# Blackhawk Production Test

BTech 450
Final Project Report

YuFeng Deng
Department of Computer Science
University of Auckland

Supervisors

Academic Mentor:
Dr S. Manoharan
University of Auckland
NewZealand

Industrial Mentor:
Andrew Radcliffe
Blackhawk Tracking System Ltd.
NewZealand

October 2008

# Abstract

The main goal of this project is designing an application that can check whether the new produced Blackhawk units are working and update the databases accordingly. The application will also be able to generate a report of the testing result. It may be able to do some statistical analysis based on the report generated. This application will be developed using C# based on .net technology. It will be used by the Production staff.

# Contents

# Chapter 1

# Introduction

This report is about the BTech 450 project I have almost done for Blackhawk tracking system Ltd Company. The project is about designing an application that can help the production staff to test the products efficiently. This chapter gives a introduction to the background.

## 1.1 The Company and products

Blackhawk is a company that focuses on developing vehicle tracking systems. They have products for cars, motorbikes and other type of vehicles. They are associated with AA and some insurance companies such as swann insurance [1]. Their business is expanding to Australia recently. They have also being developing a new product for personal use. For more information, visit http://www.theblackhawk.co.nz. Figure 1.1 is the a logo of Blackhawk.


Figure 1.1: The Company logo

The Figure 1.2 shows a Blackhawk unit for car use.

Figure 1.2: A Blackhawk Unit

The Blackhawk units are based on GPS [2] technology. Each of them has a SIM card [3] installed. Every Blackhawk unit is assigned a phone number, so that they are able to receive text message. A Blackhawk unit can track any kind of motor vehicles either by mobile phone or from the Blackhawk website. It can indicate the location of your vehicle at any time as long as it has the GPS signal (The GPRS [4] and GSM [5] signals are required too). Besides reporting geographic location of vehicles, the Blackhawk unit can do a lot of other cool things. For instance, if your vehicle moves without the system being deactivated, you will be notified by text message. You can even immobilise your vehicle by text message the next time it stops. These functionalities are very useful if the vehicle is stolen. Blackhawk unit uses internal aerials so thieves will not know there is a tracking system inside the vehicle. Sounds amazing? Blackhawk units can also detect crash and over speed. If your children take your car, you can know where they are and what speed they are driving at.

## 1.2 How does Blackhawk work

It is very important to understand how Blackhawk unit works before going to the project details. I'm not going to discuss how the Blackhawk's dual immobilisers immobilise your vehicle or how it detects a crash internally. They are not parts of the project. Instead of that, I will introduce how the Blackhawk unit is triggered, how it communicates with the web server and so on. Figure 1.3 shows a example of the Blackhawk working process.

We assume this car has a Blackhawk installed and there are several satellites, not only one. We need at least 4 satellites to get the accurate

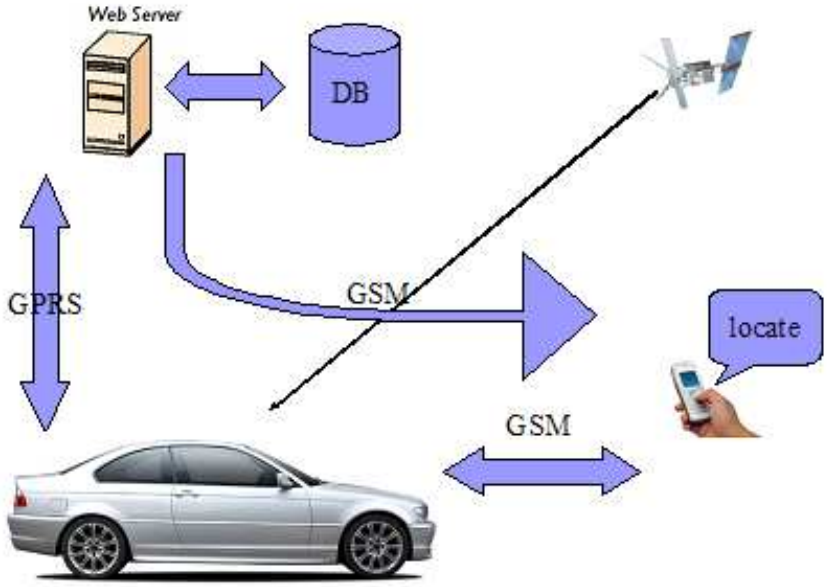


Figure 1.3: How does Blackhawk work

position [6]. In this example, the car owner wants to know the location of his car. He sends a text message "locate" to the Blackhawk unit in his car. The Blackhawk units can response to a lot of events, such as over speed. In this example, the unit is triggered by a txt message. Once the Blackhawk unit gets the "locate" message, it will connect to the web server through GPRS. It then sends an http request by HTTP get method [7]. The query string will include the latitude and longitude that stands for the address of the Blackhawk unit (same as the car). The query string can include other information such as vehicle speed and battery level. The data sent to server is called unit report. What information is included in the unit report depends on the message you send to the Blackhawk unit or other events trigger the Blackhawk unit. In this case, once the server gets the request, it finds that the command is "locate". It then searches for the exact address of the Blackhawk unit in the database based on the latitude and longitude received. Then the server sends back the address to the Blackhawk unit. The Blackhawk unit then sends the address to the owner. The owner now knows where his car is. The server can send TXT message directly to the

owner as well.

## 1.3 Motivation

There are three reasons motivate the company to do this project.

### 1.3.1 Faulty products entering the market costs us.

It's a very bad thing if a faulty product enters the market. Bad products will impress our customers negatively. And our technicians have to reinstall fine units to replace the bad ones. Both of these cost us.

### 1.3.2 There is not an efficient way to test products.

To avoid selling faulty Blackhawk units, we need the production staff in the factory to check whether the new produce Blackhawk units are working properly. The checking process is based on the unit reports sent by the Blackhawk units. It's very inconvenient for production staff to check the reports manually.

### 1.3.3 It costs a lot for testing so many products.

As we produce hundreds of Blackhawk units each time, it will take a lot of time for checking the products. The Blackhawk tracking system Ltd has to spend lots of money to the production staff for doing this. As the company growing, there will be more and more products needed in the future. At that time, it will be not possible for the production staff to check the products manually.

### 1.3.4 The production procedure

Figure 1.4 shows the current production line. Once the products are made, the production staff needs to test them before packing, The products that failed the test will be sent back for reparing, the passed products will be packed and delivered to us, then we need to enter the data of the new products to the database. Both testing products and entering data are time consuming. The main problem is the testing part. There are a bunch of products need to be tested each time, and we don't have a good way to test them. Testing the products one by one will cost too much time, we have to find a way to help the testing staff to do their job easily and quickly.

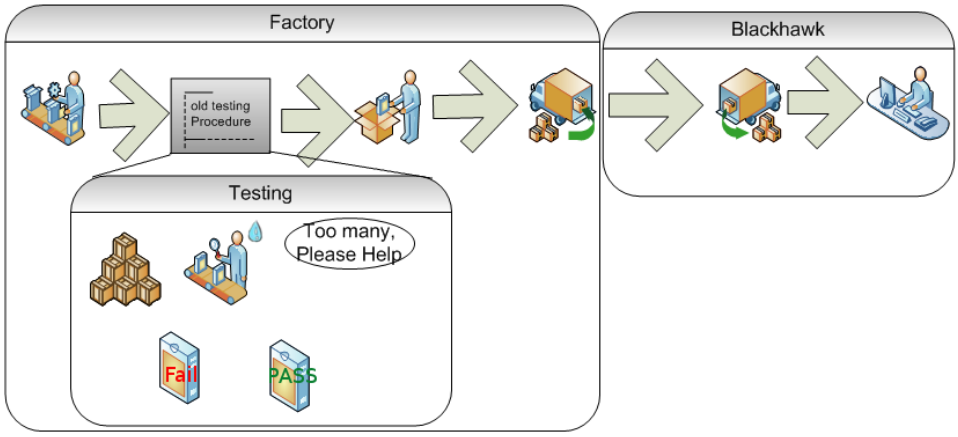


Figure 1.4: The current testing procedure

Due to these reasons, we need a tool to do the checking and update the Blackhawk database efficiently. So we will still be able to manage the production process in the future as the company grows.

## 1.4 Project Goal

The main goal of this project is to design an application that can assist the production staff to do Blackhawk unit checking more efficiently. Beyond this, we want a way to add the pass Blackhawk units to the Blackhawk database easily as well. Figure 1.5 is the new production line we want it to be. With the help of our application, the testing section becomes a very easy job. And we don't need Blackhawk staff to enter the new products' data any more.

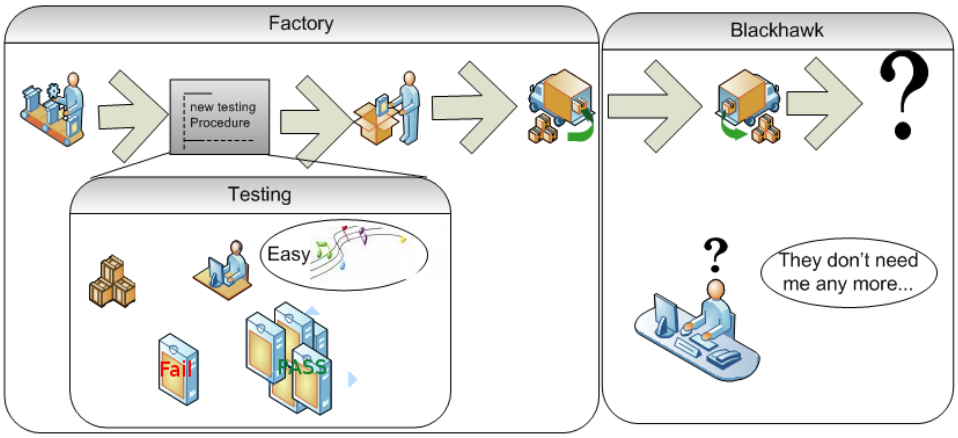


Figure 1.5: What we want it to be

# Chapter 2

# The Solution

## 2.1 The way to test

To solve the testing problem, we need to develop a software program to do the testing for human. How can an application test the product? This is the main problem we have to solve. The application cannot see any physical devices. But it can see the data in the database. We can check the unit reports sent by the Blackhawk units to the database server. From checking the Blackhawk unit reports, we can tell if they are working properly. Figure 2.1 demonstrates the testing way we have. We make the Blackhawk unit get geography data from the satellites and send this data with some other information to our server, then we check the data in the server.

### 2.1.1 Unit Reports

Figure 2.2 is a sample unit report I extracted from Blackhawk database. This is not a function one report (The application only checks function one report), but the data fields are the same. We can see that this is a "Blackhawk installed" report from the Figure 2.2.

Not everything in the report will be checked, so far, we have to check 6 fields, FIX, Battery voltage, SSI, Software version, Function code, Latitude and Longitude. If all of these fields are correct, the unit passes the test. In this way, not all the functionalities of the Blackhawk units will be tested. But it's able to filter out most of the faulty units. And other functionalities will be checked by the installers when the Blackhawks are being installed. That's not a part of this topic. Some fields of this report are easy to understand, for example, in this report the Address row gives the location of the Blackhawk unit which is 11 Lippiatt Road, Otahuhu, Auckland City. This is very

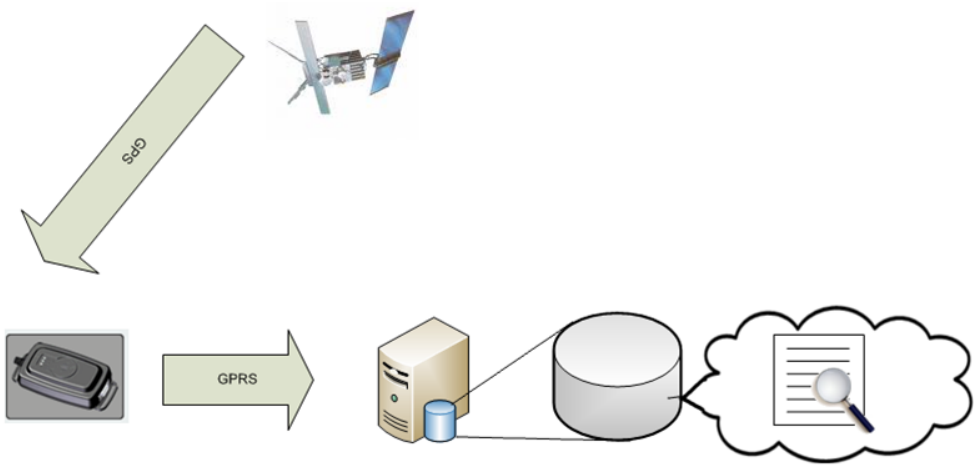


Figure 2.1: The way we can test the product

obvious. But some fields are not that obvious, let us have a look. The FIX should be something new added recently which I don't really know. ID is the event ID. Date Recorded is the time at which this report is received. SEN is the serial number of the Blackhawk unit. IMS is the number belongs to the SIM card. LAT is the latitude of the Blackhawk location. LON is the Longitude of the Blackhawk location. If you look carefully, you will find another Latitude and Longitude in this report. They also stand for the location of the Blackhawk unit, but in different format. SPD is the speed of the vehicle. BAT is the battery voltage and so on.

The application will check this report using the checking rules.

### 2.1.2 Checking Rules

Checking rules are rules we made to check the unit reports. For example, The IMS number should be the same as the IMS number from the factory database, the address should be the factory address. For example, if the factory is on 110 Queen Street, the addresses of the units that are in the factory are 110 Queen Street as well. The report should return "110 Queen Street" in the address field or the corresponding longitude and latitude range. If the report returns an address other than 110 Queen Street, the unit which sent this report has problem. If a unit sends a report which fails to pass any checking rule, we will treat it as a faulty unit. And it will be return to the factory for fixing.

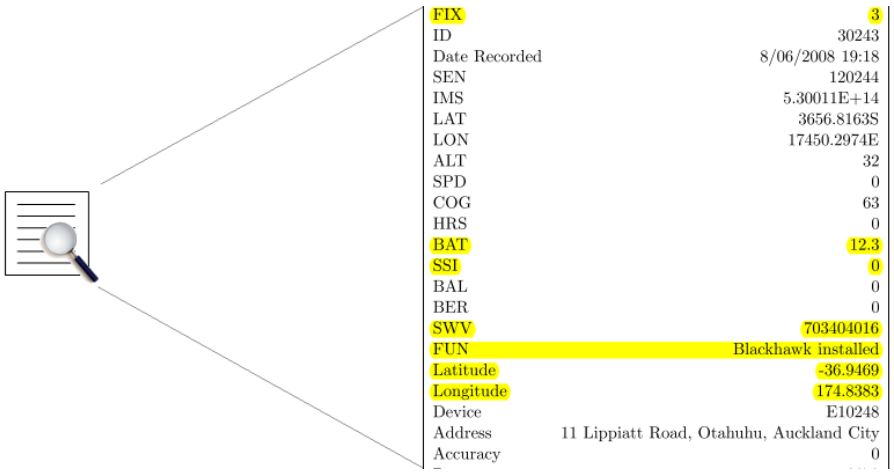


Figure 2.2: The unit report

## 2.2 The Architecture

This application will be used by the factory staff in the factory. Figure 2.3 give a simple introduction of the Architecture. On the Factory side, we have Blackhawk units, testing program, production staff and factory database. The Blackhawk units can get GPS signal from satellites and communicate with Blackhawk server. The Application can access the factory database and communicate with the Blackhawk server. The production staff are the people who using the application to test the products.

### 2.2.1 Factory Database

The factory has a database for managing all new produced Blackhawk units. This database is factory's database and has nothing to do with Blackhawk Company. The application will connect to this database to get the basic information of the new Blackhawk units. Because factory database is also located in the factory as the application, the application will treat it as local database. The Blackhawk unit information in the Factory database is different from the Blackhawk database.

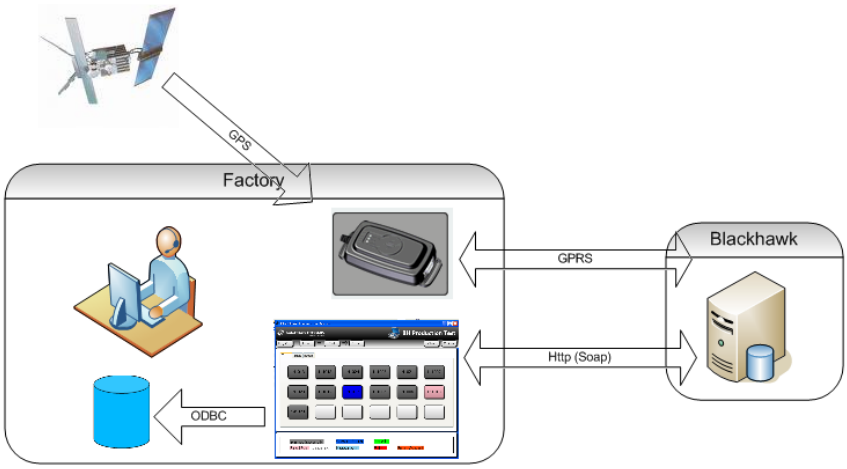


Figure 2.3: The Solution Architecture

### 2.2.2 Blackhawk Database

The Blackhawk database is the database from which the application can get the unit reports. All the reports sent by the Blackhawk units will be recorded in this database. The Blackhawk database is outside the factory. The application needs internet connection to access to the Blackhawk database.

# Chapter 3

# The Application Implementation

## 3.1 How Can We Achieve It

To check the unit report sent by the Blackhawk units, we have to get the reports from Blackhawk server first. How can we read the data from Blackhawk server? We can write SQL query to do that. But this may bring security problem, a cracker can modify the query string to execute malicious commands [8]. To solve this problem, Blackhawk provides Blackhawk APIs. The Blackhawk APIs are based on .net Web Service [9]. These APIs provide a very easy way for the client to communicate with the server. To use this APIs, the user have to login. We can constraint what APIs the user can use, so that they can only read and write some tables of the database.

### 3.1.1 Blackhawk APIs

The Blackhawk Company has provided a bunch of APIs that I can use to operate the Blackhawk database. The Blackhawk APIs are based on web service. Here is a list of the APIs:

* BlackhawkAdminService
* BlackhawkCustomerService
* BlackhawkDeviceService
* BlackhawkEventService

* BlackhawkFactoryService
* BlackhawkInvoiceService
* BlackhawkLoginService
* BlackhawkReportService
* BlackhawkSMSService
* BlackhawkTrackService

**BlackhawkLoginService**

BlackhawkLoginServeice provides the Login, Logout and other methods. To use the Blackhawk APIs, the user have to login to the Blackhawk server within the Login method. Different users will have different access rights to the Blackhawk database. The security will be managed by the Blackhawk API Server, so that the application doesn't need to worry about it.

**BlackhawkEventService**

BlackhawkEventService contains the most important APIs for this application. Using these APIs, we can get all the Blackhawk events as we need. The word "Event" here is the same meaning as report. Each time the Blackhawk unit is triggered, we treat it as an event. Once an event happen, the Blackhawk will send a report to the Blackhawk server. If we want to know what kind of event it is, we can have look at the event report.

Figure 3.1 shows all the web services I can use to get Blackhawk Events. The method FindEventsForDeviceByFunction() was mainly used in this application. But Blackhawk provided a new API GetDeviceStatus() which is easier to use. The powering on event will cause the Blackhawk unit to send a Function one report to the server. The application can use this method to get all the function one report for the devices. The function code of the report is made to indicate what kind of event is triggered. Once a Blackhawk unit is powered on, it will be automatically reset. This will cause the Blackhawk unit to set the function code to 1 and send the report including other information to the Blackhawk server. So we call this report as function one report.

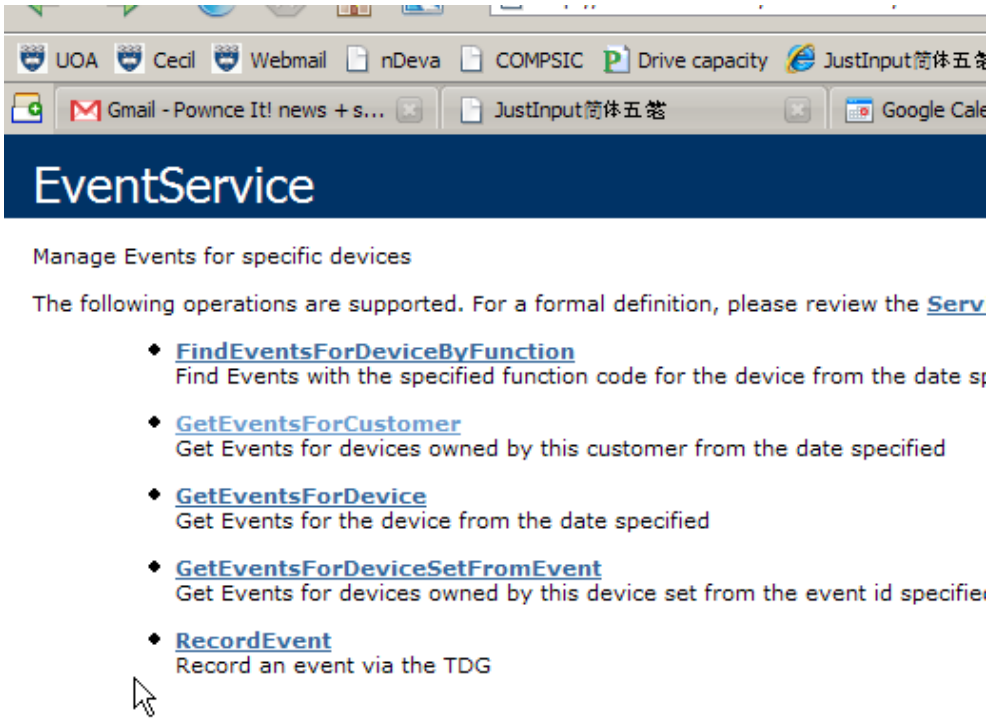


Figure 3.1: The BlackhawkEventService APIs

#### BlackhawkFactoryService

BlackhawkFactoryService has a method we can import the csv file which contains Blackhawk units data to the Blackhawk server.

#### Other Web services

Other web services are not used in the application at this stage. We may need to use them in the future. Some of the Blackhawk APIs are used by other Companies that associated with Blackhawk Tracking System Ltd.

## 3.2 Three steps

Using the application, the production staff only needs three steps to do their job, Importing, Testing and Exporting.

### 3.2.1 Importing

The task of importing is entering the product's data into the Blackhawk database. This was done by the Blackhawk staff when the products are delivered to the company. But as we want the new products to be able to send unit reports to the server, we have to enter the data before testing. Due to the security and other reasons, the Blackhawk server will reject any reports sent by unauthorized units. If the server does not have the information of the new units, the sent reports will not be saved in the Blackhawk server, and our application will have nothing to check.

Firgure 3.2 shows how the production staff can import the new products. It's very simple. What they need to do is clicking the "import" button and select the corresponding file. All the others will be done by the application.


Figure 3.2: The importing procedure

What's in the back of importing? The application will check all the fields of the importing file. If there is anything wrong with the file (such as wrong file format), the application will stop importing and pop up a Message Box

to show the error. If the file is fine, the application will poll the Blackhawk API to upload the data. After uploading, the application will read the imported data from the database and compare it with the importing file. If there is any mismatch, error message will be shown.

The application can accept two type of files, excel and csv. The production staff will be provided an excel or csv file that contains all the information of the units before importing. Figure 3.3 is an example of the excel file. Because the excel file has many columns and very wide, this figure only shows a parts of the excel file. The csv file is very similar to the excel file, but it has fewer information and can be edited using any text editor.

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | SIM Card Nmber | IMS | Phone_Number | Serial_Number | Key | Customer_ |
| 2 | QC | VF | VF | QC | QC | BH |
| 3 | 630011102089725 | 02102019811 | 141013 | 141013 | M66P5F84 | 806 |
| 4 | 630011102089742 | 02102019023 | 141015 | 141015 | 7GK4K75J | 806 |
| 5 | 630011102089743 | 02102020517 | 141024 | 141024 | 7T486Z83 | 806 |
| 6 | 630011102089744 | 02102020478 | 141008 | 141008 | N6MF48GW | 806 |
| 7 | 630011102089745 | 02102019405 | 141021 | 141021 | 8HAGJ5Q8 | 806 |
| 8 | 630011102089746 | 02102020189 | 141007 | 141007 | P2PA77G8 | 806 |
| 9 | 630011102089747 | 02102019829 | 141023 | 141023 | H7B4XP22 | 806 |
| 10 | 630011102089748 | 02102020382 | 141012 | 141012 | A363A52Y | 806 |
| 11 | 630011102089749 | 02102020506 | 141017 | 141017 | J834DPWG | 806 |
| 12 | 630011102089750 | 02102020473 | 141005 | 141005 | 6ZY78BK2 | 806 |
| 13 | 630011102089751 | 02102020476 | 141006 | 141006 | A3TK268D | 806 |
| 14 | 630011102089752 | 02102020409 | 141018 | 141018 | 3FQ3HGM5 | 806 |
| 15 | 630011102089753 | 02102020173 | 141020 | 141020 | 8W68NW2Y | 806 |

Figure 3.3: The sample excel file

### 3.2.2 Testing

Testing is divided to two sub steps, Scan and Test.

#### Scan

Scan is the way we enter the serial numbers of the Blackhawk units. Obviously, using scanner to read the serious number is much faster than using keyboard, especially when the testing person is two-finger typer who is not good at typing. Entering the serial numbers is not the only thing we do at this step. We need to trigger the Blackhawk units to send data to the Blackhawk server. Blackhawk units can be triggered by many ways; the simplest way is powering it on. To trigger the units, the testing staff scans

a unit and puts it in one holder of the tray, and powers it on, and then scan the next one till finish. To track the location of the faulty units, we designed the testing panel this way (see Figure 3.4), so that we can record the location of all the Blackhawk units. Each button on the testing panel stands for one unit. The button location on the testing panel is the same as the unit location in the tray.

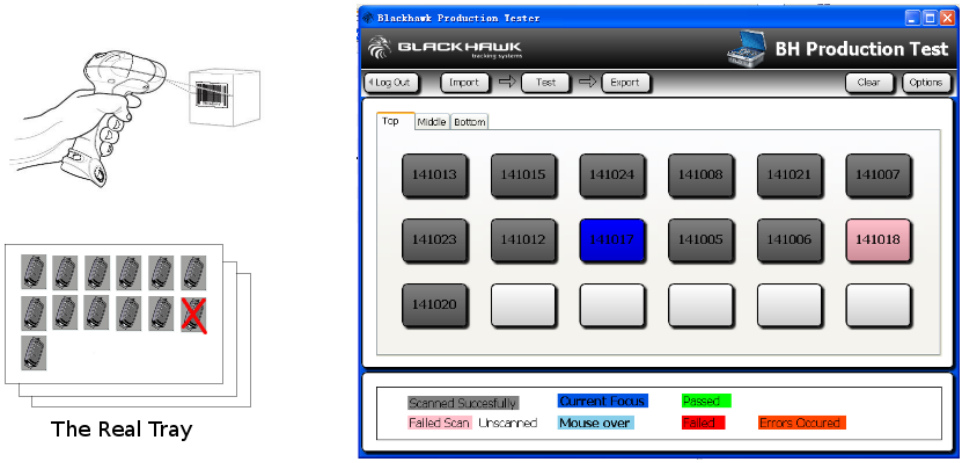


Figure 3.4: Scan Blackhawk units into application

Initially, the current focus is on the first button. When the testing person scans the first unit and puts it in the tray, the first button on the application testing panel will become black, and the scanned serial number will be displayed on the first button. The current focus will go to the second button. The application now is waiting for the second serial number to be scanned in. In this case, you can see that the last button of the second row is pink, which means the last Blackhawk unit in the second row of the tray cannot be scanned successfully. The testing staff has to remove it and place another one. The reason of failed scanning can be varied. The Blackhawk unit may not be activated in the factory database or not enabled in the Blackhawk database. Or, the Blackhawk unit is not in Blackhawk database at all (is not imported).

The current focus can be changed by clicking any button on the testing panel. For example, if you want to replace the second Blackhawk unit in the tray, you can click the second button on the testing panel, then remove the second Blackhawk unit from the tray and scan a new one. The serail number

displayed on the second button will be changed to the serial number of the new Blackhawk unit. You can delete any serial number using the context menu by right click on the button.

### Test

After scanning, we go to test section. In the Scan section, the production staff does a lot and the application do a little. But in the test section, the situation reverses. In this section, the testing staff clicks the "Test" button then leave. All the testing stuff will be handled by the application.

You may be interested in what the application did in the background. Let me explain it a little bit. After clicking the "start" button, the application will get and check the units reports from the Blackhawk database every 30 seconds. The units that sent problem reports will be highlighted orange. The units that sent fine reports will be highlighted green. Other units whose reports cannot be received will be highlighted red after clicking stop button.

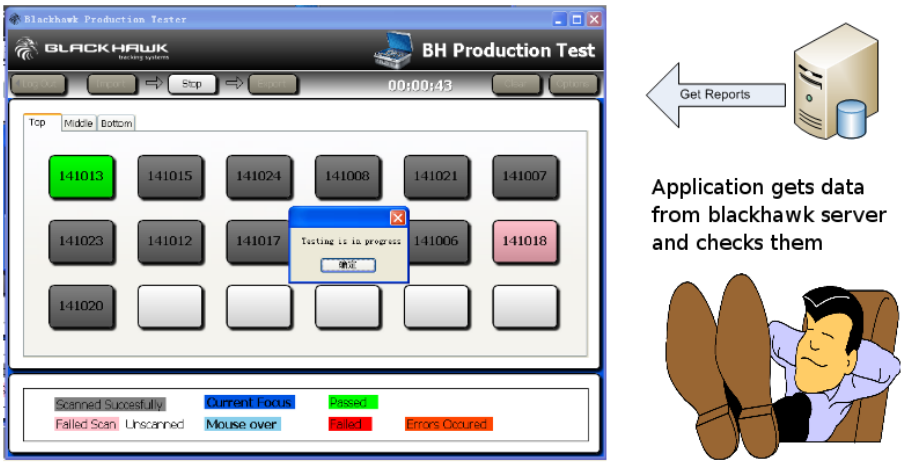


Figure 3.5: Test and wait

### Why every 30 seconds

Once the tester put a Blackhawk unit in the tray and power it on, the Blackhawk unit will be reset. Then it will send a function one report to the Blackhawk server. This process will take some time. Because the Blackhawk unit has to warm up and gather all the information that needs to be sent,

such as Latitude and Longitude. The latitude and longitude are got from the GPS satellites. This procedure does take a long time. If the GPS signal is too week, it take longer time for the Blackhawk unit to get the latitude and longitude. If there is not GPS signal, the testing cannot be done. The Blackhawk Company is planning to set up a GPS magnifier in the factory to avoid any GPS signal problem. Once all the information is gathered, the Blackhawk unit will connect to the Blackhawk server through GPRS. As GPRS is a kind of slow connection, it also takes time to send the report to the Blackhawk server. Beyond that, we need to consider the GPRS signal as well. For these reasons, the Blackhawk server will not get the units reports immediately after the units are powered on. Some reports may be received earlier; some reports may be received later. To make sure we can get all the sent reports, we set a polling interval which is 30 seconds (can be changed).

### 3.2.3 Exporting

Exporting is not real exporting; we use this word to make the application more understandable. It includes two steps, update Blackhawk database and update factory database.

#### Update Blackhawk Database

Exporting is the step we make the new products usable. The imported units are put in the factory device set; a device set is a group of Blackhawk units in Blackhawk database. Units in factory device set are ready to be tested. The units in Blackhawk device set are ready to use. What Exporting do is moving the units from factory device set to Blackhawk device set. Figure 3.6 shows a example of exporting. Unit 141013 has passed the test and exported, it will be packed and delivered to the Blackhawk Company. It's ready for use once it's delivered to Blackhawk.

#### Update Factory Database

The Factory also needs to manage the Blackhawk units as well. They need to know what Blackhawk units are activated. The factory database is much simple comparing to Blackhawk database. It has only 9 columns. They are ID, QCID, BHID, SIMSERIALNO, WO, TIME, USER, ACTIVATION and GPS. After checking the report, the application should set the Activation and GPS columns of passed units to true.

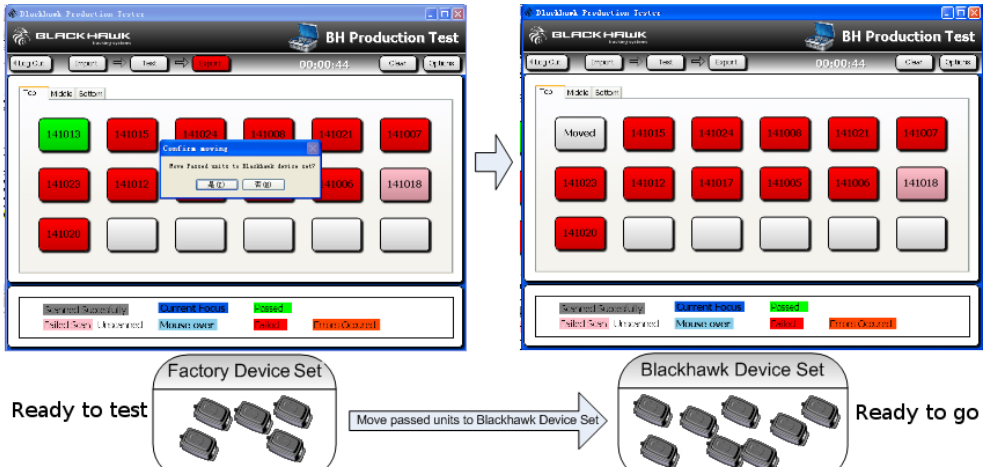


Figure 3.6: Export Blackhawk units to Blackhawk dataset.

## 3.3 Other functionailties

### 3.3.1 Confiuration

To make the application more flexible, we need to add the configurability to the application. For example, if we rent another production line from other factory, we can configure the latitude and longitude range accordingly. The application does supply a setting panel for configuration. Figure 3.7 is the options panel we have currently. I'm using a XML [10] file to save the configuration. The setting can be change by editing the configure file directly.

```
<BlackhawkSettings>
  <Options>
    <FUN Type="String" Value="1" />
    <Min_LAT Type="String" Value="" />
    <Max_LAT Type="String" Value="" />
    <Min_LON Type="String" Value="" />
    <Max_LON Type="String" Value="" />
    <Min_Bettery Type="String" Value="12.0V" />
    <Max_Bettery Type="String" Value="12.0V" />
    <Min_FIX Type="String" Value="3" />
```


Figure 3.7: The option panel

```
  <SWV Type="String" Value="703404009" />
  <Min_SSI Type="String" Value="9" />
  <Min_Battery Type="String" Value="12.0" />
  <Max_Battery Type="String" Value="15" />
  <Database Type="String" Value="E:\stuff\UOA\2008S2\BTech450\app\6-oct-2008\Fac
  <ChkFactoryDB Type="Boolean" Value="True" />
  <Max_Units Type="Int32" Value="2" />
</Options>
<Setup>
  <WebServiceUrl Type="String" Value="http://www.bhtrack.net/BlackhawkAPI/" />
</Setup>
<Login>
  <Username Type="String" Value="fengd" />
  <Password Type="String" Value="password" />
  <RememberMe Type="Boolean" Value="True" />
</Login>
<Appearance>
  <WindowIsMaximized Type="Boolean" Value="False" />
  <WindowTop Type="Int32" Value="102" />
  <WindowLeft Type="Int32" Value="247" />
  <WindowWidth Type="Int32" Value="800" />
  <WindowHeight Type="Int32" Value="600" />
```

```
  </Appearance>
</BlackhawkSettings>
```

C# provide a XMLSettings Class which we can use to read and write this XML file easily. For example, if I want to read the Login username, use following code.

```
// Create a XMLSettings Object
public XMLSettings Settings = new XMLSettings("BlackhawkSettings");
// Load the XML file.
Settings.Load("blackhawkSettings.xml");
// Read the attribute values.
Settings.GetString("Login\tUsername");
```

Before I knowing this XMLSettings Class, I used ConfigReader to read the XML file. Although the ConfigReader is not convenient as XMLSettings, it is also easy to use as well. The main problem is that it can only read the XML file but not write. It requires a ConfigWriter to do the writing which is not that convenient.

The other way to save the settings is saving the setting into the windows registry. I didn't try to read and write windows registry before in a program. But I don't think that it's difficult. I know how to read and write windows registry using the windows command reg.exe. I believe that there is a c# library which can do similar thing as reg.exe. I chose not to use registry because I want to make the application portable. So it can be copied to any computer in the factory including the settings. An advantage of using the registry is that different user can use different settings if we save the settings in the CURRENT_USER node in the registry. But in our application, we don't really need personalised settings.

### 3.3.2 Generating Reports

For debugging purpose, the application should be able to generating reports for the checking result. Then the technician can use this report to debug the faulty Blackhawk units. Although the application can show the problems of the Blackhawk units, but the information shown on the application panel is not printable and not convenient to read directly from the monitor. Generated text format file will be easy to read and can be printed out easily.

It will be easy to write the result report in a plain text file. But the plain text file has no ability to highlighted text which we want to emphasis.

For example, I want to highlight the error value in red, so the technician can found the problem more quickly. I chose to generate the report in XML format because I can easily convert the XML format to other human readable format such as html.

```
<!--report_1-1-2008.xml-->
<Testing Results>
<Unit>
<Number>1</Number>
<Serial No>123456</Serial No>
<IMS>123456789012345</IMS>
<Phone No>0211234567</Phone No>
<Status>Unknown</Status>
<Location>X</Location>
...
</Unit>
<Unit>
<Number>2</Number>
<Serial No>123456</Serial No>
<IMS>123456789012345</IMS>
<Phone No>0211234567</Phone No>
<Status>Faulty</Status>
<Faulty_Location>Sydney</Faulty_Location>
<Faulty_IMS>987654321012345</Faulty_IMS>
...
</Unit>
<Unit>
<Number>3</Number>
<Serial No>123456</Serial No>
<IMS>123456789012345</IMS>
<Phone No>0211234567</Phone No>
<Status>Pass</Status>
<Location>Auckland University</Location>
...
</Unit>
...
</Testing Results>
```

The faulty items will be recorded as an element with a name start with Faulty_ This XML file is not that straight forward for human. But it can

be converted to other file format easily. Figure 3.8 is a example of HTML format.

| Serial No | IMS | Phone No | Status | Location |
|---|---|---|---|---|
| 123456 | 123456789012345 | 0211234567 | Pass | Auckland |
| 123456 | 123456789012345 | 0211234567 | Unkown | X |
| 123456 | 123456789012345 | 0211234567 | Faulty | Auckland |
| --Faulty Location: Sydney | | | | |
| --Faulty IMS: 987654321012345 | | | | |
| 123456 | 123456789012345 | 0211234567 | Unkown | X |
| 123456 | 123456789012345 | 0211234567 | Faulty | Auckland |
| --Faulty IMS: 987654321012345 | | | | |

Figure 3.8: The sample HTML format

In this html format, passed units are highlighted green, faulty units are highlighted yellow. Other units whose report cannot be received are highlighted red. The faulty values are highlighted pink and displayed below the yellow lines. For example, the third row has an IMS number which is 123456789012345, but the unit report has a different value which is 987654321012345. The faulty IMS is displayed below line 3 and has a pink color.

To convert XML format to HTML format, I can either use a XSLT file to do the transfer or use the XmlReader and StreamWriter to generate manually. I would like to write a XSLT file to do this, but the first thing is learning XSLT file first. I am still lack of knowledge about XSLT, I am keen to learn it in the future. Converting XML file to other format is also possible. This functionality has not been implemented yet.

### 3.3.3 Statistic Support

We have not implemented anything about statistical analysis yet. This is an extra functionality which is not very important. We may implement it in the future if necessary.

# Chapter 4

# Manage the project

## 4.1 Version Control

To avoid losing of work, we use Subversion (SVN) [11] to backup the project. After doing a lot of change to the program, I commit the new version of the program to the Blackhawk subversion server. The client we use is TortoiseSVN [12]. SVN is a very good solution for backup. Before using SVN, I had to duplicate the whole project folder to do the backup. This way consumes a lot of disk space. And, this way is not safe enough, because it does not protect from losing of computer or hard drive damage. SVN is intelligent, it knows which files are modified and which files are not. When you commit a new version, it will only upload the modified files. Normally, these are some C# code files that are very small. So, using SVN to do a backup is very fast.

If you have made your code messy, and want your old version back, you can use "SVN update" to get your latest version on the SVN server. You can get any version back by specify the version number if you want.

Figure 4.1 is a snap shot of the TortoiseSVN client. TortoiseSVN is embedded to the context menu. Right click a folder or a file to pop up the context menu, then you can use svn commit you upload the new version to the server, or use svn update to update the local file from the svn server. TortoiseSVN menu gives more functionality if you need.

## 4.2 Software Distribution

To make our application easy to install, we have to pack the application to a setup package. So the factory staff can install the application by follow-

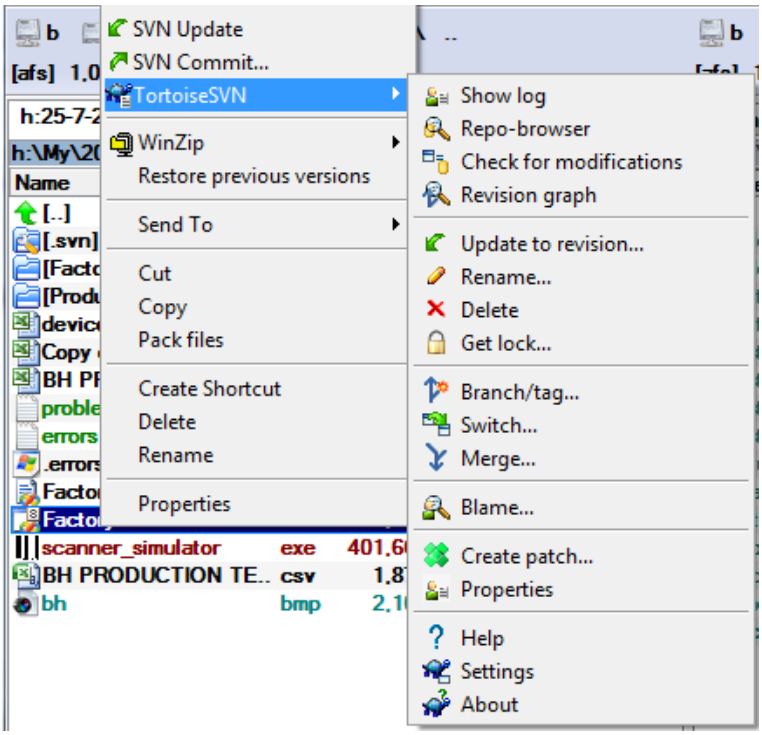


Figure 4.1: The svn client

ing the setup wizard. I know the two most popular distribution software, installshield and wiseInstall. But I didn't know that the visual studio has this kind of tool as well. Actually, we don't need to use any third-party software. Visual studio can do this for us [13]. To build a setup package, start visual studio, create a new project, double click "Other project types", then click "Setup and Deployment". Then click the 'Setup Project" on the right window. In the Setup project, you can customize the application folder, the desktop shortcut, the registry and so on. Press F6 to build the project, there will be two setup files generated, one is exe file, and the other one is MSI file.

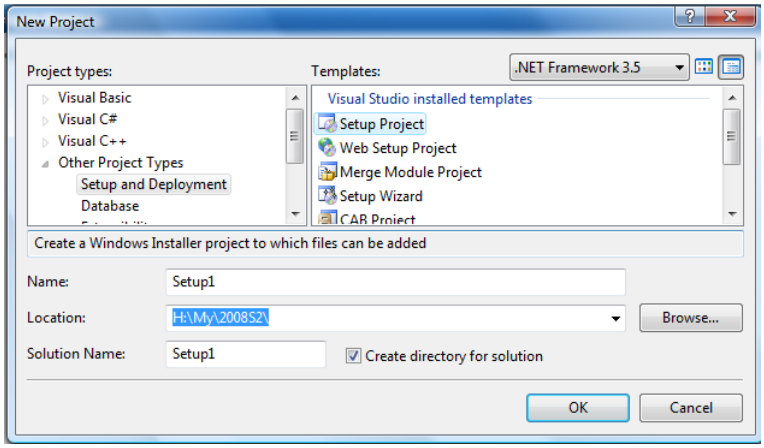


Figure 4.2: The setup project

# Chapter 5

# Conclusion

## 5.1 Further work

The program is almost done. But it has not been used in the real situation. I tested it with dummy data and keyboard. It works well in the lab. As we didn't test it in real world, there should be some bugs we didn't find. We will test the program in real situation and fix bugs found; improve usability, performance and tidy the code. I do think that we can make this program better and easier to use. For example, we can add a drag and drop functionality so that we can swap the Blackhawk location easily. The program still cannot generate report nor do any statistical analysis, if necessary, we will implement these later.

## 5.2 Learning outcomes

### 5.2.1 Writing Software Specification

I wrote a software specification for the project. I didn't do it well. I found that it's very important to write a good software specification before coding. Otherwise, you may code something useless.

### 5.2.2 Think about the usability

The usability is essential for a application. I have been developing this application, so that the program is obvious to me. I can use it without any problem. But it doesn't mean that the application is obvious to the factory user as well. We have to make the software interface as simple as possible. So everyone can use it without doubt.

### 5.2.3 Great skill improvement on developing in visual studio.

At the beginning of the project, I was not good at using visual studio. I had troubles with the layouts, events, debugging and so on. I can now deal with visual studio well. I didn't like developing using visual studio before, being using visual studio for one year, I changed my mind, visual studio is quite good a developing environment.

### 5.2.4 Learn to use APIs.

Blackhawk has many APIs. This APIs provide a way that other companies can cooperate with us easily. APIs also secure the access from outside of the Company.

### 5.2.5 Use subvertion

SVN is a very good way to backup and track the old version of the project. I found it very useful.

### 5.2.6 Make setup package

Visual studio is able to make setup package, we don't need any other third party application to do this. Make good use of Visual studio :).

# Acknowledgements

I would like to say thanks to my Academic Mentor Dr S. Manoharan for providing lots information and advices on this project.

And I would like to thank my Industry Mentor Andrew Radcliffe for giving me this opportunity. He also gave many useful advices and a sample program I can work on. I also would like to thank Michael for his help on using Blackhawk APIs.

# Bibliography


[1] The Blackhawk Website: *http://www.theblackhawk.co.nz/*. Last visit: 22 Oct 2008.

[2] Yunck, T. P., W. G. Melbourne, et al. (1985). "GPS-Based Satellite Tracking System for Precise Positioning." IEEE Transactions on Geoscience and Remote Sensing: 450-457.

[3] The Sim card: *http://en.wikipedia.org/wiki/Sim_card*. Last visit: 22 Oct 2008.

[4] Cai, J. and D. J. Goodman (1997). "General packet radio service in GSM." Communications Magazine, IEEE 35(10): 122-131.

[5] Rahnema, M. (1993). "Overview of the GSM system and protocol architecture." Communications Magazine, IEEE 31(4): 92-100.

[6] Kihara, M. and Y. Nat. Defense Acad. (1994). "Study of a GPS satellite selection policy to improve positioningaccuracy." 267-273.

[7] The Http Get method: *http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html*. Last visit: 22 Oct 2008.

[8] Anley, C. (2002). "Advanced SQL Injection In SQL Server Applications." White paper, Next Generation Security Software Ltd.

[9] Möller, B. and C. Dahlin (2006). A First Look at the HLA Evolved Web Service API.

[10] Bray, T., J. Paoli, et al. (2000). "Extensible Markup Language (XML) 1.0." W3C Recommendation 6. Last visit: 22 Oct 2008.

[11] Collins-Sussman, B., B. W. Fitzpatrick, et al. (2004). Version Control with Subversion, O'Reilly Media, Inc.

[12] Küng, S., L. Onken, et al. (2008). TortoiseSVN A Subversion client for Windows.

[13] How to create a Setup package by using Visual Studio .NET: *http://support.microsoft.com/kb/307353*. Last visit: 22 Oct 2008.