

Course Project Report  
“Internet & Mobile Services”  
Prof. Francesco Ricci

Winter Semester 2009/2010  
Free University of Bozen-Bolzano

# Task Manager for Courier Service (A Mobile Application)

Kosumo (8503)  
R. Topan Berliana (8501)

# Table of Contents

1	Introduction.....	3
2	Functional Requirements.....	3
2.1	Download Tasks.....	3
2.2	View Task List.....	3
2.3	Update Status.....	4
2.4	Update Note.....	4
2.5	Group Task.....	4
2.6	View Nearby Task.....	4
2.7	Receive New Task.....	4
3	System Architecture.....	5
3.1	Client Server Communication.....	5
3.2	Data Structure.....	6
3.2.1	Text File.....	6
3.2.2	Record Store.....	6
3.2.3	Landmark Store.....	8
3.3	Package and Classes.....	8
3.3.1	Package data.....	9
3.3.2	Package gui.....	11
3.4	Libraries and Tools.....	12
3.5	User Interface.....	13
4	Problems and Issues.....	17
4.1	Structuring Classes.....	17
4.2	Location API Problem.....	17
5	References.....	18

# 1 Introduction

This is a report for the “Internet & Mobile Service” course project in winter semester 2009/2010. This document will describe a brief description of application functionality, system architecture, usability and problem or issues related with either the project or the system.

This mobile application is designed for supporting the complexity of package delivery (courier service) task management in the field. Instead of keeping the task in papers, the application will help a courier to deal with order records and providing help using location-based alert functionality.

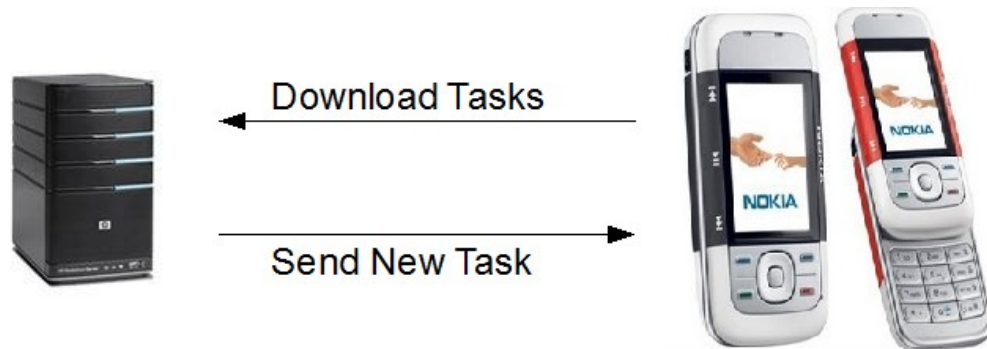


Figure : System Overview

## 2 Functional Requirements

### 2.1 Download Tasks

A courier should be able to download a list of assigned tasks everyday. In a real situation, this task list should be generated from a server-side application which manage all the tasks for all couriers. This includes canceled (delayed) assignments from previous days.

### 2.2 View Task List

This is the core functionality in this small application. Here, a courier should be able to view all assigned tasks for today, including the detailed view of each task. On the main list, the application provide grouping option based on delivery status or order type.

### **2.3 Update Status**

This functionality will be used by a courier to update their task completion status during the day.

### **2.4 Update Note**

The courier should be able to write and update note for a certain task. As mentioned before, one of the purpose is to describe the reason or problem for task cancellation.

### **2.5 Group Task**

To make easier in managing tasks, the application should be able to group the tasks according to certain parameters.

### **2.6 View Nearby Task**

To help reminding the courier when there is a task which is located nearby the current position of the courier, the application should be able to provide alert based on GPS location identification.

### **2.7 Receive New Task**

A courier should be able to receive a new task from the head office and save it to the task list.

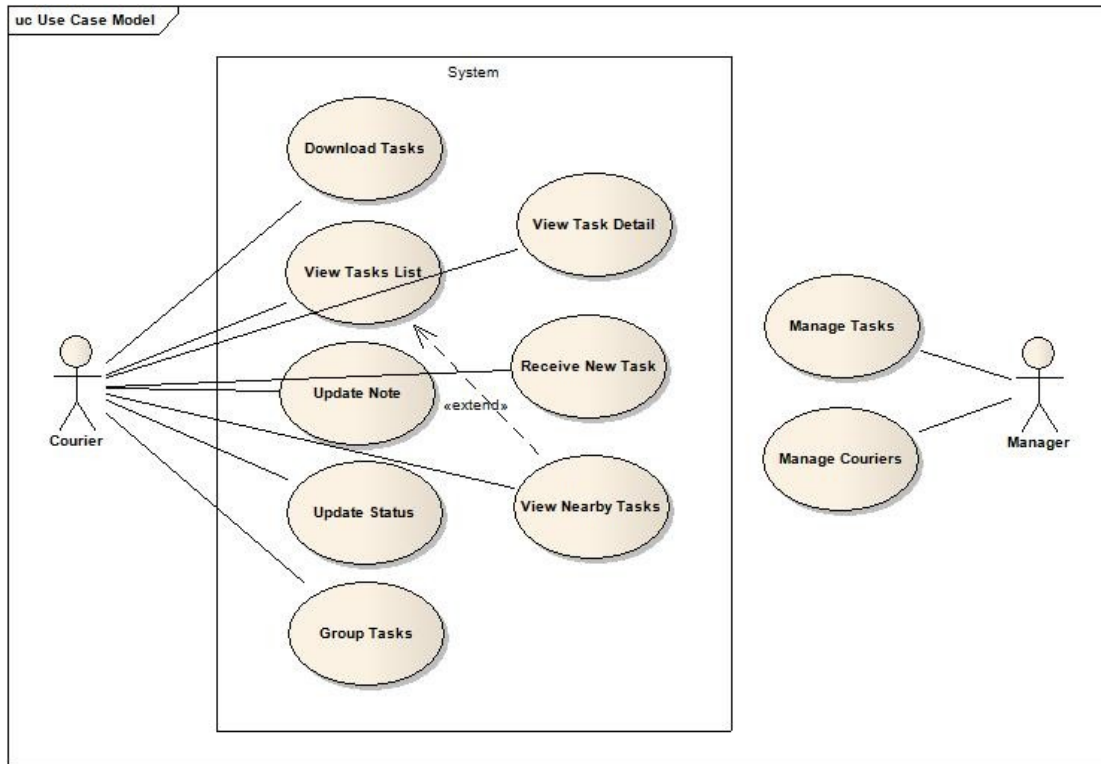


Figure : System Functionality in Use Case Diagram

### 3 System Architecture

#### 3.1 Client Server Communication

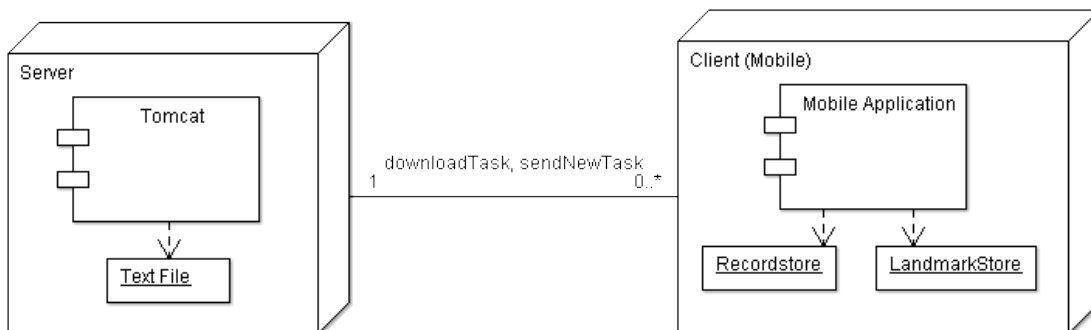


Figure : Client-Server Architecture

As depicted in the above image, the server keeps a text file containing the task list for each client. The client (mobile application) is then required to download the task list and store it locally using Record Store persistent storage. The `HttpConnection` is performed by the client in order to download the file from the server. The incoming text file is then parsed manually to identify specific field of each task. Finally, a new Record Store is created to store the task list.

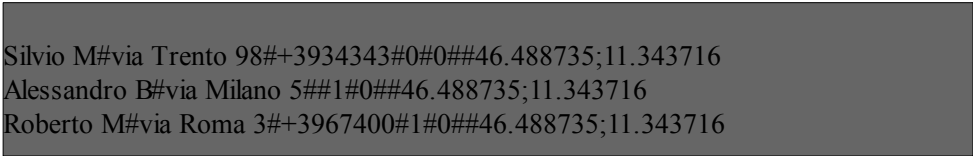
Another communication is needed when the server send a new additional task to the client. In this case, the task is sent as a text message using SMS which is then added to the existing client's Record Store. In case the client refuse to receive the additional task, a confirmation SMS will be sent back to the server.

## 3.2 Data Structure

### 3.2.1 Text File

A text file is stored in the server. In a larger scale of implementation, a Database Management System (DBMS) should be considered.

The following figure describes the content/format of the text file.



```
Silvio M#via Trento 98#+3934343#0#0##46.488735;11.343716  
Alessandro B#via Milano 5##1#0##46.488735;11.343716  
Roberto M#via Roma 3#+3967400#1#0##46.488735;11.343716
```

**Figure : Text file content**

### 3.2.2 Record Store

Record Store is used for keeping the records which have been downloaded from the server, in the mobile device. The following table describes structure of the record store.

<b>taskDb</b>
id
name
address
phone
type
status
note

**Figure : Record Store Structure**

*ID*

This field indicates as identification key for the record.

*Name*

This field describes the name of the recipient.

*Address*

This field is allocated for the recipient's full address (including the city and zip code).

*Phone*

This is the phone number of the package recipient.

*Type*

This is the type of the order/task. There will be 2 order types:

- **Delivery.** Delivering packages (or other goods) to the recipients.
- **Pickup.** Picking up packages (or other goods) from a client to the office. The items will be later send to the recipients.

### *Status*

This field indicates the status of order/task. Basically, there are 3 delivery status:

- **Undelivered.** This is the default status of an order/task, when the task is not done.
- **Delivered.** Opposite to the previous status, this is the condition when the task has been done.
- **Canceled.** This status will be given for a task which can't be finished during the day for a certain reason (for instance: wrong address or the rejected by the recipient). In this case, the courier should provide a note describing the problem. At the end of the day, all task status will be uploaded back to the server, and will be determined whether a canceled task should be retried on the following days or not.

### *Note*

As mentioned before, this field will be needed for describing small notes regarding a cancellation or another reason.

### **3.2.3 Landmark Store**

Landmark Store is used for storing the location information for individual task. Each landmark item in the Landmark Store consists of name of the landmark (associated with an ID from Record Store) and the coordinate (parsed from Text File).

## **3.3 Package and Classes**

The application consists of two packages. All classes for providing data access either using Record Store or Location API are located in `data` package. The `gui` package consists of classes for managing the user interface.

### 3.3.1 Package data

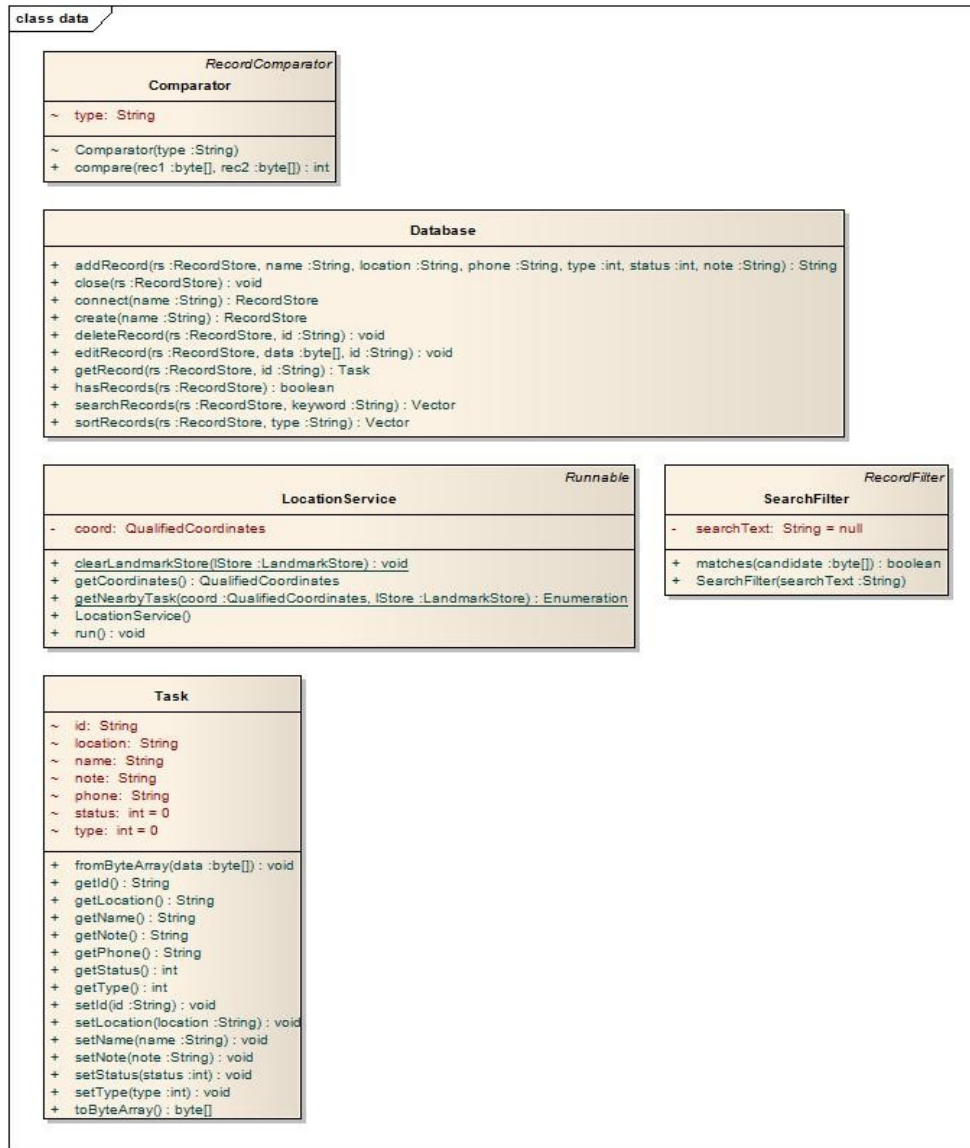


Figure : UML Class Diagram for Package data

<b>Class Name</b>	<b>Description</b>
Task	Holds information about individual task
Database	Provides Record Store related operations
LocationService	Provides location-based operations
Comparator and SearchFilter	Helper classes supporting Record Store operations

### 3.3.2 Package gui

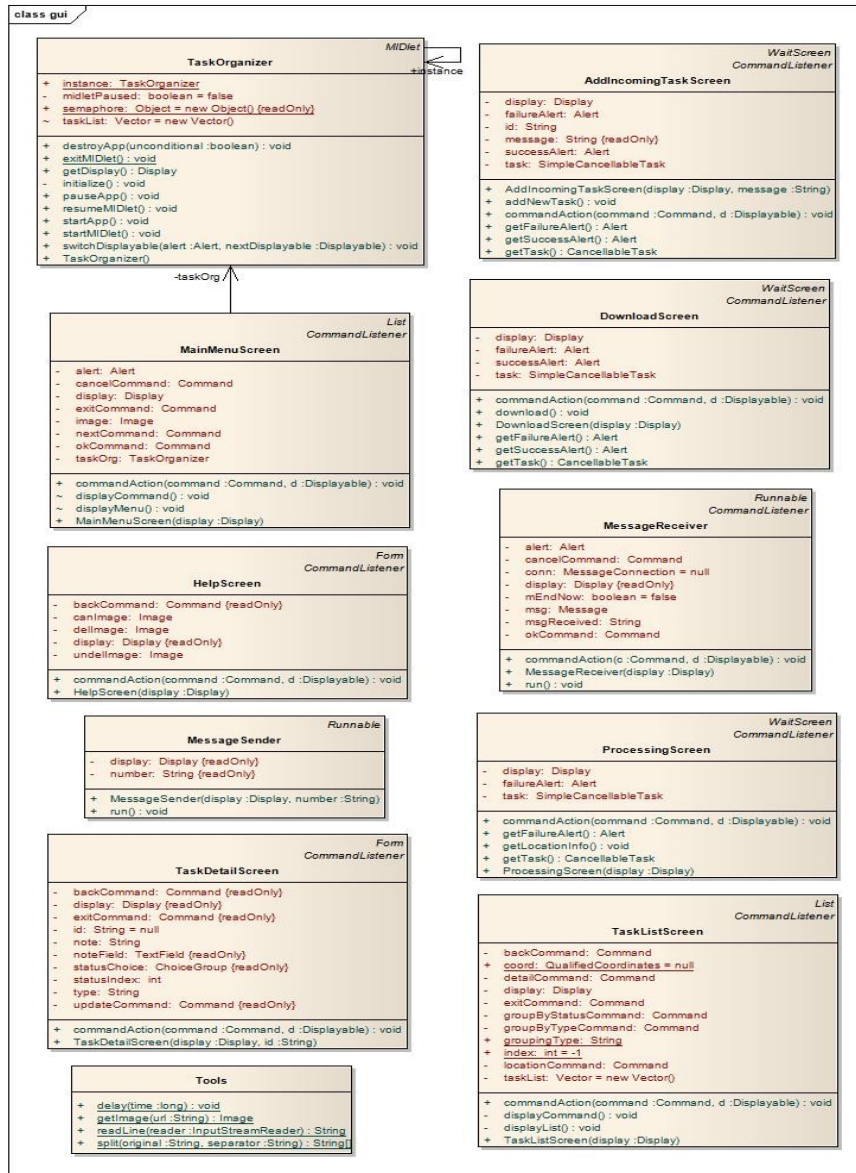


Figure : UML Class Diagram for Package gui

<b>Class Name</b>	<b>Description</b>
MainMenuScreen	Displays main menu of the application
DownloadScreen	Manages file download process
ProcessingScreen	Manages location based process
TaskListScreen	Displays list of tasks stored in the phone
HelpScreen	Displays information about icons and symbols used in the application
TaskDetailScreen	Displays detail information of particular task and user interface for updating data
AddIncomingTaskScreen	Manages incoming additional task from the server
MessageReceiver	Provides functionality for receiving task from the server
MessageSender	Provide functionality for sending sms back to the server
Tools	Provides utility methods for working with string, file, and image
TaskOrganizer	The entry point of the application. This is a MIDlet class for starting up the application and initializing some methods.

### 3.4 Libraries and Tools

The followings are several tools and libraries that have been used for developing this application:

- NetBeans IDE
- Sun Wireless Toolkit (WTK)
- Apache Tomcat
- JSR-179 Location API
- JSR-120 Wireless Messaging API
- WaitScreen component, a part of NetBeans MIDP Components for easily managing long running background tasks (i.e. network and location-based service operations)

In addition, there are some codes that are adapted from other projects and sources in internet. A comment is added in the corresponding code to indicate the reused parts.

### 3.5 User Interface



Main menu of the application



Error alert is appeared when user try to view task list but no file has already been downloaded



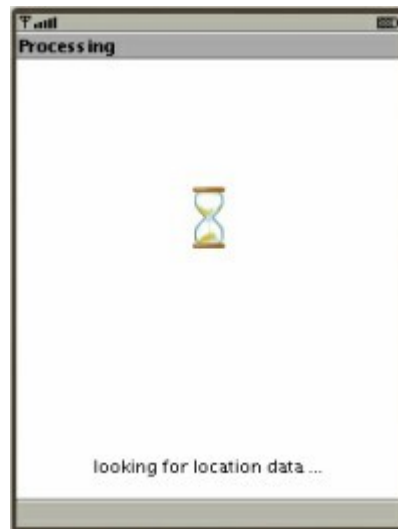
Download confirmation appeared after user selects menu Download



Confirmation alert of successful download



The list of tasks downloaded from the server



Process indicator is appeared after user selects to view nearby task



The application displays list of nearby tasks



Alert indicating availability status of location service



Detail of specific task. This UI also provide menu to update the data (status and note)



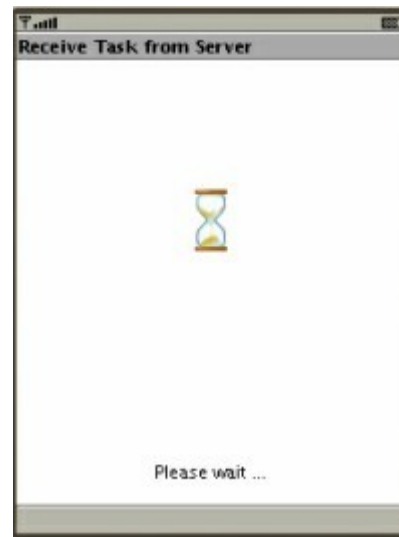
Grouping task by status



Grouping task by type



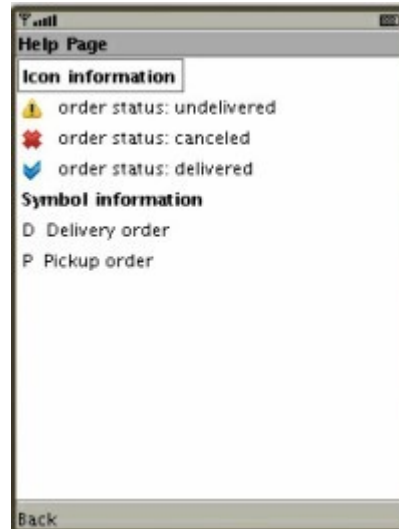
Information alert to indicate incoming additional task from server



Process indicator of storing new additional task



Confirmation of SMS sending to server to respond the cancellation of additional task



Help page to give information about icon and symbol used in the application

## 4 Problems and Issues

### 4.1 Structuring Classes

The main problem that has been encountered during development was designing the class structure of application's user interfaces. It is easy to use only one class for managing the whole user interfaces of the application. However, this approach is not suitable for developing in group. As a result, the user interfaces were separated into several classes and one main MIDlet was used for starting up the application. Here, the problem is the dependency of variables between classes. Though it will make the code difficult to read, we decide to solve this problem by passing the dependent variables into the class constructor and by using some static variables.

### 4.2 Location API Problem

This application works well using NetBeans 6.8 (WTK 2.5.2) on Ubuntu. When we tested using NetBeans 6.8 (WTK 3.0) on Windows, it failed in executing a LandmarkStore operation (to be specific delete Landmark operation). The difference between WTK 2.5.2 and 3.0 is the version of

location API. WTK 2.5.2 is using Location API 1.0.1, while WTK 3.0 is using version 1.0. We believed this is not the cause of the problem. We supposed this problem occurred because security-related problem and then tried to define some API requested permissions in application descriptor. However, this also can't solve this problem. For this reason, we provide two versions of this app:

- **normal:** should be work using WTK 2.5.2
- **limited:** the landmark delete operation is commented (`gui.DownloadScreen.java` line:79).

Every time user download task from server, a landmark is added for each task and stored in Landmark Store. Normally we remove all the landmarks (from previous app execution) in landmark store before adding the new one. However, in this version we remove (comment) this delete operation. As consequence, when user try to download task list, the landmark from previous execution can't be cleared and it will be mixed with the new landmark. this mixed of data cause a confusion later when user try to find nearby tasks.

## 5 References

1. Lecture handouts. <http://www.inf.unibz.it/~ricci/MS/index.html>
2. Databases and MIDP, Part 2: Data Mapping.  
<http://developers.sun.com/mobility/midp/articles/databasemap/>
3. Visual Mobile Designer Custom Components: Creating Wait Screens for Mobile Applications.  
<http://netbeans.org/kb/docs/javame/waitscreen.html>
4. Reading a text file line by line.  
[http://wiki.forum.nokia.com/index.php/CS001006 -  
\\_Reading a text file line by line](http://wiki.forum.nokia.com/index.php/CS001006_-_Reading_a_text_file_line_by_line)