)
- toString
 - com::activitytracker::SecureString, [101](#)
- topPanel
 - com::activitytracker::MainWindow, [69](#)
- totalAltitudeAscended
 - com::activitytracker::RunStats, [94](#)
- totalAltitudeDescended
 - com::activitytracker::RunStats, [95](#)
- totalDistance
 - com::activitytracker::RunStats, [95](#)
- User
 - com::activitytracker::User, [105](#)
- userExists
 - com::activitytracker::DBManager, [46](#)
- WEIGHT
 - com::activitytracker::UserAttribute, [116](#)
- weight
 - com::activitytracker::User, [112](#)