University of Southern Queensland

Faculty of Engineering & Surveying

Human Interface Device (HID) keyboard application for Android

A dissertation submitted by

F. Houweling

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Engineering, Computer Systems Enginnering

Submitted: January, 2013

Abstract

Human Interface Device (HID) keyboard application for Android is a software application for the Android operating system , it is endeavouring to improve usability of Android mobile phones and tablets, by providing an additional user data input method, through the integration of Bluetooth keyboards. This is to be accomplished through a software solution bridging the missing link between the operating systems native Bluetooth protocol stack and the operating systems Input Method Editor (IME). The IME is the primary method that the users application receives textual input from the user. The application is named *Always Blue Ready*. The application must:

1. Be able to search, discover, pair and connect with Bluetooth kseyboards.
2. Decode keystrokes and make them available for the operating system.
3. Improve the usability of Android devices through the use of the external Bluetooth keyboard.
4. Must strive to protect user data, or at least inform the user of potential risks.

Benefits for the end user from the implementation of this project will include:

1. Significant improvement in user typing speed.
2. Valuable screen real estate returned to the applications interface.
3. Improve user ergonomics of phones and tablets.

Current project information is published at the website: `http://www.houweling.com.au/abr/`

University of Southern Queensland

Faculty of Engineering and Surveying

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Prof F Bullen

Dean

Faculty of Engineering and Surveying

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

F. Houweling

0050058934

Signature

3 January 2013

Date

Acknowledgments

Thank you to my Family, without their support I would not have accomplished anything, Katherine, Emily, Katie, Aimee, Bailey, Noah and Logan. An extra big thank you to Katherine, Katie and Josh Hurn for proof reading my draft.

I'd also like to acknowledge and thank my supervisor, Dr Alexander Kist, whom gave me the space to get on with the task and took the time to help when I asked.

Thanks also to Dr John Leis, your LaTeX dissertation template saved a lot of formatting time.

Finally I'd like to thank a good friend John Palmer, who's caring attitude, proof reading of my draft and regular follow up helped in so many ways.

F. Houweling

University of Southern Queensland

January 2013

Contents

List of Figures

List of Tables

Nomenclature

Table 1: Nomenclature

Term	**Definition**
Android	Open source operating system by Google Inc.
API	Application Programming Interface
Bluetooth	Wireless communications protocols and standards, relating to personal area networks.
C/C++	Two separate but related programming languages, **C** is procedural in focus and **C++** is object orientated.
Davlik	Androids Virtual Machine.
Eclipse	IDE for Java development.
IDE	Integrated Development Environment.
IME	Input Method Editor.
Java	Java is a programming language.
L2CAP	*Bluetooth:* Logical Link Control and Adaptation Protocol.
HCI	*Bluetooth:* Host Controller Interface.
HID	*Bluetooth:* Human Interface Device.
NDK	Native Development Kit, used when performance counts, allowing the use of C/C++ code.
OBEX	*Bluetooth:* Object Exchange.

continued ...

. . . continued

Term	Definition
OS	Operating System.
PAN	*Bluetooth:* Personal Area Network.
QoS	Quality of Service.
RFCOMM	*Bluetooth:* Radio frequency communications.
SDK	Software Development Kit.
SDP	*Bluetooth:* Service Discovery Protocol.
SQL	Structured Query Language.
SQLite	the SQL database that is available on all Android devices.[1]
SVN	Subversion, a version control system for electronic files.
TCP/IP	Transmission Control Protocol/Internet Protocol.
thread	A separate task belonging to a process that executes independently of the process and other tasks and is OS Managed.
USB	Universal Serial Bus.
UUID	*Bluetooth:* Universally Unique Identifier.
VM	Virtual Machine.
WAP	*Bluetooth:* Wireless Access Protocol.
XML	Extensible Mark-up Language.

[1] See: `http://www.sqlite.org/`

Chapter 1

Introduction

This dissertation documents progress of the project Human Interface Device (HID) keyboard application for Android, the projects purpose is to allow the user to input typed data wirelessly through a physical keyboard for use in Android applications using Bluetooth as the wireless protocol. This feedback relationship is depicted in Figure 1.1.

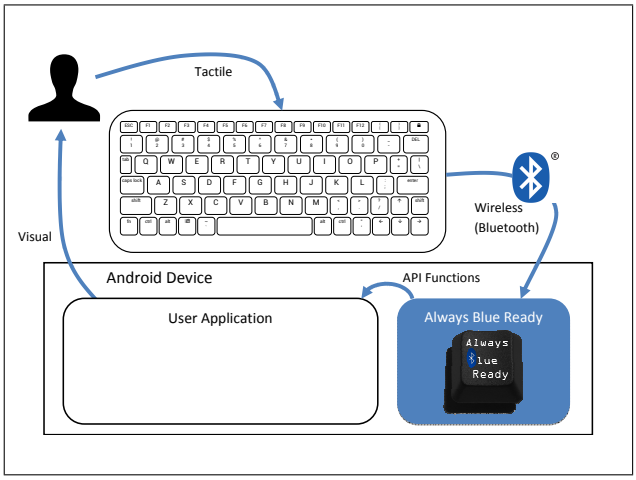


Figure 1.1: Projects role in the user feedback loop.

Modern portable computing devices such as smart-phones and tablets rarely feature a hardware keyboard, including many versions of the Android operating system. The users current solution for data input is to use various on screen keyboards similar to the one depicted in Figure 1.3a (on Page 4), this places limitations on the usability of these devices for anything other than casual use. Another option is to install

one of a handful of third party add-on software.

The project has been named Always Blue Ready and it's application icon is depicted in Figure 1.2a. This project will improve usability of Android mobile phones and tablets. This improvement will be realised through providing an additional user data input path, through the integration of Bluetooth keyboards such as the one depicted[1] in Figure 1.2b.

(a) Application icon. (b) Typical portable Bluetooth keyboard.

Figure 1.2: Application icon and Typical Bluetooth keyboard.

This is to be accomplished through a software solution bridging the missing link between the operating systems native Bluetooth support and the operating systems Input Method Editor (IME) system, through the implementation of the HID layer. Factors contributing to the lack of Bluetooth keyboard support include:

1. Early Android devices are missing the HID portion of the Bluetooth protocol stack.
2. Some manufactures tamper with the stock Bluetooth protocol stack (likely to reduce complexity), thereby breaking functionality.
3. Immaturity of the Android Bluetooth protocol stack, as is evidenced when viewing early android operating system source code.

The user encounters great difficulty using a Bluetooth keyboard when their device is impacted by the issues mentioned above. Use of third party software can provide a solution. Despite the fact that most Bluetooth enabled Android devices include the

[1] Bluetooth Keyboard Make: Fission, Model: S-KW427IBS

ability to discover and pair with Bluetooth keyboards, effected devices are unable to natively maintain a connection with the paired keyboard and are unable to receive user input.

Current software attempting to fill this gap, and available on the android market place is either:

1. Too expensive[2].
2. Does not work reliably.
3. Frequently crashes (force close).
4. Suffers from keying input faults (wrong key mappings).
5. Requires hacking the phone to gain root access.

The author recently discovered that more modern versions[3] of the Android operating system can now connect with Bluetooth keyboards. This project is not made redundant by this discovery, due to the opportunity to exploit gaps in functionality between existing solutions and Androids new native support.

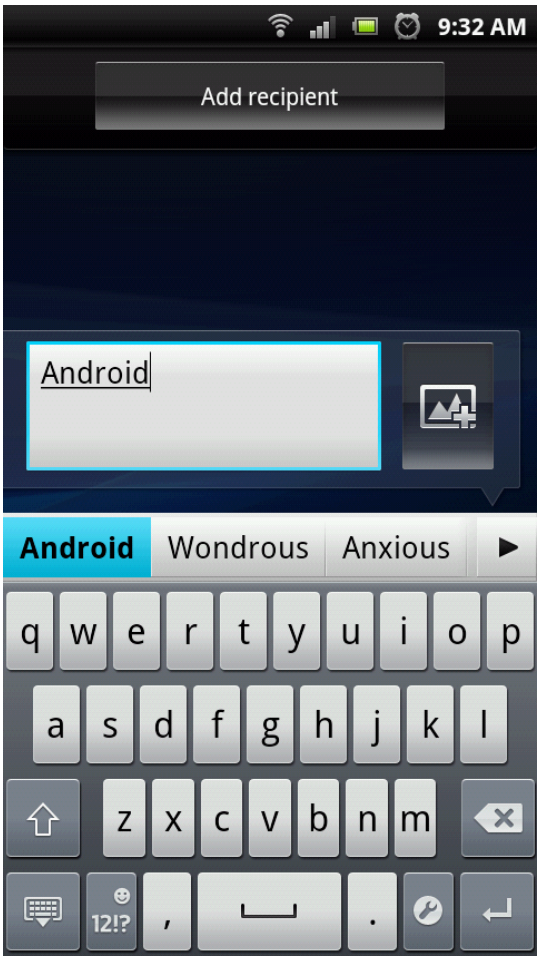
The projects goal is to transition the users experience from using the stock input method as depicted in Figure 1.3a to the scenario depicted in Figure 1.3b.

Benefits for the end user from the implementation of this project will include:

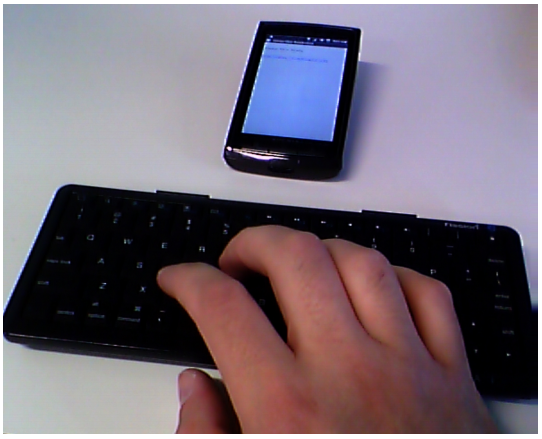
1. Significant improvement in user typing speed.
2. Valuable screen real estate returned to the applications interface.
3. Improve user ergonomics of phones and tablets.

[2]BlueInput (`http://www.teksoftco.com/`) is sold for €9.99 EUR = $12.51 AUD (as at 11th October 2012).

[3]Version 4.0.4, also known as Ice Cream Sandwich (ICS)



(a) Example of a standard Input Method Editor (IME).



(b) Using a Bluetooth keyboard to enter text.

Figure 1.3: Different data input methods.

1.1 Overview of the Dissertation

This dissertation is organised as follows:

Chapter 2 - Literature Review and Bluetooth Keyboard Software Review reviews the literature available relating to key technologies utilised, as well as a review of existing solutions. (p. 7)

Chapter 3 - Application Design discusses the application design. (p. 17)

Chapter 4 - Android Applications introduces the development process for the android platform, discusses the basic structure of android applications and includes a discussion relating to the history of the Android operating system. (p. 35)

Chapter 5 - Bluetooth details the relevant parts of the Bluetooth protocol stack. (p. 47)

Chapter 6 - Application Testing details the projects testing activities and methodology. (p. 55)

Chapter 7 - Conclusions and Further Work provides details relating to work completed, conclusions for the project and details further work. (p. 65)

Appendix A - Project Specification is the project specification. (p. 73)

Appendix B - Subversion Change Log contains the subversion message log for the source code branch of the project. (p. 75)

Appendix C - Application Source Code contains the entire project source code current to 30th December 2012. (p. 89)

Appendix D - Test Cases contains the Test Cases at time of publication (30th December 2012). (p. 145)

Chapter 2

Literature Review and Bluetooth Keyboard Software Review

This chapter discusses the following key concepts.

1. A review of the available literature.
2. Comparison of existing third party solutions.

2.1 Literature Review

This project will need to utilise the following key resources:

1. Bluetooth resources are required because the project is dependant on the wireless communications channel that the keyboard data is transported through.
2. USB-HID will be required because the Bluetooth encapsulated keyboard data will be encoded with this standard.
3. Android resources are required to understand the architectural implications and

coding standards that are required to develop Android applications.

A review of available literature from `http://scholar.google.com/`, The USQ Library (Including Electronic Resources and Standards), Internet Google Keyword search, Ipswich Public Library and Amazon Books reveals that there are many resources available discussing individual aspects, but no single resource that encompass the breadth of topics that this project will require, especially in sufficient detail.

An overview of the suitable literature identified to date follows:

2.1.1 Bluetooth

Bluetooth is a group of standards and specifications that are maintained by the Bluetooth SIG (Bluetooth SIG 2012*a*). Members of the Bluetooth SIG have access to these specifications. These standards and specifications contain detailed technical information including Bluetooth protocols, data formats and device profiles[1]. The Bluetooth HID encapsulates the USB-HID specification. Most of the readily available textbooks discussing Bluetooth are antiquated; authors such as Huang (2007), Kumar et al (2004) and Hopkins (2003) discuss Bluetooth at various conceptual levels, then progress to specific implementations that do not at all resemble the Android Bluetooth API. These resources are useful only in providing grounding in the concepts involved with Bluetooth connectivity. Specific examples of Android Bluetooth applications are published on the *The official site for Android developers.* (Google Inc. 2012) website, however they are just examples of Peer to Peer applications over RFCOMM connections, typically chat oriented and documentation is limited to a description of the example as implemented. This resource is useful for informing best coding practice and style. Most general Android programming books such as Darcey (2011) and Steele (2011) are limited in their

[1]Bluetooth profiles include Human Interface Device (HID) and Service Discovery application Profile (SDP)

discussion of Bluetooth programming.

2.1.2 Universal Serial Bus - Human Interface Device (USB-HID)

Traditionally keyboards are physically connected to their host, most modern keyboards of this nature connect through the USB interface, and *www.usb.org Web site*(USB Implementers Forum, Inc 2012*b*) maintain and publish the USB-HID specification(Universal Serial Bus (USB) 2012) which details the protocols and data formats that hosts[2] and keyboards implement to ensure interoperability is maintained.

2.1.3 Android Programming

The most accurate and up to date information on Android programming is published through the *The official site for Android developers.* (Google Inc. 2012) website, including technical articles, API reference and example applications. Many printed texts discussing Android programming exist, most follow similar patterns:

1. Introducing Android.
2. Explaining the Android architecture.
3. User Interface Design.
4. Discussing the messaging system called Intents.
5. Introduction to common API's.
6. Data storage and services.
7. Threading.
8. Services.
9. Application Publication.

[2]Hosts refers to Software and Hardware that is responsible for communicating with the connected USB device.

This is often accomplished through the development of an example application. Often it is an application derived from some pre-existing example published elsewhere in the public domain. Some of the better quality texts include *Android Wireless Application Development* (Darcey & Conder 2011), *Beginning Android 4* (Murphy 2011), *Beginning Android 4* (Allen 2012),*Beginning Android Application Development* (Lee 2012) and *The Android Developers Cookbook* (Steele & To 2011).

2.1.4 Software Testing

Many books relating to testing exist, the primary resource utilised for this project is *Software Engineering and Testing an introduction.* (Agarwal, Tayal & Gupta 2008). This book is presented in two parts, the first part is generally testing focused coverage includes the topics:

1. Software life-cycle.
2. Quality.
3. System design.
4. Software testing.

The second part focuses on languages such as visual basic which is not relevant to this project.

Other resources include *The Art of software testing*(Myers, Badgett, Thomas & Sandler 2004), this book provides an introduction to software testing principles, including:

1. The concept of "human testing", which is equivalent to white box testing.
2. Test case design.
3. Unit testing.
4. High order testing.
5. Debugging.

The text *principles of software testing.*(Burnstein 2003) Includes coverage of the following topics:

1. An introduction to testing as part of software engineering.
2. Introduction to testing fundamentals.
3. And then begins discussing defects, including defects in specifications and requirements.
4. Defects in design.
5. Coding defects.
6. Testing defects.

2.2 Bluetooth Keyboard Software Reviews

A competitive review of existing solutions has been completed; the results are tabulated in Table 2.1 on page 13. It is the intention to develop a solution that exploits gaps in functionality, cost and ease of use compared to the solutions that currently exist. Table 2.1, Table 2.2 and Table 2.3 beginning on page 13 summarises some of the reviewed characteristics for the competitive applications, some of these characteristics deserve further explanation:

Permissions required refers to Android Application permissions that are requested from the user at time of install, without these permissions the application could not function and cannot be installed, undeclared application permissions will cause the operating system to refuse successful execution of the privileged functions. Permissions also effect the users psychological level of trust in the application and can have a negative impact on user acceptance, especially if out of place permissions are not justified to the user.

Target OS refers to the supported API level as described in Table 4.2 - Android Versions on page 45.

The average user rating, $\frac{5Star}{1Star}$ ratio and the votes section of the table is attempting to gauge the collective end-user opinion of the competitive product.

Table 2.1: Existing 3rd Party Solution Review

Product Name	**BlueInput**	**BlueKeyboardJP**	**Bluetooth Keyboard Easy Connect**
Cost	€9.99	Free (+Adds) $2.00 (-Adds)	Free
Needs Root Access	No	No	Yes
Target OS	7	7	8
App Size (MB)	0.304	4.5	0.43
Average User Rating (Stars)	2.9	3.2	3.3
Downloads (x1000, thousands)	10 to 50	50 to 100	50 to 100
5 Stars	57	200	97
1 Star	73	176	70
$\frac{5Star}{1Star}$ **Ratio**	0.781	1.136	1.386
Votes	184	636	240
Works on available hardware?	$\frac{2}{3}$	1st time Yes 2nd time No	No
Permissions Required			
ACCESS_COARSE_LOCATION	-	yes	-
ACCESS_FINE_LOCATION	-	yes	-
BIND_DEVICE_ADMIN	-	-	yes
BIND_INPUT_METHOD	yes	-	-
BLUETOOTH	yes	yes	yes
BLUETOOTH_ADMIN	yes	yes	yes
INTERNET	yes	yes	-
RECEIVE_BOOT_COMPLETED	-	-	yes
SET_ACTIVITY_WATCHER	-	-	yes
VIBRATE	-	yes	-
WRITE_EXTERNAL_STORAGE	-	-	yes

Table 2.2: Existing 3rd Party Solution Developer URL's

Product Name	**Developer URL**
BlueInput	`http://www.teksoftco.com`
BlueKeyboardJP	`http://blog.elbra.in`
Bluetooth Keyboard Easy Connect	`http://forum.xda-developers.com/showthread.php?t=925474`

Table 2.3: Existing 3rd Party Solution Comments and Observations

Product Name	**Comments and Observations**
BlueInput	Too expensive, users complain of poor customer service.
BlueKeyboardJP	Low reliability, key stroke errors, privacy concerns (location permissions).
Bluetooth Keyboard Easy Connect	Immature application and requires device to be hacked for root access.

2.3 Chapter Summary

This chapter discussed the following concepts:

1. Detailed the literature review relating to relevant project topics.
   (a) Discussion relating to Bluetooth literature, including published standards and specifications.
   (b) Android programming resources.
   (c) The Universal Serial Bus - Human Interface Device (USB-HID) specifications and standards.
   (d) Testing literature review.
2. Review of existing software solutions.

Chapter 3

Application Design

This chapter discusses the intended design of the final application, including a breakdown of the major elements, key concepts are:

1. Application structure.
2. State based machine.
3. Utilisation of the Android native database
4. Input Method Editor (IME).
5. User interface
6. Threads and long running operations.

3.1 Application Structure

The general architecture of the application is represented by Figure 3.1 - Application block diagram. The application is required to utilise the Bluetooth API to interface with the external input device, in this case a Keyboard. The application is required to manage typical Bluetooth activities, especially:

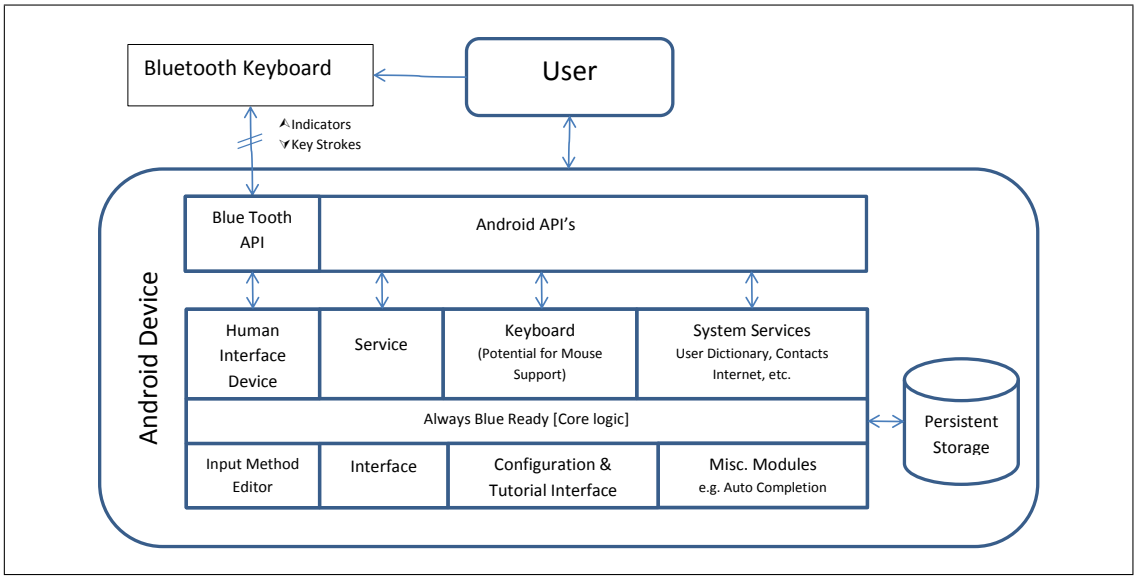


Figure 3.1: Application block diagram.

1. Searching for Bluetooth devices.
2. Interrogating devices to determine if the device offers HID services.
3. If not yet paired, pairing to the selected device.
4. Connecting to the keyboard.

Once a connection is established the application will decode keystrokes received using the HID protocol described in *HID Information* and *Human Interface Device Profile (HID)* (USB Implementers Forum, Inc. 2012*a*, Bluetooth SIG 2012*c*) and inject them into the operating systems keystroke queue. The application will require an interface to allow the user to manage and configure the external Bluetooth devices and view the application tutorial.

3.2 Broad Aims

Broadly the application must satisfy a few other generic constraints. Due to their generic nature the success in these criteria will be somewhat subjective.

3.2.1 Small Memory Footprint

Resources in mobile devices are at a premium, smaller applications have a number of significant benefits.

1. Less likely to be killed off by Android resource management.
2. Greater user acceptance.
3. Faster user downloads.

The application memory footprint will be compared to alternative solutions to ensure that the project is not worse than competing solutions. Additionally the application behaviour will be tested in simulated low memory conditions.

3.2.2 Robust and Reliable

The application should be designed to be forgiving of varied hardware, unexpected user use patterns and unexpected situations. This means a greater level of in code error checking, leading to improved recovery back to the normal application state, as well as an in application tutorial to assist the user to quickly synchronise their attempted use with the applications designed usage patterns. The final quality will be further enhanced through undertaking various testing methods, with varied hardware combinations. Success in this criterion will be measured through user Beta testing.

3.3 Specific Requirements

These requirements are mandatory for the Application before it can be published to the Android market place. The application must:

1. Be able to search, discover, pair and connect with Bluetooth keyboards.

2. Be required to decode keystrokes and make them available for the operating system.
3. Improve the usability of Android devices through the use of the external Bluetooth keyboard.
4. Must strive to protect user data, or at least inform the user of potential risks.

3.4 Time Permitting

These goals are desirable features that add value, but in themselves are not critical to the end user experience, they may be considered future design goals.

3.4.1 Predictive Text

Accelerate users typing efforts by using predictive text for use in auto completion hints. This could be facilitated with a combination of the following methods:

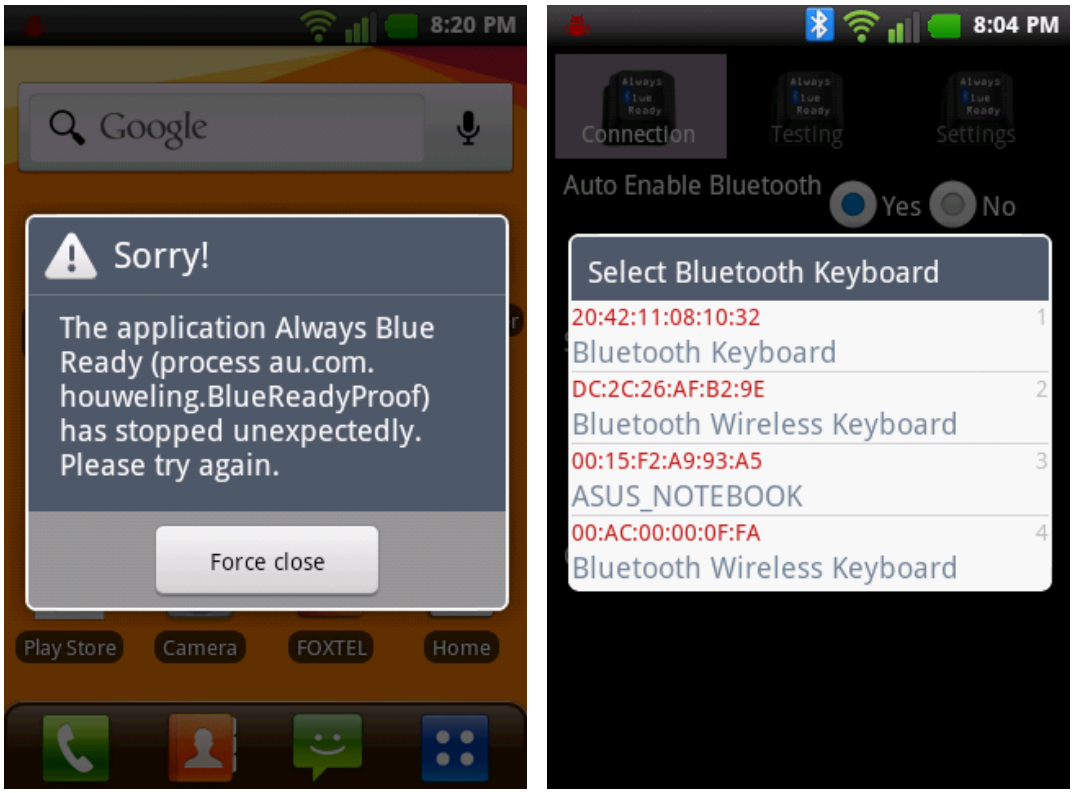
1. Dictionary.
2. User Dictionary.
3. Learning.
4. Calender.
5. Contacts.

3.4.2 Active Battery Management

Unfortunately the very act of using Bluetooth will result in a reduction in the duration the host device is able to remain functioning correctly. By making the application aware of the systems battery state there is an opportunity to modify the applications behaviour to improve the battery life.

3.5 Test Application

A test application is designed to act as a proof of concept to confirm technical viability. In addition this test application serves to build key skills, learn specific coding methods, gain knowledge of system issues and provide a testing framework to interrogate and interact with external keyboards. The intention is to use reusable coding to allow an acceleration of the final application to be realised. An example of an issue identified



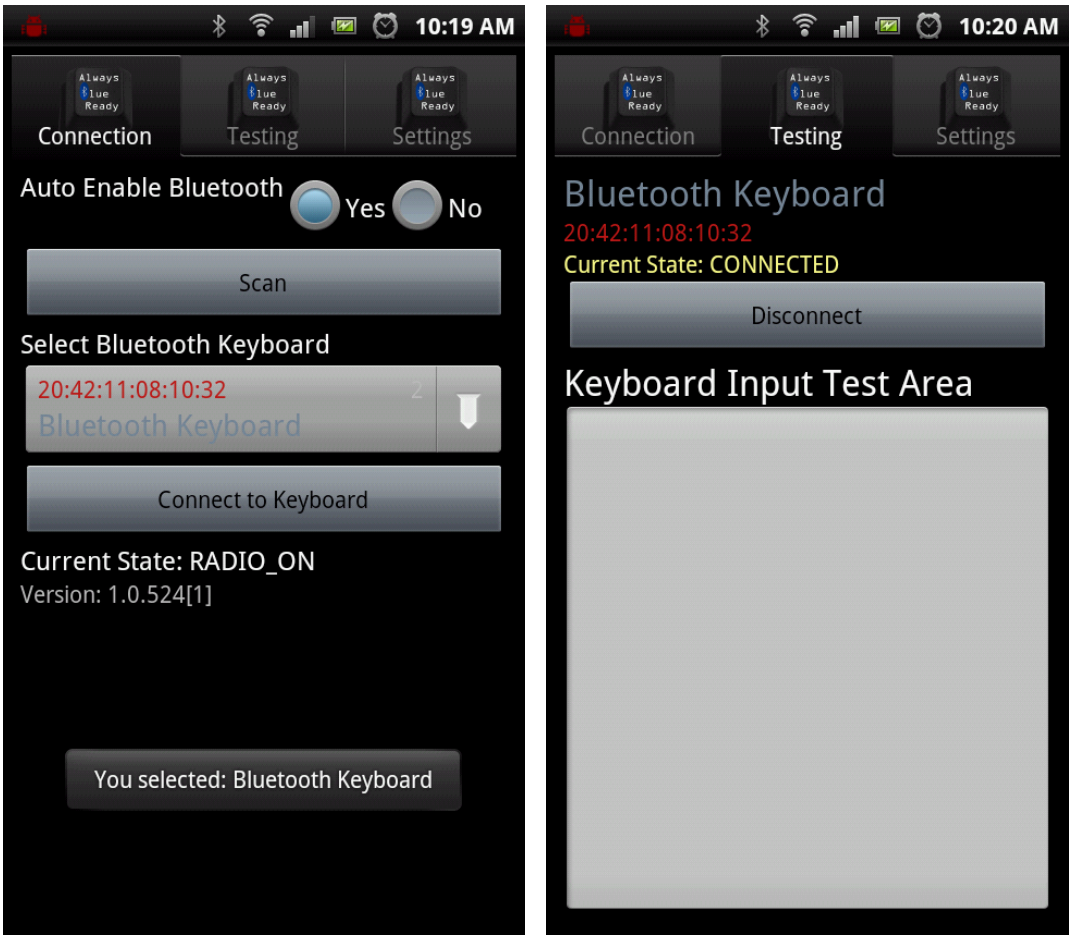
(a) Typical force close error message. (b) Open spinner drop down box.

Figure 3.2: Examples of issues encountered.

relates to long running tasks not being permitted to execute in the User Interface process thread, doing so will result in a *force close* error message as depicted in Figure 3.2a.

The solution is to implement a separate thread to handle long running tasks. This

does have a side effect, as the thread is not permitted to update the user interface, this results in a requirement to pass messages between the user interface and the long running thread. Figure 3.3a and Figure 3.3b depict recent screenshots from the current test application.



(a) Main application dialogue after selecting a keyboard in the spinner.

(b) The testing screen for the display of keystroke data.

Figure 3.3: Application dialogues.

Another issue that consumed significant time is the design of the Android spinner form element, the Figure 3.2b - Open spinner drop down box. This figure depicts four instances of the keyboard entries. The layout of the spinner entries are defined using an

XML file and is processed by the Android framework, if the design of the entry contains a *clickable=true* property as depicted in Listing 3.1 on line 13. The presence of this property prevents the ability to select any entry.

Listing 3.1: Defective spinner entry due to *clickable* element

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent"
6      tools:context=".TestActivity" >

9      <ToggleButton
10         android:id="@+id/onAirIndicator"
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:clickable="true"
14         android:textOff="@+id/Off"
15         android:textOn="@+id/On" />

18     <TextView
19         android:id="@+id/deviceName"
20         android:layout_width="wrap_content"
21         android:layout_height="wrap_content"
22         android:layout_alignParentLeft="true"
23         android:layout_alignParentRight="true"
24         android:layout_below="@+id/onAirIndicator"
25         android:text="@string/KeyboardName"
26         android:textAppearance="?android:attr/textAppearanceMedium"
27         android:textColor="@drawable/grey" />

30     <TextView
31         android:id="@+id/deviceAddress"
32         android:layout_width="wrap_content"
33         android:layout_height="wrap_content"
34         android:layout_alignParentRight="true"
35         android:layout_alignParentTop="true"
36         android:layout_toRightOf="@+id/onAirIndicator"
37         android:text="@string/KeyboardAddress"
38         android:textAppearance="?android:attr/textAppearanceSmall"
39         android:textColor="@drawable/red" />
41 </RelativeLayout>
```

The final definition of this entry is defined in Appendix C - Application Source Code in Listing C.15 on page 141

Development of the test application will finalise once reliable Bluetooth connection to a keyboard can be maintained, displaying input key codes in a log window. Reaching this milestone will mark the commencement of development for the final application.

(a) Application icon.

(b) Collection of typical application icons.

Figure 3.4: Application icons.

3.6 Final Application

The final distributable application will be developed after development of the test application concludes. This will incorporate lessons learned from the test application and reuse source code from the test application. The application will be installable and executable just like any other application the user may have installed, the application icon as depicted in Figure 3.4a will be available on the users home screen and may appear similar as depicted in Figure 3.4b.

Table 4.2 on page 45 identifies that at the time of writing 98.8% of devices support the API level 7 or higher. Therefore this project will target the minimum API level of 7, if

a technical limitation is encountered that can be resolved by lifting the minimum target to 8 then the reduction of the potential market share to 92.2% is tolerable.

3.7 State Based Machine

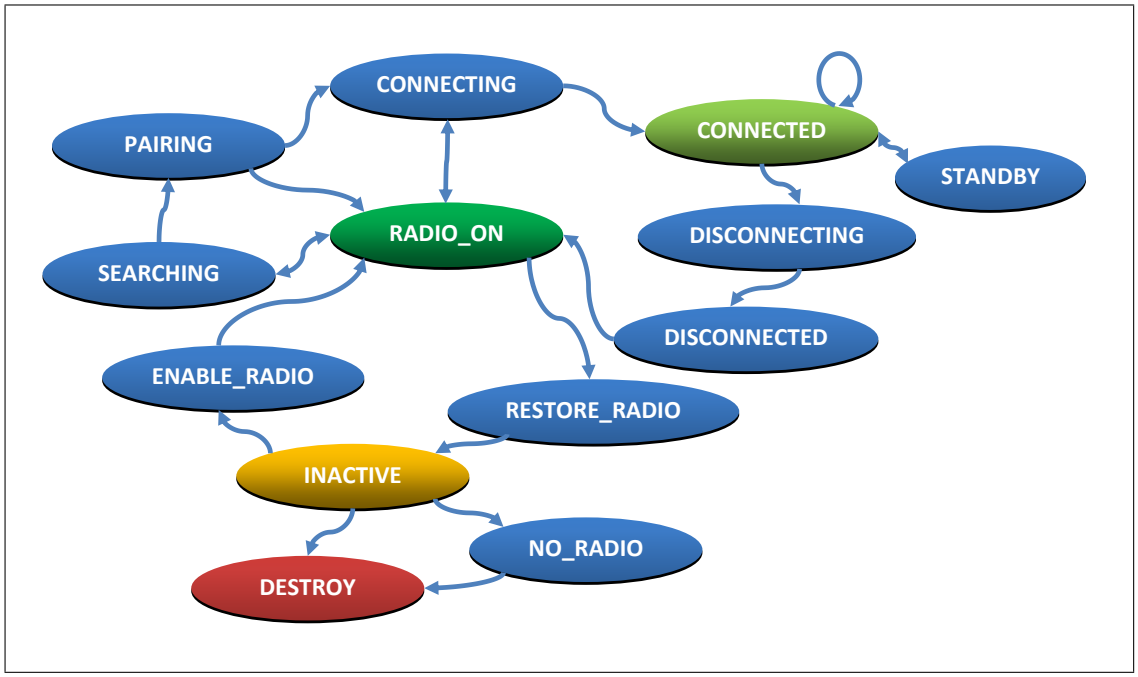


Figure 3.5: State diagram.

The application is based on a finite state based machine, the current design of this state based machine is depicted in Figure 3.5 and tabulated in Table 3.1. The State Based Machine is implemented in the file BlueReadyStateBasedMachine.java, this class maintains a state lookup table for selecting the next logical state. This next logical state is referred to as the transition state. The State Based Machine maintains a *current* and *target* state variable defined by an enumeration of the valid system states. The applications background thread calls the state based machines process() function once per iteration. This allows the state based machine to perform long running operations, such as waiting for system events to occur.

The purpose of process() function is two fold:

1. To initiate transitions between states through a call to the transition() function.
2. Process state based logic.

The first task is to compare the CurrentState to the TargetState variable, if they do not match, the state transition() function is executed which returns true upon a successful transition. The process() function is also designed to process small state based tasks each iteration. The transition() function searches the state transition table using current and target states as a composite key, if a matching entry is located and it is enabled the current state is changed to the transition state. If no entry is located a direct transition will be assumed.

Some state transitions will be user initiated and others transition automatically based on system events or logical transitions.

The implementation of the applications State Based Machine is documented in Appendix C - Application Source Code, Listing C.10 - State Based Machine Implementation on page 124

Table 3.1: State Transition Table, part (a)

TRANSITION STATE		Target State			
		INACTIVE	RADIO_ON	SEARCHING	PAIRING
Current State	INACTIVE	*INACTIVE*	ENABLE_RADIO	ENABLE_RADIO	ENABLE_RADIO
	RADIO_ON	RESTORE_RADIO	*RADIO_ON*	*SEARCHING*	SEARCHING
	SEARCHING	RADIO_ON	*RADIO_ON*	*SEARCHING*	*PAIRING*
	PAIRING	RADIO_ON	*RADIO_ON*	RADIO_ON	*PAIRING*
	CONNECTING	RADIO_ON	*RADIO_ON*	RADIO_ON	RADIO_ON
	CONNECTED	DISCONNECTING	DISCONNECTING	DISCONNECTING	DISCONNECTING
	DISCONNECTING	DISCONNECTED	DISCONNECTED	DISCONNECTED	DISCONNECTED
	DISCONNECTED	RADIO_ON	*RADIO_ON*	RADIO_ON	RADIO_ON
	DESTROY				
	NO_RADIO				
	STANDBY	CONNECTED	CONNECTED	CONNECTED	CONNECTED
	ENABLE_RADIO	RADIO_ON	*RADIO_ON*	RADIO_ON	RADIO_ON
	RESTORE_RADIO	*INACTIVE*	INACTIVE	INACTIVE	INACTIVE

TRANSITION STATE		Target State			
		CONNECTING	CONNECTED	DISCONNECTING	DISCONNECTED
Current State	INACTIVE	ENABLE_RADIO	ENABLE_RADIO	ENABLE_RADIO	ENABLE_RADIO
	RADIO_ON	*CONNECTING*	CONNECTING	CONNECTING	CONNECTING
	SEARCHING	PAIRING	PAIRING	PAIRING	PAIRING
	PAIRING	*CONNECTING*	CONNECTING	RADIO_ON	RADIO_ON
	CONNECTING	*CONNECTING*	*CONNECTED*	CONNECTED	CONNECTED
	CONNECTED	DISCONNECTING	*CONNECTED*	*DISCONNECTING*	DISCONNECTING
	DISCONNECTING	DISCONNECTED	DISCONNECTED	*DISCONNECTING*	*DISCONNECTED*
	DISCONNECTED	RADIO_ON	RADIO_ON	RADIO_ON	*DISCONNECTED*
	DESTROY				
	NO_RADIO				
	STANDBY	CONNECTED	*CONNECTED*	CONNECTED	CONNECTED
	ENABLE_RADIO	RADIO_ON	RADIO_ON	RADIO_ON	RADIO_ON
	RESTORE_RADIO	INACTIVE	INACTIVE	INACTIVE	INACTIVE

Table 3.2: State Transition Table, part (b)

TRANSITION STATE		Target State			
		DESTROY	NO_RADIO	STANDBY	ENABLE_RADIO
Current State	INACTIVE	*DESTROY*	*NO_RADIO*	ENABLE_RADIO	*ENABLE_RADIO*
	RADIO_ON	RESTORE_RADIO	RESTORE_RADIO	CONNECTING	RESTORE_RADIO
	SEARCHING	RADIO_ON	RADIO_ON	PAIRING	RADIO_ON
	PAIRING	RADIO_ON	RADIO_ON	CONNECTING	RADIO_ON
	CONNECTING	RADIO_ON	RADIO_ON	CONNECTED	RADIO_ON
	CONNECTED	DISCONNECTING	DISCONNECTING	*STANDBY*	DISCONNECTING
	DISCONNECTING	DISCONNECTED	DISCONNECTED	DISCONNECTED	DISCONNECTED
	DISCONNECTED	RADIO_ON	RADIO_ON	RADIO_ON	RADIO_ON
	DESTROY	*DESTROY*			
	NO_RADIO	*DESTROY*	*NO_RADIO*		
	STANDBY	CONNECTED	CONNECTED	*STANDBY*	CONNECTED
	ENABLE_RADIO	RADIO_ON	RADIO_ON	RADIO_ON	*ENABLE_RADIO*
	RESTORE_RADIO	INACTIVE	INACTIVE	INACTIVE	INACTIVE

TRANSITION STATE		Target State
		RESTORE_RADIO
Current State	INACTIVE	ENABLE_RADIO
	RADIO_ON	*RESTORE_RADIO*
	SEARCHING	RADIO_ON
	PAIRING	RADIO_ON
	CONNECTING	RADIO_ON
	CONNECTED	DISCONNECTING
	DISCONNECTING	DISCONNECTED
	DISCONNECTED	RADIO_ON
	DESTROY	
	NO_RADIO	
	STANDBY	CONNECTED
	ENABLE_RADIO	RADIO_ON
	RESTORE_RADIO	*RESTORE_RADIO*

3.8 Database

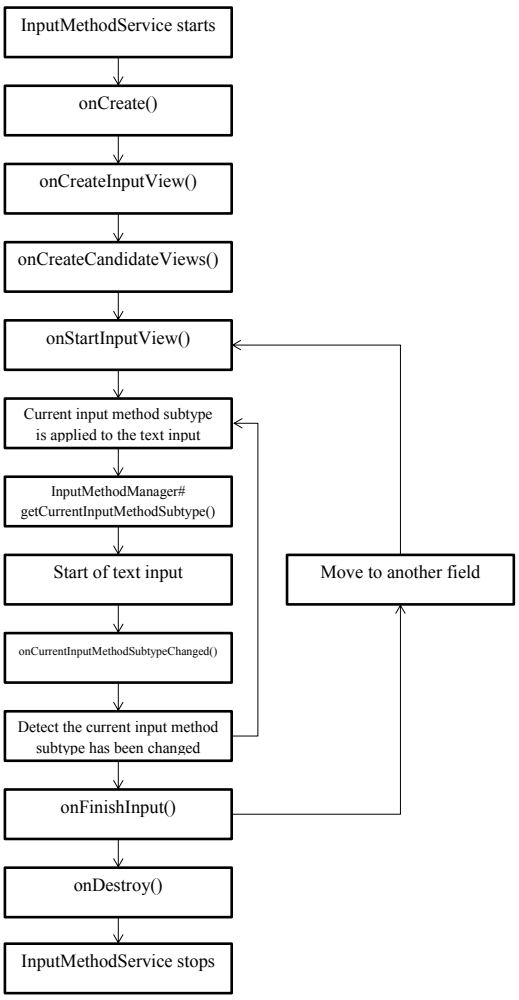
The application utilises Androids native support for the SQLite database for the long-term storage of Bluetooth device information, including:

1. Device MAC address.
2. Device human readable name.
3. Timestamps.
4. Various flags.

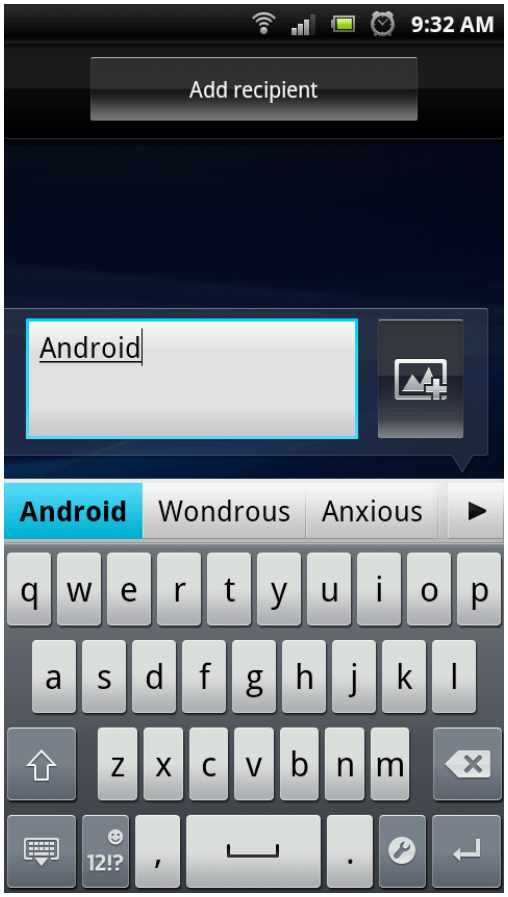
This allows the applications form fields such as the device selection spinner to populate directly from the database adapter. Additionally the application gains access to Bluetooth device data from previous executions, before the Bluetooth radio is enabled, which otherwise requires the application to wait for the radio to enter the enabled state.

3.9 Input Method Editor (IME)

Android devices use a system called the Input Method Editor (IME) to receive user typed information, typically this is displayed as an on-screen keyboard similar to Figure 3.6b



(a) Input method editor life-cycle(android.com 2012*c*).



(b) A typical Input Method Editor (IME).

Figure 3.6: Android IME life-cycle and An example of a typical IME.

Figure 3.6a details the typical life cycle of the IME. The application will require further development to implement the IME service, further design in this area is limited due to

technical issues encountered relating to Bluetooth HID communications.

3.10 User Interface

The final application will have the following user interfaces:

1. Application Tutorial.
2. Input Method Editor (IME).
3. Installation assistant to handle initial install tasks, including:
   (a) Enable the IME.
   (b) Selection of the IME as the default input method.
   (c) Initial application configuration.
4. Configuration interface to enable the user to alter application settings such as:
   (a) Bluetooth keyboard pairing and Keyboard selection.
   (b) IME on-screen configuration and features.
   (c) Keyboard key mapping editor.
   (d) Dictionary selection.
5. Potentially a keyboard testing screen.

The Input Method Editor will automatically display whenever the Android operating system requests it's display.

The application initial configuration interface will display when the application is launched through it's application icon, until the initial configuration is completed successfully or if the applications IME is no longer the default input method.

The application tutorial will automatically display whenever the application icon is invoked after initial installation.

With the exception of the IME interface, all interfaces will be available through a tabbed interface allowing the user to select other interfaces as desired.

3.11 Threads

Android is very strict about how much time a user interface process can consume during blocking operations. If Android detects an Activity that is performing long running operations within it's user interface process, then it will cause the process to force close. This is to ensure that the users interface remains responsive. The solution for an application that must perform long running operations is offloading the processing to a separate thread. The thread and the user interface process must communicate through messaging systems.

The application is designed to invoke one thread to handle this situation, this threads sole responsibility is to continually call the State Based Machines process() function, passing any required messages to the user interface thread.

3.12 Chapter Summary

This chapter discussed the following concepts:

1. The applications structure.
2. State Based Machine.
3. Utilisation of the Android native database
4. Input Method Editor (IME).
5. User interface
6. Threads and long running operations.

Chapter 4

Android Applications

This chapter discusses the following key concepts.

1. Development.
2. Android application structure.
3. Security model, application permissions.
4. Application life-cycle.
5. Android history.

4.1 Development

The software tools required for development on the Android platform are free. Google requires developers to register for a developer account[1] to publish applications to Googles Market place.

[1]Developer account registrations is currently at a cost of $25 USD,
Source: `http://developer.android.com/distribute/googleplay/publish/register.html`

The typical development path is through the installation of the Eclipse development environment from the website `http://www.eclipse.org/`. This requires the Java Development Kit (JDK), version 6 is currently recommended from the following website `http://www.oracle.com/technetwork/java/javase/downloads/index.html`.

The next component is the Android SDK Tools, they may be downloaded from the website `http://developer.android.com/sdk/index.html`.

Once both Eclipse and the Android SDK Tools are installed and up-to-date, the Android SDK Tools are incorporated into the Eclipse IDE, this is accomplished through using the *Install New Software* selection in the Help menu within the Eclipse IDE, then typing the following website into the *work with* field `https://dl-ssl.google.com/android/eclipse`.

The most accurate and up to date information on Android programming is published through the `http://developer.android.com/` website, including more detailed instructions on setting up a development environment as well as technical articles, API reference and example applications.

Many printed texts discussing Android programming exist, most follow similar patterns: introducing Android, explaining the Android architecture, User Interface Design, Discussing Intents , introduce common API's, Data storage services, threads, services and application publication. This is often accompanied with the development of example applications.

Some of the better quality texts include *Android Wireless Application Development* (Darcey & Conder 2011), *Beginning Android 4* (Murphy 2011), *Beginning Android 4* (Allen 2012),*Beginning Android Application Development* (Lee 2012) and *The Android Developers Cookbook* (Steele & To 2011).

4.1.1 Application Debugging

There are a wide selection of application debugging options available to the developer.

The Android operating system maintains an application log file known as logcat, this is where applications are able to silently write log messages to using the *Log* class, this file may be inspected from the debug target device or from within the Eclipse IDE.

The Android SDK Tools include a virtual device emulator, this provides the developer with a virtual android device where the application can be installed and tested. This virtual device includes emulation of some of the typical device features including simulated GPS, Camera (Via host system web-cam) and Audio.

The Eclipse IDE contains a *Debug* perspective where applications may be debugged using virtual or real hardware over the Android SDK Tools, Android Debug[2] facility. This enables the developer to set breakpoints, single step through application source code and inspect variables.

4.2 Application Structure

Android applications are compiled into machine byte code for interpretation by the Davlik Virtual Machine (Ratabouil 2012, p. 59). Android applications can be either developed as Java, Native C/C++ or a hybrid of the two. Independent of the applications source code origin, they result in the same compiled file for the user to install.

Applications are traditionally sourced and installed through the Android Market place[3], although alternative markets and distribution methods exist.

[2]The Android SDK Tools debug tool is known as adb.

[3]`http://play.google.com`

The compiled file frequently referred to as the *Android Application* is in compressed archive format and is a combination of the machine code[4] and resources that the application depends on, including the Application Manifest[5], graphics, icons, native libraries and data (Such as XML files). The resulting archive file is given an extension of *.APK*. Androids Davlik Virtual Machine is based on the Java Virtual Machine (Darcey & Conder 2011, p. 25).

4.3 Security and Permissions

Android is built upon a modified version of the Linux operating system, this allows the operating system to act as the Hardware Abstraction Layer (Darcey & Conder 2011, p. 23), including services for processes management, memory allocation, network resources, interprocess message services and security services.

Each android application runs in its own process with its own instance of the Davlik Virtual Machine. Each application is segregated from others through the association with a different user profile. This user profile is created by the operating system during application installation (Darcey & Conder 2011, p. 25). This effectively sandboxes each application and prevents each application from directly accessing other applications data. Access to outside resources must occur through the systems defined API's[6].

The application manifest is an XML format file that describes the application, including:

1. What services and activities the application offers.
2. The versions of the Android Operating system that the application was designed for.
3. Any permissions the application requires to operate.

[4]known as Dex format (Ratabouil 2012, p. 60)

[5]The application Manifest is described in the Section 4.3 on page 38

[6]API stands for Application Programming Interface

Table 4.1: Examples of Sensitive Information and Services

1.	GPS location data.
2.	Contact list data.
3.	File system access.
4.	Internet access.

(Lee 2012, Allen 2012, p. 30,p. 33)

Access to sensitive information and services such as the ones depicted in Table 4.1 - Examples of Sensitive Information and Services is guarded by the operating system through the use of Androids permissions model. Applications request access to required services by declaring them in the application manifest. The declared permissions the application requires to call restricted API functions is requested from the user at the time of application installation, a denial will prevent application installation. These permissions are secured at runtime by the effected operating system API functions and are validated when the sensitive information or service is accessed.

During the publication process the .APK file is signed with a self signed certificate that is able to identify the applications developer(Darcey & Conder 2011, p. 601). The private key is held[7] by the applications developer (Darcey & Conder 2011, p. 26).

4.4 Application Life-cycle

When the application is invoked as an activity, the application receives notification that various events are occurring. This occurs as the applications process state is managed by the process manager. The large range of state methods exist to improve application responsiveness (Darcey & Conder 2011, p. 75).

[7]Application upgrades must be signed with the same private key otherwise installation will fail.

4.4.1 onCreate()

The applications first such event as depicted in Figure 4.1 as the onCreate() method, this event is invoked when the application is first started (or has been restarted after being killed for some reason[8]), this is an opportunity for the application to initialise data structures, configure forms and initiate other activities such as creating threads to handle long running operations, if the application is restarting after a previous execution the application is passed in a *Bundle* containing values stored by the previous applications process instance[9](Allen 2012, p. 194).

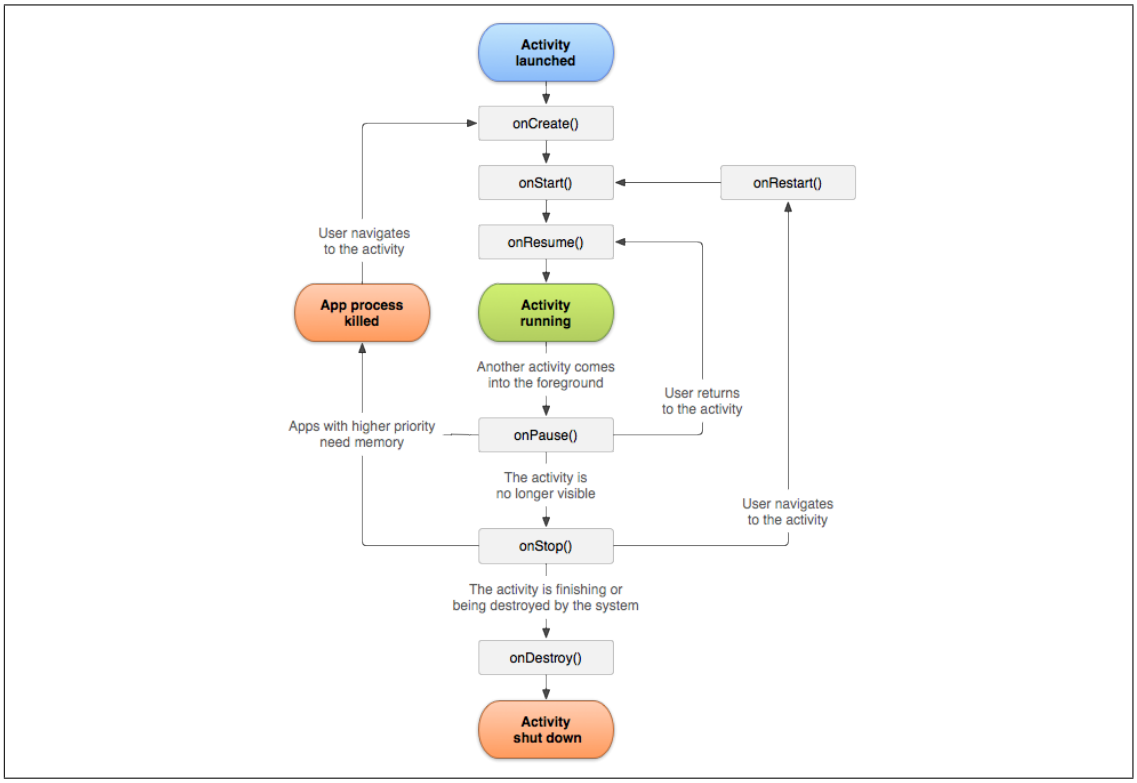


Figure 4.1: Activity life-cycle.[10]

[8]Applications may be destroyed as the result of a device rotation, they are immediately restarted.

[9]The Bundle's values are saved when the Android operating system calls the onSaveInstanceState() method.

[10]Source: `http://developer.android.com/reference/android/app/Activity.html`

4.4.2 onStart()

The onStart() method is called when the application becomes visible either as the result of being launched or returning to the foreground.

4.4.3 onResume()

onResume() is the final method to be called prior to the application returning to its running state. It is an opportunity to restart suspended threads from the onPause() method.

4.4.4 onPause()

This method is called when another application is coming to the foreground causing the current application to become a background process. The application should release access to any resources with exclusive access restrictions, such as the devices camera (Allen 2012, Lee 2012, p. 195,p. 27). Because Android reserves the right to kill off the process without further notice, this may be the only chance the application gets to save data, therefore it is highly advised to save any uncommitted data (Darcey & Conder 2011, p. 74).

4.4.5 onStop()

This function is called once the application is no longer visible. Android does not guarantee that this method will be called, in low memory conditions this is very likely.

4.4.6 onDestroy()

Since Android may not call this method in low memory situations, the main focus for this method is cleanly releasing resources.

4.4.7 onRestart()

This method will be called when the activity is becoming visible again after previously losing visibility.

4.4.8 onRestoreInstanceState()

This method is called whenever the application is restated, allowing the application the opportunity to restore application state and other simple values from the Bundle.

4.4.9 onSaveInstanceState()

This method is called whenever the application is destroyed and it is likely that the activity will restart, giving the application the opportunity to Bundle state variables and other simple values. The focus is on speed therefore no complicated processing may occur. Complex data including handles to system resources such as connected sockets, must be managed in alternative ways (Allen 2012, p. 196). One such method is the method call onRetainNonConfigurationInstance() (Allen 2012, p. 203,p. 227).

4.5 System Messaging Through Intents

Within the Android Operating System asynchronous messages may be passed between Activities. When an Activity sends such a message it is called an Intent(Darcey & Conder 2011, p. 69). Such a message is used when an Application wishes to utilise an external process to perform some action, for example dialling a phone number or displaying a map. An Intent message object may contain additional application specific data. When the intent is a Bluetooth discovery message, the additional application defined information could contain the discovered Bluetooth devices MAC[11] address. An application may also register for notification of broadcast events by creating an IntentFilter object and registering it with the operating system. In this way, applications may receive notification of system events as they occur, for example battery status notification, file download completion or the camera button was pushed(Lee 2012, p. 235). There are many such notifications that can occur including Bluetooth notifications such as:

1. Device disconnection.
2. Device discovery.
3. Bluetooth radio state change.
4. Bluetooth device discovery completion.
5. Bonding status notification (Pairing).

4.6 Android Background

Android was originally developed by a company founded by Andy Rubin and named *Android, Inc.*, this company was acquired by Google in 2005. Google working with other members of the Open Handset Alliance (OHA) commenced development of Android. The

[11]The MAC address of a device is a unique hexadecimal number that enables messages to be uniquely transmitted between devices.

resulting Android Operating System was based on the acquired technology from Android, Inc. Google made the project open source, the first version of the resulting development was released in November of 2007, since then many manufactures have adopted the Android operating system for their hardware (Darcey & Conder 2011, Lee 2012, p. 16, p. 2). Table 4.2 provides an overview of android evolution and the respective market distribution (2012 figures). Development of the Android operating system is ongoing and many new versions have been released in the years since its first commercial release.

Android applications are required to specify a minimum supported operating system and this impacts on the applications access to API functions as they have evolved throughout Androids history, many API functions have been added, while other functions have been depreciated or removed.

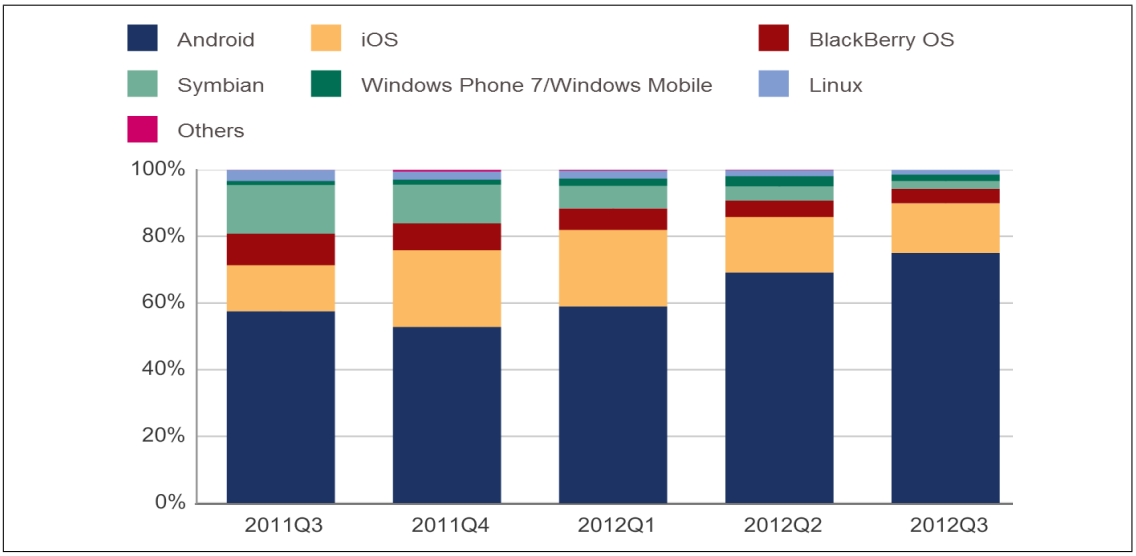


Figure 4.2: Worldwide Smartphone OS Market Share, 3Q 2012[13].

The Figure 4.2 published by the independent research organisation IDC shows just how significant the Android market is, Android shipped 104.8 Million units in quarter 2 of 2012(International Data Corporation (IDC) 2012).

[12]Sources: (Murphy 2011, p. 48), (Lee 2011, p. 20) and (Android Developers 2012)
[13]Source: (International Data Corporation (IDC) 2012)

Table 4.2: Android Versions[12]

Release date	**API Level**	**Version**	**Code name**	**Distribution (Sep⇒Oct 2012)**
Sep 2008	1	1.0		
Feb 2009	2	1.1		
Apr 2009	3	1.5	Cupcake	0.1%
Sep 2009	4	1.6	Donut	0.4%
Oct 2009	5	2.0	Eclair	
Dec 2009	6	2.0.1	Eclair	
Jan 2010	7	2.1	Eclair	3.4%
May 2010	8	2.2	Froyo	12.9%
Dec 2010	9	2.3⇒2.3.2	Ginger Bread	0.3%
Feb 2011	10	2.3.3⇒2.3.7	Ginger Bread	55.5%
Feb 2011	11	3.0	Honeycomb	
May 2011	12	3.1	Honeycomb	0.4%
Jul 2011	13	3.2	Honeycomb	1.5%
Oct 2011	14	4.0⇒4.0.2	Ice Cream Sandwich	
Dec 2011	15	4.0.3	Ice Cream Sandwich	23.7%
Jul 2012	16	4.1⇒4.2	Jelly Bean	1.8%

4.7 Chapter Summary

This chapter discussed the following concepts:

1. Getting the Android development environment setup.
2. Introduced the structure of an Android application.
3. Discussion of Androids security model.
4. Discussed the Android application life cycle.
5. Introduction to system messaging with Intents.
6. Brief review of Androids history.

Chapter 5

Bluetooth

Bluetooth is a set of protocols described by a set of specifications and standards which are maintained by the Bluetooth Special Interest Group (Bluetooth SIG[1]), for simplicity we will refer to them as *protocols*. These protocols enable Bluetooth enabled devices to communicate wirelessly within a small radius (Typically up to 10 m), devices within this sphere are able to utilise peer to peer communications and is often referred to as a Personal Area Network (PAN). This communication is subject to device security and visibility settings. Typically the device user initiates this connection and the connection is usually secured through the sharing of a secret pin. Communications may also be encrypted depending on the protocol being used for communications.

This chapter discusses the following key concepts[2].

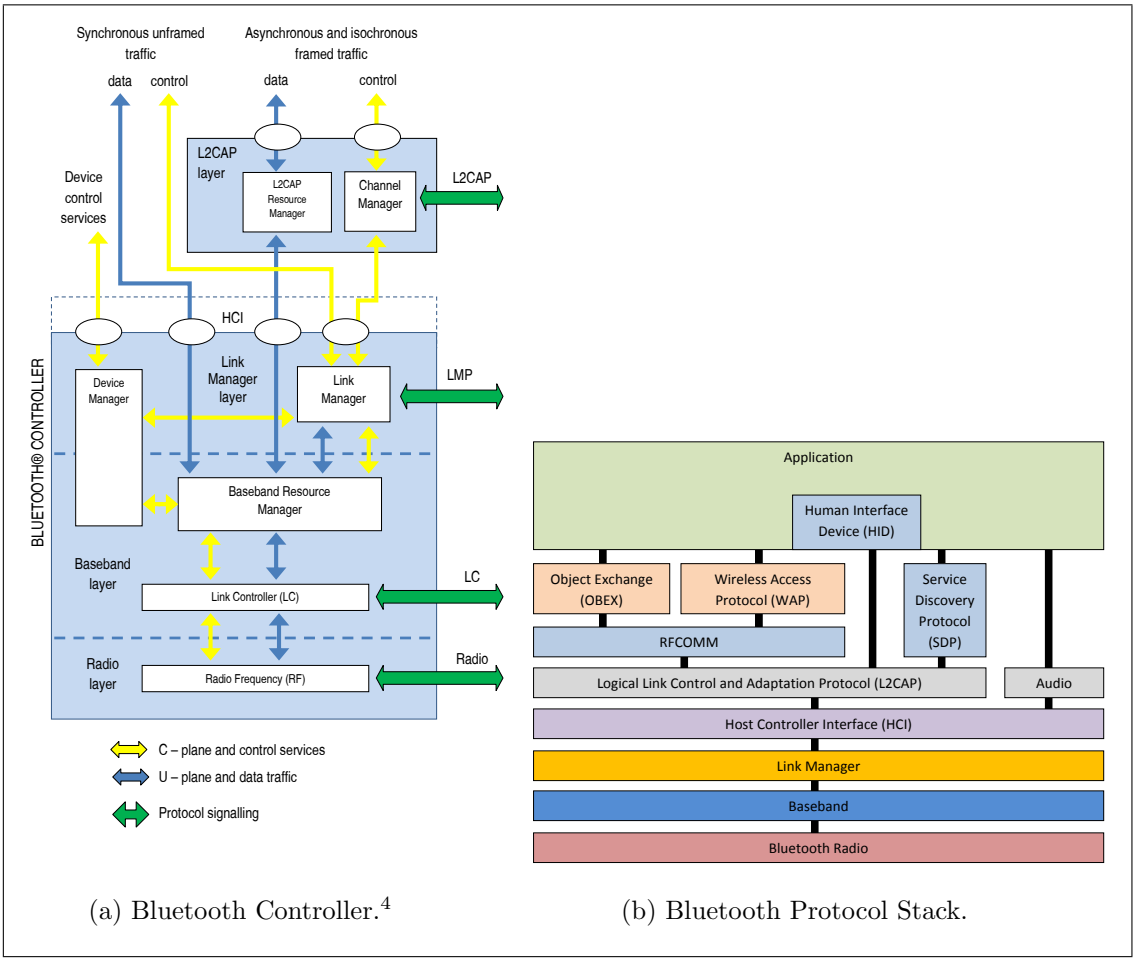
1. Bluetooth protocol stack.
2. Bluetooths use of universally unique identifiers (UUID).
3. Bluetooth socket connections.

[1] (Bluetooth SIG 2012*a*)`http://www.bluetooth.org`

[2] The topics discussed in this chapter will be restricted to areas directly relevant to this project.

5.1 Bluetooth Protocol Stack

The Bluetooth protocol stack is similar[3] to the TCP/IP protocol stack(Huang & L 2007, 11).



(a) Bluetooth Controller.[4] (b) Bluetooth Protocol Stack.

Figure 5.1: Bluetooth Protocols.

Ports and sockets can be bound and listened to for incoming connections and outgoing connections can also be established. (Hopkins & R 2003, Kumar, Kline & T 2004)

[3]Bluetooth is similar to TCP/IP from the perspective of the application performing communications through Bluetooth connections using sockets.

[4]Source: `https://www.bluetooth.org/Building/HowTechnologyWorks/Architecture/Overview.htm`

As depicted in Figure 5.1b each layer builds on the next until the applications desired level of abstraction is achieved.

5.1.1 Logical Link Control and Adaptation Protocol (L2CAP)

This layer performs significant work for the upper layers, it is responsible for disassembly and reassembly of large data packets being transmitted and received respectively as well as managing the shared access to the radio by the multiple protocols. This layer is also capable of providing error detection and managing retransmission as well as ensuring QoS levels of higher protocols are considered. Within Android this layer is not normally exposed to the Application layer through the API, access is gained through advanced techniques called reflection.

5.1.2 Host Controller Interface (HCI)

The Host Controller Interface (HCI) in Figure 5.1b is the interface between the physical Bluetooth hardware and the software portion of the protocol stack. (Kumar et al. 2004, 10) and (Bluetooth SIG 2012*b*).

The hardware abstracted by the HCI consists of:

1. The *Bluetooth Radio* is a radio frequency transceiver and is responsible for the physical transmitting and receiving of the data traffic over the 2.4 GHz frequency allocation.
2. According to *Bluetooth application programming with the Java APIs* (Kumar et al. 2004, p. 10) the *Baseband* layer is responsible for "channel processing and timing", The online resource titled *Core Architecture Blocks* (Bluetooth SIG 2012*b*) discusses responsibility for "all access to the radio medium".
3. The *Link Manager* is responsible for managing the link between devices, (Bluetooth

SIG 2012*b*) states that the Link Manager controls logical links using the link management protocol.

These hardware layers similar to Figure 5.1a are typically encapsulated in the same hardware module and made available to the host device through the HCI protocols.

5.1.3 Human Interface Device (HID)

Traditionally keyboards are physically connected to their host, most modern keyboards of this nature connect through the USB interface. The USB-HID specification[5] details the protocols and data formats that USB hosts and keyboards must implement, ensuring interoperability.

The Bluetooth - HID protocols, *when present* provides methods to communicate with input devices such as keyboards. The Bluetooth HID protocol is an encapsulation of the USB HID specification. For example, some of the types of messages that can be communicated include recovering keystrokes and modifying indicators such as caps lock. *This project will be required to implement or emulate this layer*, for this reason the HID protocol has been drawn in Figure 5.1b to be part of the Application layer, in traditional implementation it would be located between the RFCOMM and SDP protocols.

The Bluetooth HID protocol communicates using the UUID[6] profile of 4388 (0x1124 Hexadecimal). The full 128 bit UUID equals 00001124-0000-1000-8000-00805F9B34FB.

[5] `http://www.usb.org/`

[6] See: Section 5.2 - Universally Unique Identifiers (UUID) on page 51

5.1.4 Object Exchange (OBEX)

OBEX is a protocol adopted by the Bluetooth standards from the IrDA standards. It is designed to manage the transfer of file data between devices.

5.1.5 Radio Frequency Communications (RFCOMM)

RFCOMM is responsible for providing stream based connections and is conceptually similar to serial port connections.

5.1.6 Service Discovery Protocol (SDP)

Service Discovery is part of the device discovery process, the Bluetooth device queries external Bluetooth devices to build a database of services that the device offers, including the nature of these services.

5.2 Universally Unique Identifiers (UUID)

Bluetooth uses the UUID to classify services that a device offers, some are defined by the Bluetooth standards, others are application defined. Applications use the UUID to identify and establish connections to services they are interested in. (Kumar et al. 2004, Huang & L 2007, Hopkins & R 2003, p. 143, p. 17-20, p. 58-65) A standard UUID is represented by a 128 bit hexadecimal string formatted as follows:
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, where each x represents a hexadecimal value.
Profiles defined by the Bluetooth standard utilise the base UUID:
xxxxxxxx-0000-1000-8000-00805F9B34FB, where the xxxxxxxx portion is replaced with the profile or service identifier.

5.3 Bluetooth Socket Connections

The Bluetooth socket is an endpoint within the communications link(Huang & L 2007, p. 21). Applications utilise a socket either as a client or as a server, the application specifies the service profile for the connection, this service profile is similar in concept to ports in TCP/IP.

A client application initiates communication with the server by attempting connection specifying the remote device MAC address and service profile (UUID), when there is a server socket listening for connections a connection will (usually) be established. Bluetooth application sockets are usually RFCOMM or L2CAP sockets(Huang & L 2007, p. 22-23).

An application creates a server socket when it must wait for connections from other Bluetooth devices, the service is *bound* to the specified service profile socket and *listen*s for connections from clients, the *bound* service then waits for a connection, once a connection is started the server must either *accept* or reject the connection(Huang & L 2007, p. 22-23).

Once a connection is established a bidirectional communication between endpoints can commence.

5.4 Chapter Summary

This chapter discussed the following concepts:

1. The general structure of the typical Bluetooth protocol stack.
2. Bluetooths usage of universally unique identifiers.
3. How the application establishes a connection to another device through socket connections.

Chapter 6

Application Testing

This chapter discusses the following key concepts.

1. A summary of the types of testing that will be developed.
2. An introduction to functional testing.
3. Provide an account of the current tests cases for this project.

This chapter draws on multiple books discussing software testing including: Android Wireless Application Development and Software Engineering and Testing (Darcey & Conder 2011, Agarwal, Tayal & Gupta 2008, Burnstein 2003, Myers, Badgett, Thomas & Sandler 2004, Lewis 2009).

6.1 Software Testing

Before publication of the completed application can occur, extensive testing must take place, this is to ensure a high quality and fit for market application is produced. This project's Software Testing activities are designed to: ensure compliance with design

goals, to protect the end user from faulty software and protect the reputation of the developer. These quality objectives will be accomplished through various type of testing including:

1. Alpha testing.
2. Beta testing.
3. Functional testing.
4. Integration testing.
5. Regression testing.
6. Unit testing.
7. User acceptance testing.
8. Structural testing.
9. System testing.

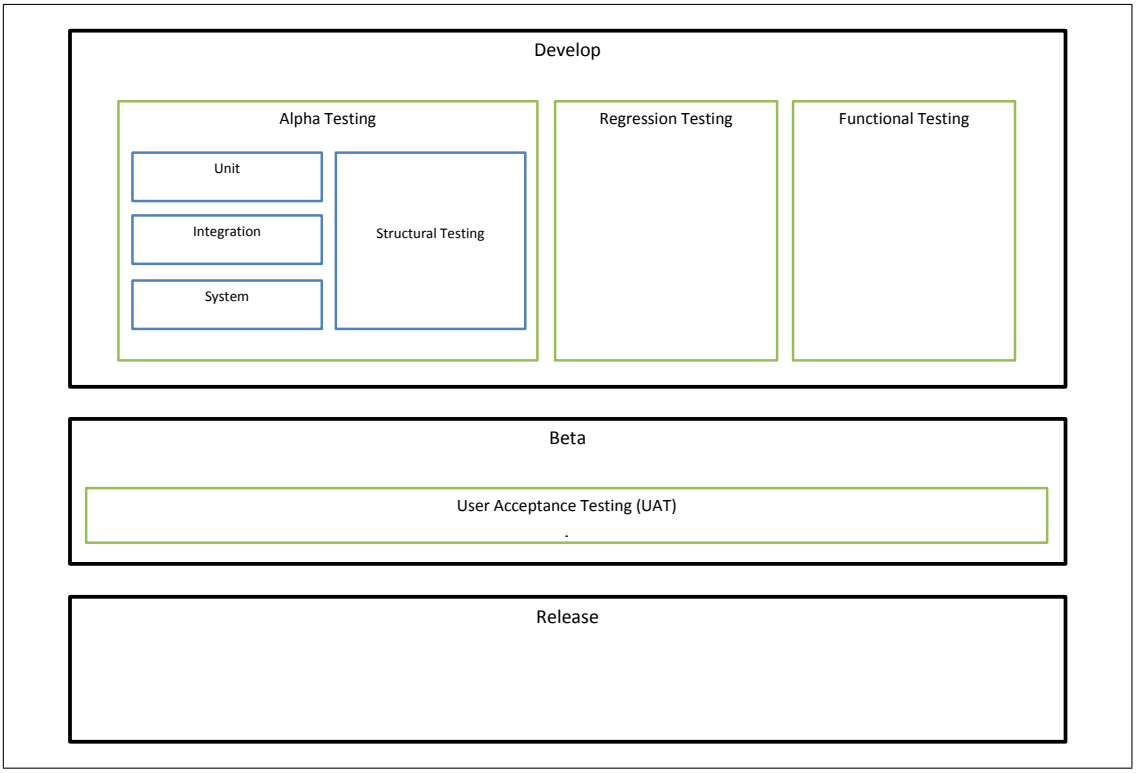


Figure 6.1: Testing type relationship diagram.

The relationship between the testing areas utilised by this project are depicted in Figure 6.1 - Testing type relationship diagram. A brief introduction to these testing areas can be located in the following sections, with a specific focus on *Functional Testing*.

For testing to be most effective the tests should be documented and tracked in a register. The tests for this project are contained in a comma separated values file that track the current status of all tests identified.

Test cases for this project are documented in Appendix D beginning on page 145.

6.2 Alpha Testing

Alpha testing is any testing that takes place by the developers or the developers internal testing team before releasing the version for outside evaluation. Alpha testing can include most forms of testing including Unit, Integration, System and Structural testing. Once application maturity is reached and Alpha, Regression and Functional testing has successfully completed, testing will progress to beta testing.

6.3 Beta Testing

Beta testing is performed by a Larger controlled group, usually by invitation or key stake holders within the client organisation. Testing usually occurs outside of the developers domain. It is important that this group of testers are informed of expectations as well as the responsibilities they are accepting. It is critical that the test users do not attempt to use the system in place of current production systems, this must therefore be clearly communicated. Beta testing will often reveal defects that the developers or the alpha test team did not locate previously. Beta testing can use a mixed array of reporting to collect defect information, including bug reports, automated data collection (such as log

files), silent observation and test user surveys. All defects uncovered should be recorded in a central register for classification, investigation and review. Once the beta testing has concluded with a satisfactory outcome including the user acceptance testing, the product life-cycle will enter the release phase.

6.4 Functional Testing

The book *Software engineering and testing* (Agarwal et al. 2008, p. 192) provides a good grounding in the concepts of functional testing and the following definition of functional testing is based largely upon the definitions defined within this book. Functional testing is sometimes also referred to as black-box testing, test cases are developed by drawing upon the system (or application) requirements and specifications. Functional testing does not consider the systems internal structure. The test cases are designed to guide the tester to manipulate input to affect the output thereby exercising the functional specifications of the system. Testing occurs from the perspective of the end user. Developer bias is avoided when an independent third party performs testing, the tester does not require development skills. Functional testing can either have a positive or negative focus. Positive testing attempts to prove either the presence or absence of bugs effecting the correct operation. Negative testing seeks to confirm the absence of specific types of bugs relating to unexpected inputs (e.g. Out of bounds values.)

6.5 Integration Testing

Integration testing is the process of testing related units in combination with a focus on identification of undiscovered bugs, testing of the application will occur in multi faceted

approach, including top-down[1] , bottom-up[2] and incrementally[3] dependant on context and path of least resistance.

6.6 Regression Testing

Regression testing is concerned with ensuring resolved issues and bugs that were previously identified and resolved have not reoccurred unexpectedly in subsequent versions of the upgrade or development iteration.

6.7 Unit Testing

Testing isolated application code functions and classes with test data to compare expected output to actual output. Some opportunity exists for automation in this area.

6.8 User Acceptance Testing

User acceptance testing (UAT) is part of the final delivery process. Once User acceptance testing has completed successfully, the following cycle will commence:

1. Review end user feedback.
2. Respond to user requests.
3. Bug fix.

[1]top-down testing refers to when an application is tested from the top of the main application down towards the finer grained functions.

[2]bottom-up testing refers to when units are tested from the lowest levels and progress upwards towards the main application.

[3]Incremental testing is where units are tested in isolation from the rest of the application and as the development progresses more units are incorporated into the integrated tests.

4. Develop new features and upgrades.
5. Testing including Regression tests.

6.9 Structural Testing

Structural testing is also called white-box testing, it is performed with full knowledge of the source code. The purpose is to develop tests that ensure all logical paths through the module being tested have been executed. In other words, all true and false paths are explored and loop boundaries are tested. This results in additional test cases. The resulting test cases may be automated. Testing focus is usually on small sections of the code at a time. This form of testing grows in scale as the programs complexity increases. (Agarwal et al. 2008, p. 173-175)

6.10 System Testing

System testing is focused on finding errors in interactions between system components and the subsystems. According to Software Engineering and Testing (Agarwal et al. 2008, p. 172), this level of testing is performed during the phases Alpha, Beta and Acceptance testing. This projects System testing is predominantly Beta testing focused.

6.11 Testing Status

The table 6.1 provides an overview of the current test cases, as at publication (30th December 2012). Tests that have not yet occurred do not indicate a completed date or outcome (Blank). Each test case is documented fully in Appendix D - Test Cases on page 145.

Table 6.1: Test Cases

Test ID	Test Name	Tester	Completed	Outcome
TUN1	State Transition	Fred		
TRE2	Spinner	Fred	7/12/2012	Pass
TFU3	Scanning	Fred	7/12/2012	Pass
TFU4	Connecting	Fred	7/12/2012	Pass
TFU5	Disconnecting	Fred	7/12/2012	Fail
TFU5	Disconnecting	Fred	14/12/2012	Pass
TFU6	Enable Radio	Fred	6/12/2012	Pass
TFU7	Radio Restore Off	Fred	7/12/2012	Pass
TFU8	Radio Restore On	Fred	7/12/2012	Pass
TUN9	Key Polling	Fred	7/12/2012	Fail
TFU10	Tab Switching	Fred	7/12/2012	Pass
TSY11	Data Persistence SQL	Fred	7/12/2012	Pass
TSY12	Data Persistence Application State	Fred	7/12/2012	Fail
TUN13	Database Upgrade	Fred	7/12/2012	Pass
TUN14	Database Creation	Fred	7/12/2012	Pass
TIN15	CRUD	Fred		
TRE16	Thread Creation	Fred	7/12/2012	Pass
TSY17	Message Passing	Fred	7/12/2012	Pass
TIN18	Intent Reception	Fred		
TFU19	App State Switch Exit	Fred		
TFU20	App State Switch Home	Fred		
TFU21	Interface response to device rotation	Fred		
TST22	Low task memory	Fred		
TST23	Low storage space	Fred		
TFU24	Interface portrait small form factor	Fred		

continued . . .

. . . continued

Test ID	Test Name	Tester	Completed	Outcome
TFU25	Interface landscape small form factor	Fred		
TFU26	Interface portrait large form factor	Fred		
TFU27	Interface landscape large form factor	Fred		
TFU28	Input Method Editor	Fred	9/12/2012	Fail
TUN29	Correct operation of NullBooleanField-Helper	Fred		
TFU30	Device Compatibility Review	Fred	10/12/2012	Pass
TFU31	Android OS Compatibility Survey	Fred		
TFU32	Bluetooth keystroke capture	Fred	9/12/2012	Fail
TFU33	Keystroke utilised by user process.	Fred	9/12/2012	Fail
TFU34	Test application executes on phone device.	Fred	9/12/2012	Pass
TFU35	Test application executes on tablet device.	Fred	9/12/2012	Pass
TBE36	Application Upgrade	Fred		
TSY37	Heavy CPU Load	Fred		
TST38	Restart ability after forced termination.	Fred		
TSY39	Connected Bluetooth Device turn off	Fred	9/12/2012	Fail
TSY40	Low battery behaviour	Fred	9/12/2012	Fail
TSY41	Standby behaviour	Fred		
TSY42	Device configuration change	Fred		
TSY43	Other app turning Bluetooth off	Fred		
TSY44	Bluetooth device pairing.	Fred	9/12/2012	Fail
TFU45	Responsiveness	Fred		
TFU46	Usability verification	Fred		
TST47	Data Security	Fred		

6.12 Chapter Summary

This chapter discussed the following concepts:

1. The testing strategies that this project employs.
2. Definitions of testing concepts, including Functional Testing.
3. Lists detailing the status of current test cases currently in use.

Chapter 7

Conclusions and Further Work

A lot of issues were experienced during this project, one of the more serious ones related to the error from the Android phone (LG P690f) as depicted in Figure 7.1, which implies that the L2CAP[1] layer has not been implemented, or completed in a way to make it suitability available, despite the use of reflection[2] to gain access to hidden operating system function calls[3].

```
...E/BLZ20_WRAPPER(3486): ##### ERROR : blz20_wrp_connect: protocol BTPROTO_L2CAP not yet supported#####
```

Figure 7.1: Bluetooth error message.

Other issues included:

1. Difficulty with user interface elements such as the Spinner control click-ability failing due to a child element containing a *clickable=true* property, thereby breaking

[1] L2CAP is defined in Chapter 5 - Bluetooth in Section 5.1.1 on page 49

[2] Reflection is an advanced technique to discover and access details about Java objects, including methods, fields and constructors. It provides the developer the ability to invoke these methods within security constraints.

[3] the author suspects that the RFCOMM layer directly incorporates the L2CAP layer.

the framework.

2. Multiple *force close* issues relating to programming errors and long running operations.
3. Android virtual device emulation not supporting Bluetooth protocol stack.

If the project was assessed right now, these multiple issues would cause a conclusion of a project failure. Despite this, the author still feels that this project is achievable with enough time and resources.

This project did teach a lot of lessons relating to Android development, including:

1. Threads for off loading long running activities.
2. Database interaction for storing application data in SQLite databases.
3. Upgrade procedures how Android handles the upgrade process between program versions, for example database structure alterations.
4. User Interface interactions and design, connecting underlying elements to the data source through database Adapters.
5. Debugging of applications through emulators as well as through the use of real hardware.
6. Intent filters enable the application to capture events that occur which it has an interest in, e.g. device discovery.
7. Inter process communications through message passing.
8. Technical understanding of the Bluetooth protocol stack.

The other learning outcome is that projects that at first appear simple often are not, and that there are a many unforeseen difficulties that are likely to be encountered and overcome, requiring a much greater time commitment than first anticipated.

If this project depended on connecting to some other device through RFCOMM many issues would have been avoided due to the relative maturity of this protocol in the Android Bluetooth stack.

7.1 Work Completed and Achievement of Project Objectives

The following details the status of the following objectives from the Project Specification:

1. The research component is documented within the following chapters:
   (a) Chapter 3 - Application Design details the design of the application.
   (b) Chapter 4 - Android Applications details the basics of what is required to setup a development environment for Android and details the basics of implementing Android applications.
   (c) Chapter 5 - Bluetooth details the Bluetooth protocols including the USB-HID protocol encapsulated by the Bluetooth HID protocols.
2. Evaluation of alternative solutions and design document.
   (a) Chapter 2 - Literature Review and Bluetooth Keyboard Software Review details the *alternative solutions*.
   (b) With the sole exception of this dissertation, the *design document* has not been completed due to unresolved issues relating to the Bluetooth stack.
3. *Technical feasibility* has not yet been realised due to previously mentioned reasons. However significant code for a complex test application is listed in Appendix C - Application Source Code beginning on page 89, the evolution of this code is detailed in the subversion log file reproduced in Appendix B - Subversion Change Log beginning on page 75.
   (a) Completion of the *Software solution* was not possible due to previously mentioned issues.
4. The test plan is documented in Chapter 6 Application Testing and Appendix D - Test Cases.
5. Software Testing (including Functional testing) has commenced.
6. *Time Permitting:* Hardware testing is not yet possible.
7. *Time Permitting:* Evaluate the design and document the results, other than this

dissertation there are no significant evaluations and related documentation.

7.2 Further Work

To complete this project:

1. Multiple Bluetooth layers will need to be replaced due to immaturity of the native support, this will include the L2CAP and the HID layers.
2. The detailed design document will need development after proof of concept is realised through the reception of keystrokes.
3. The Input Method Editor requires development.
4. The in application tutorial requires development.
5. Significant testing will be required to ensure portability and stability.
6. Auto-completion, dictionary integration and predictive learning are areas that would add value to the finished product.

Item 1 above is a significant piece of work, it is highly recommended that the implementation utilise the native development kit NDK[4]. (Ratabouil 2012)

[4]it is the authors opinion that this undertaking would be worthy of a dissertation on it's own.

References

Agarwal, B., Tayal, S. & Gupta, M. (2008), *Software Engineering and Testing*, Jones and Bartlett Publishers, Massachusetts, United States of America.

Allen, G. (2012), *Beginning Android 4*, www.apress.com, New York, United States of America.

Android Developers (2012), 'Platform versions', `http://developer.android.com/resources/dashboard/platform-versions.html`. [Online; accessed April 2012].

android.com (2012*a*), 'Bluetoothdevice', `http://developer.android.com/reference/android/bluetooth/BluetoothDevice.html`. [Online; accessed October-2012].

android.com (2012*b*), 'Bluetoothsocket', `http://developer.android.com/reference/android/bluetooth/BluetoothSocket.html`. [Online; accessed October-2012].

android.com (2012*c*), 'Creating an input method', `http://developer.android.com/resources/articles/creating-input-method.html`. [Online; accessed October-2012].

Bluetooth SIG (2012*a*), 'Bluetooth sig website', `https://www.bluetooth.org/`. [Online; accessed August-2012].

Bluetooth SIG (2012*b*), 'Core architecture blocks', `https://www.bluetooth.org/Building/HowTechnologyWorks/Architecture/Overview.htm`. [Online; accessed October-2012].

Bluetooth SIG (2012*c*), 'Human interface device profile (hid)', `https://www.bluetooth.org/Building/HowTechnologyWorks/ProfilesAndProtocols/HID.htm`. [Online; accessed August-2012].

Burnstein, I. (2003), *practical Software Testing: a process oriented approach*, Springer-Verlag New Yourk, Inc., Illinois, United States of America.

Casad, J. (2011), 'Smart developer linux special', (January), 98.

Darcey, L. & Conder, S. (2011), *Android wireless application development*, Addison-Wesley, Boston, United States of America.

Froehlich, C. (2011), *Android App Development*, Alpha, A member of Penquin Group (USA) Inc., New York, United States of America.

Google Inc. (2012), 'The official site for android developers.', `http://developer.android.com/`. [Online; accessed August-2012].

Hopkins, B. & R, A. (2003), *Bluetooth for Java*, apress, Berkeley, United States of America.

Huang, A. & L, R. (2007), *Bluetooth essentials for programmers*, Cambridge University Press, New York, United States of America.

International Data Corporation (IDC) (2012), 'Android and ios surge to new smartphone os record in second quarter, according to idc', `http://www.idc.com/getdoc.jsp?containerId=prUS23638712`. [Online; accessed December 2012].

Kumar, C., Kline, P. & T, T. (2004), *Bluetooth application programming with the Java APIs*, Morgan Kaufmann Publishers, San Francisco, United States of America.

Lee, W.-M. (2011), *Beginning Android Application Development*, Wiley Publishing, Inc., Indiana, United States of America.

Lee, W.-M. (2012), *Beginning Android 4 Application Development*, John Wiley & Sons, Inc., Indiana, United States of America.

Lewis, W. E. (2009), *Software testing and continuous quality improvement*, Auerbach Publications, Taylor & Francis Group, Florida, United States of America.

Murphy, M. (2011), *Beginning Android 3*, www.apress.com, New York, United States of America.

Myers, G. J., Badgett, T., Thomas, T. M. & Sandler, C. (2004), *The ART of software testing, 2nd EDN*, John Wiley & Sons, Inc., New Jersey, United States of America.

Oracle and/or its affiliates (2012), 'Java ™ reflection api', `http://docs.oracle.com/javase/7/docs/technotes/guides/reflection/index.html`. [Online; accessed December 2012].

Ratabouil, S. (2012), *Android NDK*, PACKT Publishing, Birmingham, United Kingdom.

Steele, J. & To, N. (2011), *The Android developer's cookbook.*, Addison-Wesley, Boston, United States of America.

Universal Serial Bus (USB) (2012), 'Device class definition for human interface devices (hid)', `http://www.usb.org/developers/devclass_docs/HID1_11.pdf`. [Online; accessed 18 April 2012].

University of Southern Queensland (2012), *ENG4111 Research Project Part 1 - Project reference book - Semester 1 2012*, Toowoomba.

USB Implementers Forum, Inc. (2012*a*), 'Hid information', `http://www.usb.org/developers/hidpage/`. [Online; accessed August-2012].

USB Implementers Forum, Inc (2012*b*), 'Homepage for www.usb.org web site', `http://www.usb.org/`. [Online; accessed 18 April 2012].

Appendix A

Project Specification

The project specification is located on the following page.

ENG4111/ENG4112 – Research Project
Project Specification - S1&2 2012

For:	Fred Houweling fred@houweling.com.au
Topic:	Design and Development of Human Interface Device (HID) Bluetooth keyboard application for Android devices.
Supervisors:	Dr Alexander Kist
Enrolment:	ENG4111 – S1, External, 2012; ENG4112 – S2, External, 2012
Project Aim:	This project seeks to investigate and subsequently develop a marketable software solution for the issues involved in providing hardware keyboard support (through Bluetooth) to applications running that run on the Android devices.
Sponsorship:	Fred Houweling
Programme:	Issue C, 21st March 2012

1. Research background information and relevant standards covering all aspects of this project including: Android development, Bluetooth and HID protocols, input method's, user interface issues, orientation issues (Portrait/Landscape), screen resolution issues, Android market place, quality testing, and other relevant topics.
2. Evaluate alternative implementations and develop design document.
3. Analyse technical feasibility using a simple test program (Bluetooth keystroke capture and display).
   3.1. Build a software solution to provide keyboard support to Android applications using the Bluetooth keyboard.
4. Develop a comprehensive test plan.
5. Undertake initial functional testing.

As time permits:-

6. Test application on various hardware platforms including phones, tablets and different keyboards.
7. Evaluate the design and document results.

AGREED _____ (Student) _____ (Supervisor)

Date: 21 / March / 2012 Date / / 2012

Examiner/Co-Examiner: _____

Appendix B

Subversion Change Log

This is the subversion revision log for this project[1].

Please refer to Appendix C Application Source Code located on page 89 for details on gaining access to the project subversion (SVN) repository, which includes a more detailed account of the projects history.

Table B.1: Subversion Log

Rev	**Who**	**Date**	**Description**
296	fred	24-02-2012	Final Year Research Project
297	fred	24-02-2012	Folder for Part 1 items.
298	fred	24-02-2012	Folder to hold the Blue tooth Project source code.
299	fred	24-02-2012	Initial Import
300	fred	24-02-2012	A folder to hold all Java Projects
301	fred	24-02-2012	Reorganise
302	fred	24-02-2012	Test Project
303	fred	24-02-2012	Importing the Test Project.
304	fred	24-02-2012	Clean up
305	fred	24-02-2012	Clean up
312	fred	07-03-2012	Check in WIP
314	fred	15-03-2012	Failed Projects
315	fred	16-03-2012	Added proof of concept folder.
316	fred	16-03-2012	Initial Project
317	fred	16-03-2012	Fresh Start
318	fred	17-03-2012	Initial Base Version
319	fred	18-03-2012	Working on user interface elements. Does not run, Null pointer exception
320	fred	20-03-2012	Working on the tabbed code.

continued ...

[1]**Note:** *Due to its historic nature and the inability to alter the comments after the fact, the text is provided as is and may include issues with grammar and spelling.*

. . . continued

Rev	Who	Date	Description
321	fred	22-03-2012	added images, and fixed a force close relating to a value returned by a private member function.
322	fred	22-03-2012	Fixed the error preventing the text being displayed. Also replaced the stock icons with my own versions.
323	fred	25-03-2012	Check in work in progress.
324	fred	25-03-2012	Adding bluetooth code to the Test Activity
325	fred	26-03-2012	Adding code to display the bonded device name and class information.
326	fred	18-04-2012	Adding a wait for bluetooth to turn on, need to setup threads.
327	fred	21-04-2012	Adding a class to maintain the bluetooth state.
328	fred	22-04-2012	Considering using LaTeX for dissertation.
329	fred	25-04-2012	Working on bluetooth connection thread and associated state based machine. Also added some static source images for the application icon. And included my brief experiment with using LaTeX (Abandoned due to learning curve, will document with MS Word).
330	fred	25-04-2012	Work in progress. Started documenting State based transitions.
331	fred	27-04-2012	Fixed connection_tab layout for the radio group horizontal layout.
333	fred	02-05-2012	Progressing the project appreciation. Started thinking about the IME Service class.
334	fred	08-05-2012	Checking the current version of the Project Appreciation. Also working on Activity/Service communication.
335	fred	08-05-2012	Working on service binding to allow communications between service and interface.
336	fred	09-05-2012	Progressing development. Project Appreciation: Added a few missing references. (a few more are yet to be added.)
337	fred	09-05-2012	Redesigning the Thread code, trying to get it to post messages back.
338	fred	14-05-2012	Fixing FC issues, almost have the thread right.
339	fred	15-05-2012	Working on Project Appreciation, As well as Thread/UI communications.

continued . . .

. . . continued

Rev	Who	Date	Description
345	fred	17-05-2012	Fixed a force close, and enabled the SBM to transition to RADIO_ON
350	fred	18-05-2012	Adding project documentation. Renamed Project_Appreciation.docx to Disertation.docx and Created a new Project_Appreciation.docx in the parent directory (Doc).
351	fred	18-05-2012	Checking in Project Appreciation progress.
352	fred	21-05-2012	Checking in work in progress.
353	fred	22-05-2012	Added a project plan, expanding the project appreciation.
354	fred	23-05-2012	Finalising Project Appreciation Modified application to add entries to the device selection spinner. Renamed the tabs to reflect the intended function. Also added a few new images.
355	fred	23-05-2012	Finished Project Appreciation. Parking for the moment pending progress in other subject.
356	fred	07-06-2012	Adding a state transition table.
357	fred	08-06-2012	Adding transition class to manage state transitions.
358	fred	12-06-2012	working on State Transition Code
359	fred	12-06-2012	Adding Bluetooth code.
360	fred	18-06-2012	Working of a FC issue that occurs on real devices relating to Looper.prepare() Added a few newly identified states.
361	fred	18-06-2012	Hooking up Activity start, pause, resume and destroy calls. Added code to return the BT Radio to it's original state when Activity is destroyed.
362	fred	18-06-2012	Fixed a force close issue, also added a debug log message to monitor state transitions.
363	fred	25-06-2012	Working on device searching related states.
364	fred	26-06-2012	More work on state transitions.
365	fred	29-06-2012	Adding Scan button, and visibility code.
366	fred	30-06-2012	Adding methods to start discovery and to receive advice of discovered devices.

continued . . .

. . . continued

Rev	Who	Date	Description
374	fred	22-07-2012	Adding intent filters to capture device discovery events. Also working on revised Project Appreciation.
375	fred	22-07-2012	Bug fix, forgot to call unregisterReceiver in onStop.
376	fred	25-07-2012	Work in progress
377	fred	26-07-2012	pre submission
378	fred	31-07-2012	Adding extra classes to handle persistent storage, also uploading th efinal version of the project preliminary report.
379	fred	31-07-2012	Extending the database helper classes.
380	fred	31-07-2012	Added create table information.
381	fred	31-07-2012	Working on a Keyboard item view.
382	fred	31-07-2012	Continuing hooking up the database.
383	fred	31-07-2012	Finished hooking up the TestActivity form to the Database, yet to test, after adding methods to populate database with discovery and pairing information.
384	fred	31-07-2012	Work in progress, Database query currently causes a FC.
385	fred	01-08-2012	Fixed all current FC issues, Database connected to the Spinner. Need to work on populating the database.
386	fred	02-08-2012	Check-in work in progress.
387	fred	02-08-2012	Check-in WIP.
388	fred	02-08-2012	Work in progress.
389	fred	03-08-2012	wip
390	fred	03-08-2012	WIP
391	fred	04-08-2012	Finished first cut on InsertOrUpdate function. Hooked up pairing search and search discovery to database.
392	fred	07-08-2012	Fixed a few bugs that lead to FC issues.
393	fred	15-08-2012	Fixed issue with duplicate data records with same address getting inserted into the keyboard db.

continued . . .

. . . continued

Rev	Who	Date	Description
394	fred	15-08-2012	Fixed a few nasty bugs relating to trying to use null objects.
395	fred	15-08-2012	Fixed remaining bug relating to database entries (default values and the values being set in the database true/false vs. 1/0.
396	fred	20-08-2012	Adding abstract
397	fred	22-08-2012	Adding method to update the form data when data has changed.
398	fred	23-08-2012	Fixing bugs in form update refresh, still need to find out why device_found intent is not updating the spinner...
399	fred	30-08-2012	Updated icon art work to reflect name alteration and added a script to hopefully create the icon set automatically... (Sourced from: http://registry.gimp.org/node/25592 links to: http://izvornikod.com/Blog/tabid/82/EntryId/5/GIMP-script-for-creating-Android-icons-at-once.aspx)
400	fred	30-08-2012	Fixed application icons and artwork. the transparency now works correctly.
401	fred	03-09-2012	resized text on application icon to improve readability.
402	fred	04-09-2012	Adding presentation.
403	fred	04-09-2012	Added a menu to refresh the database manually.
404	fred	04-09-2012	Adding image to PPT.
405	fred	08-09-2012	Working on adding spinner item selection, also cleaned up some old comments.
406	fred	09-09-2012	Working on the look and feel of the spinner, also added a connect button.
407	fred	09-09-2012	added a prompt to the spinner
408	fred	10-09-2012	Fixed a nasty bug preventing the spinner from allowing selections from it's drop down. In short setting anything Clickable or Focusable in the layout for the spinner items will break it's click ability.
409	fred	11-09-2012	Renamed TestActivity to ConnectActivity, Added TestingActivity, Started building a form for managing data input from the keyboard.
410	fred	11-09-2012	Preparing the interface for keyboard data reception.

continued . . .

. . . continued

Rev	Who	Date	Description
411	fred	12-09-2012	Adding data base CRUD. also adding many other changes.
412	fred	12-09-2012	Finished first cut of database CRUD (Create, Read, Update, Delete).
413	fred	12-09-2012	Work in progress.
414	fred	12-09-2012	WIP
415	fred	12-09-2012	Starting work on the code to establish a bluetooth connection.
416	fred	13-09-2012	Working on presentation and getting the application to connect to the HID.
417	fred	13-09-2012	Working on the connecting path... This revision will force close...
418	fred	14-09-2012	Work in progress.
419	fred	14-09-2012	Major cleanup of code, adding header comment.
420	fred	14-09-2012	Cleanup repo
421	fred	15-09-2012	Adding presentation pdf file Modify a minor look and feel item.
422	fred	15-09-2012	WIP
423	fred	15-09-2012	Improved stability.
424	fred	15-09-2012	Clean-up presentation.
425	fred	16-09-2012	PPT WIP
426	fred	16-09-2012	Working on content for ppt.
427	fred	16-09-2012	PPT WIP
428	fred	16-09-2012	PPT WIP
429	fred	17-09-2012	
430	fred	18-09-2012	Updating Presentation.pptx Added an image Typos feedback etc
431	fred	19-09-2012	Checking work in progress.
432	fred	19-09-2012	Added a few slides and the video.
433	fred	19-09-2012	PPT WIP
434	fred	20-09-2012	Adding my presentation sign-off sheets.
435	fred	07-10-2012	Building dissertation document.

continued . . .

. . . continued

Rev	Who	Date	Description
436	fred	07-10-2012	Gearing up for major production.
437	fred	07-10-2012	Work in progress.
438	fred	08-10-2012	Creating chapter stubs.
439	fred	08-10-2012	Check in WIP
440	fred	09-10-2012	Working on the repository.
441	fred	09-10-2012	Altered some Image names to reflect purpose. Finished the SVN REPO Log export setup for inclusion into dissertation appendix. Checked in updated project specification.
442	fred	09-10-2012	Some last minute code clean up. Adding new figures to the dissertation. Corrected a few errors in the dissertation. Adding other content to the dissertation.
443	fred	11-10-2012	Added a few files, Lots of dissertation work to check in.
444	fred	12-10-2012	check in days work.
445	fred	12-10-2012	Expanding the blue tooth section.
446	fred	13-10-2012	Quick edit of blue tooth info
447	fred	13-10-2012	Check in work in progress, Bluetooth expansion. Added some extra citations and a definition for QoS.
448	fred	14-10-2012	Attempting to switch to L2CAP based connection instead of RFCOMM. Also restored the missing source URL for a portion of code and moved effected function towards the end of the source file blueToothConnection.java
449	fred	14-10-2012	Days changes to dissertation being checked in.
450	fred	16-10-2012	Checking in last nights work. Dissertation: Added 3rd party solution table. Finished redrawing diagram from Bluetooth Controller from the BT SIG website.
451	fred	16-10-2012	Clean up repository, about to move large content.
452	fred	16-10-2012	Relocating this large file.
453	fred	16-10-2012	Removed a file that is not really required and is rather large.

continued . . .

. . . continued

Rev	Who	Date	Description
454	fred	16-10-2012	Relocating images
455	fred	16-10-2012	Relocate USQ's bengdis
456	fred	16-10-2012	Move bengdis folder.
457	fred	16-10-2012	Adding a few tables.
458	fred	16-10-2012	Checking in WIP, Pick it up again tonight.
459	fred	16-10-2012	Added State Transition Table, still needs cleaning. Also added the condensed version of the state transitions in the relevant section of the source code.
460	fred	17-10-2012	check in days work.
461	fred	21-10-2012	Finished tweaking the source code listing colours to replicate Eclipse IDE colours, also reduced source font size on A.Kist request. Removed "Chapter Overview" heading from all chapters. Fixed punctuation in the nomencl.tex file. Made a few items into proper enumerations in LaTeX in design.tex.
462	fred	21-10-2012	Looking at the connection code
463	fred	21-10-2012	Fixed a formatting issue in the transition table in design.tex. Started modifying the state engine to utilise the simplified transitions that have been identified from the state table. i.e. Made the TargetState List based so there are less entries to add and search.
464	fred	21-10-2012	Added method and constructor to help the state lookups.
465	fred	22-10-2012	Just implemented the entire state transition table as it is currently understood.
466	fred	22-10-2012	Fixed a TransitionState bug relating to populating the TargetState list. Discovered an error in the state lookup table relating to state transitions due to the new states introduced for ENABLE_RADIO and RESTORE_RADIO.
467	fred	22-10-2012	updated some of the conclusions to reflect better the current source status.

continued . . .

. . . continued

Rev	Who	Date	Description
468	fred	24-10-2012	Fixed state transition table, yet to test. Some work on the conclusion. Updated state diagram to better reflect the ability to return to RADIO_ON from the CONNECTING state.
469	fred	24-10-2012	Working on the state table design, trying to reduce margin,
470	fred	25-10-2012	Got cell rotation in MiKTeX to work, making the table less wide and preventing the use of the margin space.
471	fred	26-10-2012	Checking in Work in progress.
472	fred	26-10-2012	Fixing the state table
473	fred	27-10-2012	Checking in work in progress.
474	fred	28-10-2012	Finished the state table
475	fred	28-10-2012	Fixed a few formatting errors in the state table, as well as a typo.
476	fred	30-10-2012	Started fixing abstract
477	fred	31-10-2012	Working on the new abstract.
478	fred	06-11-2012	Adding work in abstract.
479	fred	09-11-2012	Working on Abstract
480	fred	11-11-2012	Modifying abstract to simplify it.
481	fred	12-11-2012	Adding additional keyboard assets.
482	fred	15-11-2012	Fixed up the look of the keyboard image. Created a new image showing a simple overview of the applications purpose. Added registered Bluetooth Logo files.
483	fred	20-11-2012	Work on abstract, nearing completion.
484	fred	20-11-2012	Modify the abstract layout, also reinstated a incorrectly deleted file.
485	fred	21-11-2012	Almost have the Abstract on 1 page.
486	fred	21-11-2012	Abstract is now 1 page.
487	fred	22-11-2012	Fixed an error in the State Transition table where Current and Transition states were transposed, due to a error in the excel file that creates that block of code.

continued . . .

. . . continued

Rev	Who	Date	Description
488	fred	24-11-2012	Started work on Chapter 1 and 2
489	fred	24-11-2012	Forgot to save one file before last commit.
490	fred	28-11-2012	Added a test spreadsheet.
491	fred	28-11-2012	started building a table structure for the test cases.
492	fred	29-11-2012	Update to the test case register
493	fred	29-11-2012	Added many new test cases to the register.
495	fred	30-11-2012	Adding test case newcommand WIP.
496	fred	30-11-2012	Writing the description of functional testing.
497	fred	02-12-2012	Got reading Test Cases from a CSV file working.
498	fred	02-12-2012	Fixed Test Case table look.
499	fred	02-12-2012	Started adding individual test case tables.
500	fred	03-12-2012	Finished the table structure for individual test cases.
501	fred	03-12-2012	Test Case table layout improvements.
502	fred	03-12-2012	Reduced table width. Still need to work out why the first record is marked 9.2 instead of 9.1
503	fred	03-12-2012	Adding more detail to the test cases.
504	fred	04-12-2012	Added 4 new references, working on the testing section.
505	fred	04-12-2012	Clean up the test case section.
506	fred	04-12-2012	Working on the testing chapter.
507	fred	05-12-2012	Added sections to expand on the testing conversation.
508	fred	05-12-2012	Work in progress.
509	fred	05-12-2012	Working on the testing section more.
510	fred	05-12-2012	Finish work for the night. Added a diagram of how I see the testing fitting together.
511	fred	06-12-2012	Checking in current work on the testing chapter.

continued . . .

. . . continued

Rev	Who	Date	Description
512	fred	06-12-2012	Changed source code listing to grey scale, too costly to print. Shame... Also fixed a error in displaying test steps in the test case tables. Expanded the test cases content.
513	fred	06-12-2012	Added some detail to test cases
514	fred	06-12-2012	Adding detail to test cases.
515	fred	07-12-2012	Added lots more detail to the test cases, more yet to add.
516	fred	07-12-2012	Expanded test case table further.
517	fred	08-12-2012	Imporved Acknowledgement page, Fixed up appendix A to reflect last agreed version. Added testing results.
518	fred	08-12-2012	Moved some state related messages to the strings.xml resource. Ran some more tests and updated the test case register. Fixed a bug in state transition from ENABLE_RADIO to RADIO_ON before radio had actually completed turning on.
519	fred	08-12-2012	Expanded test case table further.
520	fred	08-12-2012	Adding support for disconnect tab.
521	fred	09-12-2012	Fixed Disconnect button, Completed test case log. Started adding SVN version number into build.
522	fred	09-12-2012	Finished hooking up code to inject SVN version number into application to simplify testing.
523	fred	09-12-2012	tried to include details for working copy version.
524	fred	09-12-2012	added text to structural testing.
525	fred	10-12-2012	Working on the design and introduction. Fixing figure display issues.
526	fred	10-12-2012	replaced app screen grab with more current version.
527	fred	10-12-2012	Almost completed the testing chapter.
528	fred	11-12-2012	Moved test cases to Appendix D from the testing chapter
529	fred	11-12-2012	Removed the USBHID and Work Completed chapters. (Content added to existing chapters.) Reworked the nomenclature into longtable format (Trying to remove a blank page) Inserted Chapter D into the build. Other minor changes.

continued . . .

. . . continued

Rev	Who	Date	Description
530	fred	13-12-2012	Added autoref's into introduction to bring the content description inline with the actual chapter numbers and titles. Fixed a few misc things.
531	fred	14-12-2012	Adding in Introduction edits. Modified source image for some figures in the introduction chapter.
532	fred	14-12-2012	Finished first edit of introduction. Moved android versions to applications.tex. Started expanding the applications chapter.
533	fred	14-12-2012	Working on the applications chapter.
534	fred	15-12-2012	Checking in nights changes. Started work on the Applications section.
535	fred	16-12-2012	Expanding the applications chapter further.
536	fred	16-12-2012	Further expansion of the Applciations section. Added XMl to nomencl.tex
537	fred	16-12-2012	Expanding the applications chapter.
538	fred	17-12-2012	Checking in changes to applications chapter
539	fred	17-12-2012	Added a link to be added soon. Idc graph.
540	fred	17-12-2012	just updated the PDF
541	fred	18-12-2012	Major work arranging information in the applications, design and introduction chapters.
542	fred	20-12-2012	Adding lots of edits, Thanks Katherine, John, Katie and Josh.
543	fred	21-12-2012	Added lots of edits.
544	fred	21-12-2012	Committing tonight's changes, worked on the conclusion, design and review sections.
545	fred	21-12-2012	Adding more work in progress.
546	fred	21-12-2012	Completed the chapter summary in applications chapter. Fixed the literature review introduction.
547	fred	22-12-2012	Added extra information to the review chapter.
548	fred	22-12-2012	Adding a review of the testing literature.
549	fred	22-12-2012	Corrected typos and fixed layout in review chapters testing section.

continued . . .

. . . continued

Rev	Who	Date	Description
550	fred	22-12-2012	Minor edit to the review section.
551	fred	22-12-2012	Minor edit
552	fred	23-12-2012	Minor edit to the bluetooth section, relating to UUID
553	fred	26-12-2012	Added a missing fig image.
554	fred	28-12-2012	Working on reducing the size of the svn revision appendix, by reducing font size and tuning column widths. Also merged in a version conflict in the Bluetooth chapter (Due to updating on my Android tablet and the Windows box without committing the other first.)
555	fred	28-12-2012	Finished the Bluetooth chapter, UUID's, Sockets and Introduction. Modified the detail included into the svn log. Minor changes to the Introduction, Appendix B and C bluetooth-¿Bluetooth.
556	fred	28-12-2012	fixed a few index issues.
557	fred	29-12-2012	Major edit, fixing Source Code appendix descriptions and listing labels. Adding Debugging to the Applications chapter. Modifications to the Introduction, Review, Testing and Conclusion chapter. Other minor edits.
558	fred	29-12-2012	WIP, Minor edit.
559	fred	29-12-2012	Fixes to the design chapter.
560	fred	29-12-2012	Added a PDF figure to replace the JPEG version for the IME life-cycle figure. Added final content, edit phase is next.
561	fred	29-12-2012	Minor edits to make IME figure text more legible. Adjusted the index, removed redundant references to SQL, since they are captured by the SQLite entry.
562	fred	30-12-2012	Near final edit, corrected many typos. Modified figures to be more legible.
563	fred	30-12-2012	Final edit. Primarily using an edited copy from Katherine (Thank you). Also searched for effected/affected to confirm correct use. Capitalisations in section headings. And issues with inconsistent use of full stops.

Appendix C

Application Source Code

This code is reproduced (in as is condition) directly from its subversion repository location.

At the time of writing the project is still under active development, the most current version is securely stored in

`http://svn.houweling.com.au/svn/uni/ResearchProject/Java/BlueReadyProof`

Should the reader wish access, a request to the author, Fred Houweling <`fred@houweling.com.au`> should be made.

The code was current as at: 30th December 2012

C.1 The BlueReadyProofActivity.java Class

This class BlueReadyProofActivity.java is responsible for managing the main functionality of the application, including:

1. initialise the application.
2. connects the tabs to their respective activities.
3. creates the handler for interprocess communications.
4. creates a thread to handle long running operations.
5. create and process the application menu.
6. receives system intents and forwards them to other classes.

Listing C.1: Application Main Activity

```
1  /*
2   * Always Blue Ready
3   * Copyright (C) 2012
4   * By Fred Houweling
5   * fred@houweling.com.au
6   * All Rights Reserved
7   * Version 1.01 - 14 Sep 2012
8   *
9   * Description:
10  * This is the main application activity, it is responsible
11  * for generating the tabed view that hosts the other activities.
12  * as well as kicking of the thread that handles our long running tasks.
13  * and processes messages for the UI.
14  *
15  * Revision:
16  * Date          Version  Who         Comments
17  * ============  =======  ==========  ========================================
18  * 14 Sep 2012     1.01   Fred        Cleanup.
19  * 17 Mar 2012     1.00   Fred        Initial base version.
20  */
21 package au.com.houweling.BlueReadyProof;

23 import java.util.ArrayList;
24 import java.util.List;
25 import java.util.concurrent.atomic.AtomicBoolean;

27 import android.app.Activity;
28 import android.app.AlertDialog;
29 import android.app.TabActivity;
30 import android.content.DialogInterface;
31 import android.content.DialogInterface.OnClickListener;
32 import android.content.Intent;
33 import android.content.res.Resources;
34 import android.database.sqlite.SQLiteDatabase;
35 import android.os.Bundle;
36 import android.os.Handler;
37 import android.os.Looper;
38 import android.os.Message;
39 import android.util.Log;
40 import android.view.Menu;
41 import android.view.MenuItem;
```

```
42 import android.view.View;
43 import android.widget.TabHost;
44 import android.widget.TabHost.OnTabChangeListener;
45 import au.com.houweling.BlueReadyProof.BlueReadyStateBasedMachine.BLUESTATES;

47 public class BlueReadyProofActivity extends TabActivity implements
48         OnTabChangeListener {
49     final static String AppName = "BlueReadyProof";
50     private BlueReadyDBHelper brDatabase;
51     private SQLiteDatabase brDb;
52     private static TabHost myTabHost;
53     Resources myResourses;
54     public static Thread btt;
55     public static BlueReadyStateBasedMachine bsbm = new
          BlueReadyStateBasedMachine();
56     AtomicBoolean isRunning = new AtomicBoolean(false);
57     List<Activity> childActivities = new ArrayList<Activity>(2);
58     MenuItem mitem;
59     MenuItem mrefresh;

61     public static void switchTab(String tag) {
62         if (myTabHost != null) {
63             myTabHost.setCurrentTabByTag(tag);
64         }
65     }

67     static Handler handler = new Handler() {
68         @Override
69         public void handleMessage(Message msg) {
70             if (msg != null) {
71                 switch (msg.what) {
72                 case 0:
73                 case 1: // State messages
74                     if (msg.arg2 == 1) {
75                         ConnectActivity.requery();
76                     }
77                     ConnectActivity.setButtonState(BlueReadyStateBasedMachine
78                             .compareIntToState(msg.arg1, BLUESTATES.RADIO_ON));
79                     String messageText = "Current State: "
80                             + BlueReadyStateBasedMachine
81                                     .getStatusString(msg.arg1);
82                     TestingActivity.setText(messageText);
83                     ConnectActivity.setText(messageText);
84                     break;
85                 case 2: // Incoming keyboard data.
86                     TestingActivity.insertText((String) msg.obj);
87                     break;
88                 case 4: // Something wants to transition into a new state
89                     bsbm.setTargetState(BLUESTATES.values()[msg.arg1]);
90                     break;
91                 case 2000:
92                     if (String.class.equals(msg.obj.getClass())) {
93                         // TODO: Disable spinner, enable disconnect tab
94                         BlueReadyProofActivity.bsbm
95                                 .setTargetState(BLUESTATES.values()[msg.arg1]);
                                    //CONNECTING);
96                         switchTab((String) msg.obj);
97                     }
98                     break;
99                 default:
100                    break;
101                }
102            } else {
103                Log.w("ABR.handleMessage", "Odd, got a null msg in the Handler");
104            }
105        }

107    };
```

```
109     public void registerTab(Activity act) {
110         if (act != null) {
111             childActivities.add(act);
112         }
113     }
115     public void unregisterTab(Activity act) {
116         if (act != null) {
117             childActivities.remove(act);
118         }
119     }
121     // AtomicBoolean threadStarted = false;
122     /** Called when the activity is first created. */
123     @Override
124     public void onCreate(Bundle savedInstanceState) {
125         super.onCreate(savedInstanceState);
126         setContentView(R.layout.main);
127         myTabHost = getTabHost();
128         myResourses = getResources();
129         myTabHost.addTab(AddTab("Connection", "connect",
130                 R.drawable.ic_launcher_brk, ConnectActivity.class));
131         myTabHost.addTab(AddTab("Testing", "test", R.drawable.ic_launcher_brk,
132                 TestingActivity.class));
133         myTabHost.addTab(AddTab("Settings", "setup",
134                 R.drawable.ic_launcher_brk, DummyActivity.class));
135         myTabHost.setOnTabChangedListener(this);
136         myTabHost.setCurrentTab(0);
137     }
139     private void truncateDb() {
140         BlueReadyDBHelper.TruncateDb(getApplicationContext(), true);
141     }
143     public void getConfirmation(View view) {
144         new AlertDialog.Builder(this)
145                 .setTitle("Database Truncation Confirmation")
146                 .setMessage("Are you sure you want to empty the database?")
147                 .setPositiveButton("Yes, Truncate", new OnClickListener() {
149                     public void onClick(DialogInterface dialog, int which) {
150                         // Truncate the database
151                         truncateDb();
152                         ConnectActivity.requery();
153                     }
154                 }).setNegativeButton("No", new OnClickListener() {
156                     public void onClick(DialogInterface dialog, int which) {
157                         // Do nothing
158                     }
159                 }).show();
160     }
162     @Override
163     public boolean onOptionsItemSelected(MenuItem item) {
164         boolean rVal = super.onOptionsItemSelected(item);
165         if (item.equals(mitem)) {
166             getConfirmation(getCurrentFocus());
167         } else if (item.equals(mrefresh)) {
168             ConnectActivity.requery();
169         }
170         return rVal;
171     }
173     @Override
174     public boolean onCreateOptionsMenu(Menu menu) {
175         boolean rVal = super.onCreateOptionsMenu(menu);
176         mitem = menu.add("Truncate Database");
```

```
177         mrefresh = menu.add("Refresh");
178         return rVal;
179     }
181     @Override
182     public void onStart() {
183         super.onStart();
185         btt = new Thread(new Runnable() {
186             public void run() {
187                 boolean done = false;
188                 int result = 1;
189                 Looper.prepare();
190                 while (done != true) {
191                     // Looper.loop();
192                     if (result != 0) { // Something of interest occurred.
193                         Message msg = handler.obtainMessage(result);
194                         if (msg != null) {
195                             msg.what = result;
196                             msg.arg1 = bsbm.getCurrentState().ordinal();
197                             msg.arg2 = bsbm.isStale() ? 1 : 0;// bsbm.
                                getTargetState().ordinal();
198                             handler.sendMessage(msg);
199                         }
200                     } else { // Nothing of interest occurred, delay...
201                         try {
202                             Thread.sleep(100);
203                         } catch (InterruptedException e) {
204                             e.printStackTrace();
205                         }
206                     }
207                     result = bsbm.process();
208                     if (result < 0) {
209                         done = true;
210                     }
211                 }
212             }
214         });
215         isRunning.set(true);
216         btt.start();
217         BlueReadyStateBasedMachine.setContext(getApplicationContext());
218         // truncateDb();
219         // DEBUG: Remove test record insertion.
220         // BlueReadyDBHelper.InsertOrUpdate(this.getApplicationContext(), null,
221         // "Test", "12:34:56:78", true, true, false, false, false, false,
222         // false);
223     }
225     @Override
226     public void onPause() {
227         super.onPause();
228         this.unregisterReceiver(BlueReadyProofActivity.bsbm.getYourReceiver());
229         BlueReadyProofActivity.bsbm.onPause();
230     }
232     @Override
233     public void onResume() {
234         super.onResume();
235         BlueReadyProofActivity.bsbm.onResume();
236         this.registerReceiver(BlueReadyProofActivity.bsbm.getYourReceiver(),
237                 BlueReadyProofActivity.bsbm.getMyiFilter());
238     }
240     @Override
241     public void onStop() {
242         BlueReadyProofActivity.bsbm.onStop();
243         isRunning.set(false);
```

```
244         super.onStop();
245     }
247     @Override
248     public void onDestroy() {
249         BlueReadyProofActivity.bsbm.onDestroy();
250         // handler.getLooper().quit();
251         super.onDestroy();
252     }
254     public TabHost.TabSpec AddTab(String myTabTitle, String myTabSpec,
255             int drawableId, Class<?> myIntent) {
256         // TODO: Implement null checks for globals etc.
257         return (myTabHost.newTabSpec(myTabSpec).setIndicator(myTabTitle,
258                 myResourses.getDrawable(drawableId)).setContent(new Intent()
259                 .setClass(this, myIntent)));
260     }
262     public void onTabChanged(String tabId) {
263         /*
264          * Trying out the default look and feel...
265         for (int index = 0; index < myTabHost.getTabWidget().getChildCount();
               index++) {
267             if (myTabHost.getTabWidget().getChildAt(index).isSelected()) {
268                 myTabHost.getTabWidget().getChildAt(index)
269                         .setBackgroundColor(Color.GRAY);
270             } else {
271                 myTabHost.getTabWidget().getChildAt(index)
272                         .setBackgroundColor(Color.BLACK);
273             }
274         }
275         */
276     }
278     public SQLiteDatabase getBrDb() {
279         return brDb;
280     }
282     public void setBrDb(SQLiteDatabase brDb) {
283         this.brDb = brDb;
284     }
286     public BlueReadyDBHelper getBrDatabase() {
287         return brDatabase;
288     }
290     public void setBrDatabase(BlueReadyDBHelper brDatabase) {
291         this.brDatabase = brDatabase;
292     }
293 }
```

C.2 The TestingActivity.java Class

This file implements the connecting and connected state Activity. It is responsible for displaying the current state, data captured from the connected keyboard and initiating the state change to disconnecting once the user has pressed the disconnect button.

Listing C.2: Testing Activity

```
/*
 * Always Blue Ready
 * Copyright (C) 2012
 * By Fred Houweling
 * fred@houweling.com.au
 * All Rights Reserved
 * Version 1.01 - 14 Sep 2012
 *
 * Description:
 * This class provides the interface to display connection information once the
    connecting state is started.
 *
 * Revision:
 * Date          Version  Who         Comments
 * ============  =======  ==========  ========================================
 * 14 Sep 2012     1.01   Fred        Cleanup.
 * 11 Sep 2012     1.00   Fred        Original version, (suspect on date.)
 */

package au.com.houweling.BlueReadyProof;

import android.app.Activity;
import android.os.Bundle;
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import au.com.houweling.BlueReadyProof.BlueReadyStateBasedMachine.BLUESTATES;

public class TestingActivity extends Activity {
    private static TextView statusView = null;
    private static CharSequence statusMessageText = null;
    private static String[] buff;

    public static void setSelectedDevice(String address, String name) {
        if (nameView != null && addressView != null) {
            if (name == null && address == null && buff != null) {
                name = buff[0];
                address = buff[1];
            }
            // Pass on the selected address to the state based machine.
            BlueReadyProofActivity.bsbm.setAddress(address);
            nameView.setText(name);
            addressView.setText(address);
        }
        buff = new String[] { name, address };
    }

    private static TextView addressView;
    private static TextView nameView;
    private static EditText keycapturelog;
    private Button disconnectButton;

    public static boolean insertText(String buffer) {
        if (keycapturelog == null) {
            return (false);
        }
        keycapturelog.append(buffer);
        return (true);
    }

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.testing_tab);
        nameView = (TextView) findViewById(R.id.testingName);
        addressView = (TextView) findViewById(R.id.testingAddress);
        statusView = (TextView) findViewById(R.id.TestingStatus);
        keycapturelog = (EditText) findViewById(R.id.keycapturelog);
```

```
71          disconnectButton = (Button) findViewById(R.id.disconnectButton);
72          disconnectButton.setOnClickListener(new OnClickListener() {

74              public void onClick(View v) {
75                  // TODO disable ...
76                  if(BlueReadyProofActivity.handler!=null)
77                  {
78                      Message msg = BlueReadyProofActivity.handler.obtainMessage();
79                      if(msg!=null)
80                      {
81                          msg.what = 2000;
82                          msg.obj = "connect";
83                          msg.arg1 = BLUESTATES.RADIO_ON.ordinal();
84                          BlueReadyProofActivity.handler.dispatchMessage(msg);
85                      }
86                  }

88              }
89          });
90          // Ensure any cached text strings get displayed.
91          setText(null);
92          setSelectedDevice(null, null);
93      }

95      public static void setText(CharSequence text) {
96          if (statusView != null) { // Check if tv is not null
97              if (text != null) {
98                  statusView.setText(text);
99              } else if (statusMessageText != null) {
100                 statusView.setText(statusMessageText);
101                 statusMessageText = null;
102             }

104         } else { // Class instance onCreate not yet called, cache the text for
105                  // later.
106             statusMessageText = text;
107         }
108     }

110 }
```

C.3 The blueToothConnection.java Class

Class to manage Bluetooth socket connections to the selected hardware.

Listing C.3: Bluetooth Socket Code

```
1 /*
2  * Always Blue Ready
3  * Copyright (C) 2012
4  * By Fred Houweling
5  * fred@houweling.com.au
6  * All Rights Reserved
7  * Version 1.01 - 14 Sep 2012
8  *
9  * Description:
```

```
10  * This class controls the actual connection to a bluetooth device.
11  *
12  * Revision:
13  * Date          Version  Who         Comments
14  * ============  =======  ==========  ============================
15  * 14 Sep 2012     1.01   Fred        Cleanup.
16  * 12 Sep 2012     1.00   Fred        Original version.
17  */
18 package au.com.houweling.BlueReadyProof;

20 import java.io.IOException;
21 import java.io.InputStream;
22 import java.io.OutputStream;
23 import java.lang.reflect.Constructor;
24 import java.lang.reflect.InvocationTargetException;
25 import java.lang.reflect.Method;
26 import java.util.UUID;

28 import android.bluetooth.BluetoothDevice;
29 import android.bluetooth.BluetoothSocket;
30 import android.os.ParcelUuid;
31 import android.util.Log;

33 public class blueToothConnection {
34     private static final int TYPE_L2CAP = 3;
35     private BluetoothSocket bs;
36     // private BluetoothServerSocket bss;
37     private BluetoothDevice dev;
38     private OutputStream oStream;
39     private InputStream iStream;
40     private ParcelUuid[] uuids;
41     private int uuidindex = 0;
42     private static String prefered_uuid = "00001124-0000-1000-8000-00805F9B34FB";
43     // Hide default constructor
44     @SuppressWarnings("unused")
45     private blueToothConnection() {
46         // Empty
47     }

49     public blueToothConnection(BluetoothDevice dev) {
50         this.dev = dev;
51     }

53     public boolean connect() {
54         // Save the device the user selected.
55         // myBtDevice = btDevicesFound.get( arg2 );
56         // Query the device's services
57         if(dev == null)
58             throw new NullPointerException("Can't connect with a null device" );
59         if(uuids == null)
60         {
61             uuids = servicesFromDevice(dev);
62         }

64         // Open a socket to connect to the device chosen.
65         try {
66             //TODO: Work out the port we should be using.
67             // as per: http://www.bluetooth.org/Technical/AssignedNumbers/
                   service_discovery.htm
68             // 0x0011? or 0x1124?
69             //int port = 0x0011; // Should probably get this number from the SDP
                   database.
70             int port = 0x1124;
71             if(uuids == null)
72             {
73                 bs = createL2CAPBluetoothSocket(dev.getAddress(),port);
74                 // The following won't work because it creates a RFCOMM
                       connection.
75                 //bs = dev.createRfcommSocketToServiceRecord(UUID.fromString
                       ("00001124-0000-1000-8000-00805F9B34FB"));
76             }
```

```
77              else
78              {
79                  bs = createL2CAPBluetoothSocket(dev.getAddress(),port);
80                  // The following won't work because it creates a RFCOMM
                        connection.
81                  //bs = dev.createRfcommSocketToServiceRecord(uuids[uuidindex].
                        getUuid());
82              }
84          //} catch (IOException e) {
85          //  Log.e("abr.Bluetooth Socket",
86          //          "Bluetooth not available, or insufficient permissions");
87          } catch (NullPointerException e) {
88              Log.e("abr.Bluetooth Socket", "Null Pointer One\n"+e.getMessage());
89              return false;
90          }
92  //      bs = createL2CAPBluetoothSocket(dev.getAddress(), 0x0011);
93          try {
94              bs.connect();
95          } catch (IOException ea) {
96              // TODO: Falling back to RFCOMM is not what we want.
97              Method m;
98              try {
99                  m = dev.getClass().getMethod("createRfcommSocket", new Class[] {
                        int.class});
100                 bs = (BluetoothSocket) m.invoke(dev, 1);
101             } catch (SecurityException e) {
102                 Log.e("abr.Bluetooth Socket",
103                         "Permission Error or Other error trying to connect\n"+e.
                            getMessage());
104                 return false;
105             } catch (NoSuchMethodException e) {
106                 Log.e("abr.Bluetooth Socket",
107                         "Permission Error or Other error trying to connect\n"+e.
                            getMessage());
108                 return false;
109             }  catch (IllegalArgumentException e) {
110                 Log.e("abr.Bluetooth Socket",
111                         "Permission Error or Other error trying to connect\n"+e.
                            getMessage());
112                 return false;
113             } catch (IllegalAccessException e) {
114                 Log.e("abr.Bluetooth Socket",
115                         "Permission Error or Other error trying to connect\n"+e.
                            getMessage());
116                 return false;
117             } catch (InvocationTargetException e) {
118                 Log.e("abr.Bluetooth Socket",
119                         "Permission Error or Other error trying to connect\n"+e.
                            getMessage());
120                 return false;
121             }
122         }
124         return true;
125     }
127     public boolean close() {
128         if(oStream!=null)
129         {
130             try {
131                 oStream.close();
132             } catch (IOException e) {
133                 Log.e("abr.Bluetooth Socket",
134                         "Error trying to close connection oStream\n"+e.
                            getMessage());
135             }
```

```
136             set_iStream(null);
137         }
138         if(get_iStream()!=null)
139         {
140             try {
141                 get_iStream().close();
142             } catch (IOException e) {
143                 Log.e("abr.Bluetooth Socket",
144                         "Error trying to close connection iStream\n"+e.
                            getMessage());
145             }
146             set_iStream(null);
147         }
148         if (bs != null) {
149             try {
150                 bs.close();
151             } catch (IOException e) {
152                 Log.e("abr.Bluetooth Socket",
153                         "Permission Error or Other error trying to connect\n"+e.
                            getMessage());
154             }
155             bs = null;
156         }
157         return false;
158     }
160     public OutputStream get_oStream() {
161         try {
162             set_oStream(bs.getOutputStream());
163         } catch (IOException e) {
164             Log.e("abr.Bluetooth Socket",
165                     "Permission Error or Other error trying to connect\n"+e.
                        getMessage());
166             return null;
167         }
168         return(oStream);
169     }
171     private void set_oStream(OutputStream oStream) {
172         this.oStream = oStream;
173     }
175     public InputStream get_iStream() {
176         try {
177             set_iStream(bs.getInputStream());
178         } catch (IOException e) {
179             Log.e("abr.Bluetooth Socket",
180                     "Permission Error or Other error trying to connect\n"+e.
                        getMessage());
181             return null;
182         }
183         return iStream;
184     }
186     private void set_iStream(InputStream iStream) {
187         this.iStream = iStream;
188     }
189     public int getUuidindex() {
190         return uuidindex;
191     }
193     public void setUuidindex(int uuidindex) {
194         if(uuids==null)
195         {
196             getUuids();
197         }
198         if(uuids==null||uuidindex>uuids.length||uuidindex<0)
199         {
```

```
200             // TODO: Need to throw an exception?
201             return;
202         }
203         this.uuidindex = uuidindex;
204     }
206     /**
207      * @return the uuids
208      */
209     public ParcelUuid[] getUuids() {
210         if(uuids==null)
211             uuids = servicesFromDevice();
212         return uuids;
213     }
214     public ParcelUuid[] servicesFromDevice()
215     {
216         uuids = servicesFromDevice(dev);
217         for(int i=0;i<uuids.length;i++)
218         {
219             if(uuids[i].equals(prefered_uuid))
220             {
221                 uuidindex = i;
222             }
223         }
224         return uuids;
225     }
226     /*
227      * From: http://stackoverflow.com/questions/4263101/how-to-talk-to-a-
           bluetooth-keyboard
228      */
229     public static BluetoothSocket createL2CAPBluetoothSocket(String address, int
           psm){
230         return createBluetoothSocket(TYPE_L2CAP, -1, false,false, address, psm);
231     }
232     // method for creating a bluetooth client socket
233     private static BluetoothSocket createBluetoothSocket(
234             int type, int fd, boolean auth, boolean encrypt, String address, int
                port){
235         try {
236             Constructor<BluetoothSocket> constructor = BluetoothSocket.class.
                   getDeclaredConstructor(
237                     int.class, int.class,boolean.class,boolean.class,String.
                        class, int.class);
238             constructor.setAccessible(true);
239             BluetoothSocket clientSocket = (BluetoothSocket)
240                 constructor.newInstance(type,fd,auth,encrypt,address,port);
241             return clientSocket;
242         }catch (Exception e) { return null; }
243     }
245     /*
246      * Source: http://stackoverflow.com/questions/11003280/finding-uuids-in-
           android-2-0
247      */
248     //In SDK15 (4.0.3) this method is now public as
249     //Bluetooth.fetchUuisWithSdp() and BluetoothDevice.getUuids()
250     public static ParcelUuid[] servicesFromDevice(BluetoothDevice device) {
251         try {
252             Class<?> cl = Class.forName("android.bluetooth.BluetoothDevice");
253             Class<?>[] par = {};
254             Method method = cl.getMethod("getUuids", par);
255             Object[] args = {};
256             ParcelUuid[] retval = (ParcelUuid[]) method.invoke(device, args);
257             return retval;
258         } catch (Exception e) {
259             e.printStackTrace();
260         }
```

```
261         return null;
262     }
264     public static void startServiceDiscovery( BluetoothDevice device ) {
265         // Need to use reflection prior to API 15
266         Class<?> cl = null;
267         try {
268             cl = Class.forName("android.bluetooth.BluetoothDevice");
269         } catch( ClassNotFoundException exc ) {
270             Log.e("ABR.ssd", "android.bluetooth.BluetoothDevice not found." );
271         }
272         if (null != cl) {
273             Class<?>[] param = {};
274             Method method = null;
275             try {
276                 method = cl.getMethod("fetchUuidsWithSdp", param);
277             } catch( NoSuchMethodException exc ) {
278                 Log.e("ABR.ssd", "fetchUuidsWithSdp not found." );
279             }
280             if (null != method) {
281                 Object[] args = {};
282                 try {
283                     method.invoke(device, args);
284                 } catch (Exception exc) {
285                     Log.e("ABR.ssd", "Failed to invoke fetchUuidsWithSdp method.
                        " );
286                 }
287             }
288         }
289     }
290 }
```

C.4 The `BlueReadyDBHelper.java` Class

This class contains table create and upgrade functions. As well as functions to perform record Create, Read, Update and Delete (CRUD) operations.

Listing C.4: Database Abstraction

```
1  /*
2   * Always Blue Ready
3   * Copyright (C) 2012
4   * By Fred Houweling
5   * fred@houweling.com.au
6   * All Rights Reserved
7   * Version 1.01 - 12 Sep 2012
8   *
9   * Description:
10  * This class is a SQLite helper class.
11  *
12  * Revision:
13  * Date          Version  Who         Comments
14  * ============  =======  ==========  ================================
15  * 12 Sep 2012     1.01   Fred        Added CRUD code.
16  * 31 Jul 2012     1.00   Fred        Original version.
```

```
17  */
19 package au.com.houweling.BlueReadyProof;
21 import java.util.ArrayList;
22 import java.util.List;
24 import android.content.ContentValues;
25 import android.content.Context;
26 import au.com.houweling.BlueReadyProof.BlueReadyDB.BRkb_uuid_map;
27 import au.com.houweling.BlueReadyProof.BlueReadyDB.BRuuid;
28 import au.com.houweling.BlueReadyProof.brKeyboard;
29 import android.database.Cursor;
30 import android.database.sqlite.SQLiteDatabase;
31 import android.database.sqlite.SQLiteDatabase.CursorFactory;
32 import android.database.sqlite.SQLiteOpenHelper;
33 import android.util.Log;
34 import au.com.houweling.BlueReadyProof.BlueReadyDB.BRKeyboard;
36 public class BlueReadyDBHelper extends SQLiteOpenHelper {
37     private static final String DATABASE_NAME = "blueready.db";
38     private static final int DATABASE_VERSION = 2;
39     public BlueReadyDBHelper(Context context, String name,
40             CursorFactory factory, int version) {
41         super(context, name, factory, version);
43     }
44     public BlueReadyDBHelper(Context context) {
45         super(context, DATABASE_NAME, null, DATABASE_VERSION);
46     }
47     public void onOpen(SQLiteDatabase db)
48     {
49         super.onOpen(db);
50     }
52     @Override
53     public void onCreate(SQLiteDatabase db) {
54         createKeyboardTable(db);
55         createUUIDTables(db);
56     }
57     /**
58      * @param db
59      */
60     private void createKeyboardTable(SQLiteDatabase db) {
61         db.execSQL("CREATE TABLE " + BRKeyboard.BR_TABLE_NAME + " (" +
62                 BRKeyboard._ID                  + " INTEGER PRIMARY KEY
                    AUTOINCREMENT, " +
63                 BRKeyboard.BR_KB_NAME           + " TEXT, " +
64                 BRKeyboard.BR_KB_ADDRESS        + " TEXT NOT NULL UNIQUE, " +
65                 BRKeyboard.BR_KB_CREATED        + " TEXT NOT NULL DEFAULT (
                    datetime('now') ), " +
66                 BRKeyboard.BR_KB_LAST_SEEN      + " TEXT NOT NULL DEFAULT (
                    datetime('now') ), " +
67                 BRKeyboard.BR_KB_LAST_QUERIED   + " TEXT, " +
68                 BRKeyboard.BR_KB_IS_SDP_DONE    + " INTEGER DEFAULT 'false', " +
69                 BRKeyboard.BR_KB_IS_ON_AIR      + " INTEGER DEFAULT 'false', " +
70                 BRKeyboard.BR_KB_IS_HID         + " INTEGER DEFAULT 'false', " +
71                 BRKeyboard.BR_KB_IS_PAIRED      + " INTEGER DEFAULT 'false', " +
72                 BRKeyboard.BR_KB_IS_PREFERED    + " INTEGER DEFAULT 'false', " +
73                 BRKeyboard.BR_KB_CONNECT_COUNT  + " INTEGER DEFAULT '0'" +
74                 ");");
75         db.execSQL("create index IF NOT EXISTS idx_" + BRKeyboard.BR_TABLE_NAME
               + "_" + BRKeyboard.BR_KB_ADDRESS + " ON "+BRKeyboard.BR_TABLE_NAME+" (
               "+ BRKeyboard.BR_KB_ADDRESS +")");
76         //db.execSQL("create index IF NOT EXISTS idx_" + BRKeyboard.
               BR_TABLE_NAME + "_" + BRKeyboard._ID + " ON "+BRKeyboard.BR_TABLE_NAME
               +" ("+ BRKeyboard._ID +")");
77     }
78     private void createUUIDTables(SQLiteDatabase db) {
79         db.execSQL("CREATE TABLE " + BRuuid.BR_TABLE_NAME + " (" +
```

```
80                  BRuuid._ID                          + " INTEGER PRIMARY KEY
                        AUTOINCREMENT, " +
81                  BRuuid.BR_UUID                      + " TEXT NOT NULL UNIQUE, " +
82                  BRuuid.BR_CREATED                   + " TEXT NOT NULL DEFAULT (
                        datetime('now') ), " +
83                  BRuuid.BR_CONNECT_COUNT             + " INTEGER DEFAULT '0'" +
84                  ");");
85          db.execSQL("CREATE TABLE " + BRkb_uuid_map.BR_TABLE_NAME + " (" +
86                  BRkb_uuid_map._ID                   + " INTEGER PRIMARY KEY
                        AUTOINCREMENT, " +
87                  BRkb_uuid_map.BR_CREATED            + " TEXT NOT NULL DEFAULT (
                        datetime('now') ), " +
88                  BRkb_uuid_map.BR_UUID_ID            + " TEXT NOT NULL, " + // Refers
                        to BRuuid._ID
89                  BRkb_uuid_map.BR_KB_ID              + " TEXT NOT NULL" + // Refers
                        to BRKeyboard._ID
90                  ");");
91          db.execSQL("create index IF NOT EXISTS idx_" + BRuuid.BR_TABLE_NAME + "
                ON "+BRuuid.BR_TABLE_NAME+" ("+ BRuuid.BR_UUID +")");
92          db.execSQL("create index IF NOT EXISTS idx_" + BRkb_uuid_map.
                BR_TABLE_NAME + " ON "+BRkb_uuid_map.BR_TABLE_NAME+" ("+ BRkb_uuid_map
                .BR_KB_ID + ", " + BRkb_uuid_map.BR_UUID_ID +")");
93      }
95      @Override
96      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
97          db.beginTransaction();
98          switch(oldVersion)
99          {
100             case 1:
101                 // Insert the new connects counter
102                 db.execSQL("ALTER TABLE " + BRKeyboard.BR_TABLE_NAME + " ADD " +
                        BRKeyboard.BR_KB_CONNECT_COUNT + " INTEGER DEFAULT '0'");
103                 //db.execSQL("UPDATE TABLE " + BRKeyboard.BR_TABLE_NAME + " SET
                        " + BRKeyboard.BR_KB_CONNECT_COUNT + "='0' where " +
                        BRKeyboard.BR_KB_CONNECT_COUNT + " is null");
104                 db.execSQL("create index IF NOT EXISTS idx_" + BRKeyboard.
                        BR_TABLE_NAME + "_" + BRKeyboard.BR_KB_ADDRESS + " ON "+
                        BRKeyboard.BR_TABLE_NAME+" ("+ BRKeyboard.BR_KB_ADDRESS +")");
105                 //db.execSQL("create index IF NOT EXISTS idx_" + BRKeyboard.
                        BR_TABLE_NAME + "_" + BRKeyboard._ID + " ON "+BRKeyboard.
                        BR_TABLE_NAME+" ("+ BRKeyboard._ID +")");
106                 // add the new tables: tb_br_uuid, tb_br_kb_uuid_map
107                 createUUIDTables(db);
108                 break;
109             case DATABASE_VERSION:
110                 // Do nothing
111                 break;
112         }
113         db.setTransactionSuccessful();
114         db.endTransaction();
115         // Rebuild the database.
116         //db.execSQL("VACUUM"); We are inside a transaction, not allowed...
117     }
118     private static boolean SQLValid;
119     public static void TruncateDb(Context context, boolean sure)
120     {
121         if(context!=null&&sure)
122         {
123             BlueReadyDBHelper brDatabase = new BlueReadyDBHelper(context);
124             SQLiteDatabase BrDb = brDatabase.getWritableDatabase();
125             //BrDb.beginTransaction();
126             BrDb.execSQL("delete from "+BRKeyboard.BR_TABLE_NAME);
127             //BrDb.setTransactionSuccessful();
128             //BrDb.endTransaction();
129             BrDb.close();
130             brDatabase.close();
```

```
131         }
132     }
133     /**
134      *
135      * @param context The calling application context, passed onto {@link
           BlueReadyDBHelper} which has {@link SQLiteOpenHelper} as the super class.
136      * @param _id Cannot be null if address is null, must match a valid database
           recored PK
137      * @param name The textual description of the Bluetooth device.
138      * @param address The Bluetooth address of the device, cannot be null if _id
           is null.
139      * @param updateLastseen Set to true to force the records date to be set to
           the current timestamp.
140      * @param updateLastQueried Set to true to force the records date to be set
           to the current timestamp.
141      * @param isSdpDone The value of the SDP flag, if null no update will occur
           for this field.
142      * @param isOnAir The value of the OnAir flag, if null no update will occur
           for this field.
143      * @param isHid The value of the HID flag, if null no update will occur for
           this field.
144      * @param isPaired The value of the Paired flag, if null no update will
           occur for this field.
145      * @param isPreferred The value of the Preferred flag, if null no update
           will occur for this field.
146      * @return The value returned is the effected records PK.
147      */
148     public static String InsertOrUpdate(Context context,
149             String _id,
150             String name,
151             String address,
152             Boolean updateLastseen,
153             Boolean updateLastQueried,
154             Boolean isSdpDone,
155             Boolean isOnAir,
156             Boolean isHid,
157             Boolean isPaired,
158             Boolean isPreferred
159             )
160     {
161         if(context == null || (_id == null || _id.length()<=0) && (address ==
               null || address.length()<=0))
162         {
163             // TODO: Decide how to handle this, probably have information to add
                   ...
164             // Throw an exception?
165             return null;
166         }
167         BlueReadyDBHelper brDatabase = new BlueReadyDBHelper(context);
168         SQLiteDatabase BrDb = brDatabase.getWritableDatabase();
169         BrDb.beginTransaction();
170         String[] returnedColumns = {BRKeyboard.BR_TABLE_NAME+"."+BRKeyboard._ID};
171         // Search for device with same address or _id.
172         Cursor result;
173         if(_id!=null && _id.length()>0)
174         {
176             result = BrDb.query(BRKeyboard.BR_TABLE_NAME, returnedColumns,
                       BRKeyboard._ID + "=?",
177                     new String[] { "'"+_id.replaceAll("'", "''")+"'" },
178                             null, null, null);
179         }
180         else
181         {
182             result = BrDb.query(BRKeyboard.BR_TABLE_NAME, returnedColumns,
                       BRKeyboard.BR_KB_ADDRESS + "=?",
183                     // BRKeyboard._ID + "=? or " + BRKeyboard.BR_KB_ADDRESS +
                           "=?",
184                     new String[] { address.replaceAll("'", "''") },
185                             null, null, null);
```

```
186         }
187         String dbSQL = "";
188         String dataSQL = "";
190         String return_ID=null;
191         if(result!=null)
192         {
193             result.moveToFirst();
194             if(result.getCount()==1)          // if found update record with new
                    data.
195             {
196                 return_ID=result.getString(result.getColumnIndex(BRKeyboard._ID)
                        );
197                 SQLValid = false; // Flag the confirm the resulting SQL
                        statement should be executed, also used to confirm
198                 if(updateLastQueried!=null && updateLastQueried == true)
199                 {
200                     dataSQL += BRKeyboard.BR_KB_LAST_QUERIED + "=datetime('now')
                            ";
201                     SQLValid = true;
202                 }
204     //          Boolean updateLastseen,
205                 if(updateLastseen!=null && updateLastseen == true)
206                 {
207                     if(SQLValid==true)
208                     {
209                         dataSQL+=", ";
210                     }
211                     else
212                     {
213                         SQLValid=true;
214                     }
215                     dataSQL += BRKeyboard.BR_KB_LAST_SEEN + "=datetime('now') ";
216                 }
217     //          String address,
218                 dataSQL = dataSQLSetter(BRKeyboard.BR_KB_ADDRESS, address,
                        dataSQL);
219     //          String name,
220                 dataSQL = dataSQLSetter(BRKeyboard.BR_KB_NAME, name, dataSQL);
221     //          Boolean isSdpDone,
222                 dataSQL = dataSQLSetter(BRKeyboard.BR_KB_IS_SDP_DONE, isSdpDone,
                        dataSQL);
223     //          Boolean isOnAir,
224                 dataSQL = dataSQLSetter(BRKeyboard.BR_KB_IS_ON_AIR, isOnAir,
                        dataSQL);
225     //          Boolean isHid,
226                 dataSQL = dataSQLSetter(BRKeyboard.BR_KB_IS_HID, isHid, dataSQL);
227     //          Boolean isPaired,
228                 dataSQL = dataSQLSetter(BRKeyboard.BR_KB_IS_PAIRED, isPaired,
                        dataSQL);
229     //          Boolean isPreferred
230                 dataSQL = dataSQLSetter(BRKeyboard.BR_KB_IS_PREFERED,
                        isPreferred, dataSQL);
231                 // Update
232                 dbSQL = "update "+ BRKeyboard.BR_TABLE_NAME + " set ";
233                 dbSQL += dataSQL;
234                 dbSQL += "where "+ BRKeyboard._ID + "='"+return_ID+"'";
235             }
236             else     // if not insert a new record.
237             {
238                 // Insert
239                 SQLValid = false;
240                 dbSQL = "insert into " + BRKeyboard.BR_TABLE_NAME + "(";
241                 dbSQL+=dataSQLFormatter(BRKeyboard.BR_KB_ADDRESS,        address,
                            !SQLValid, false);
```

```
242                 dbSQL+=dataSQLFormatter(BRKeyboard.BR_KB_NAME,              name,
                            !SQLValid, false);
243                 dbSQL+=dataSQLFormatter(BRKeyboard.BR_KB_IS_SDP_DONE,
                        isSdpDone,  !SQLValid, false);
244                 dbSQL+=dataSQLFormatter(BRKeyboard.BR_KB_IS_ON_AIR,      isOnAir,
                            !SQLValid, false);
245                 dbSQL+=dataSQLFormatter(BRKeyboard.BR_KB_IS_HID,         isHid,
                            !SQLValid, false);
246                 dbSQL+=dataSQLFormatter(BRKeyboard.BR_KB_IS_PAIRED,      isPaired
                        ,   !SQLValid, false);
247                 dbSQL+=dataSQLFormatter(BRKeyboard.BR_KB_IS_PREFERED,
                        isPreferred,!SQLValid, false);
248                 dbSQL+=") values (";
249                 SQLValid = false;   // Need to reset so first element is
                        detected correctly, and
250                                     // will only be set if at least 1 item is
                                            valid anyway.
251                 dbSQL+=dataSQLFormatter(address,     address,     !SQLValid, true);
252                 dbSQL+=dataSQLFormatter(name,        name,        !SQLValid, true);
253                 dbSQL+=dataSQLFormatter(isSdpDone,   isSdpDone,   !SQLValid, true);
254                 dbSQL+=dataSQLFormatter(isOnAir,     isOnAir,     !SQLValid, true);
255                 dbSQL+=dataSQLFormatter(isHid,       isHid,       !SQLValid, true);
256                 dbSQL+=dataSQLFormatter(isPaired,    isPaired,    !SQLValid, true);
257                 dbSQL+=dataSQLFormatter(isPreferred,isPreferred,!SQLValid, true);
258                 dbSQL+=")";
259             }
260             Log.i("ABR.Sql", dbSQL);
261             // Do update / insert operations here.
262             if(SQLValid==true)
263             {
264                 BrDb.execSQL(dbSQL);
265             }
266             // Done, close the DB and exit.
267         }
268         BrDb.setTransactionSuccessful();
269         BrDb.endTransaction();
270         BrDb.close();
271         brDatabase.close();
272         return return_ID;
273     }
274     /**
275      * Used to correctly prefix and wrap data in single quotations for inclusion
              in database insert strings.
276      * @param input the String to use, will escape single quotes to double.
277      * @param control if Null nothing will be generated
278      * @param isFirst   the first entry is never prefixed with a comma.
279      * @param useQuotes false if this is a field name true if it is for data.
280      * @return The processed string.
281      */
282     private static String dataSQLFormatter(Object input, Object control, boolean
            isFirst, boolean useQuotes)
283     {
284         if(input==null||control==null)
285             return "";
286         String data="";
287         if(input.getClass().equals(String.class))
288         {
289             data = (String)input;
290         }
291         if(input.getClass().equals(Boolean.class))
292         {
293             data = (Boolean)input==true?"true":"false";
294         }
295         data = data.replaceAll("'", "''");
296         if(useQuotes==true)
297         {
```

```
298             data = "'" + data + "'";
299         }
300         SQLValid = true;
301         if(isFirst==false)
302         {
303             return ", " + data;
304         }
305         return data;
306     }
307     /**
308      * @param fieldName
309      * @param fieldData
310      * @param dataSQL
311      * @return
312      */
313     private static String dataSQLSetter(String fieldName, String fieldData,
        String dataSQL) {
314         if(fieldData!=null)
315         {
316             if(SQLValid==true)
317             {
318                 dataSQL+=", ";
319             }
320             else
321             {
322                 SQLValid=true;
323             }
324             String data="= null";
325             if(fieldData.length()!=0)
326             {
327                 data = " = '" + fieldData.replaceAll("'", "''") + "' ";
328             }
329             dataSQL += fieldName + data;
330         }
331         return dataSQL;
332     }
333     private static String dataSQLSetter(String fieldName, Boolean fieldData,
        String dataSQL) {
334         if(fieldData!=null)
335         {
336             if(SQLValid==true)
337             {
338                 dataSQL+=", ";
339             }
340             else
341             {
342                 SQLValid=true;
343             }
344             dataSQL += fieldName + " = '" + fieldData + "' ";
345         }
346         return dataSQL;
347     }
348     // Adding CRUD: [Create, Read, Update, Delete]
349     /**
350      * A function to convert the integer format for Boolean to actual Boolean
           format.
351      * Input of null will output a null.
352      * @param bool input value to test.
353      * @return if not Null, return true if bool is 1, else return false.s
354      */
355     private Boolean NullBooleanFieldHelper(Integer bool)
356     {
357         if(bool == null)
358             return null;
359         if(bool.equals(1))
360             return Boolean.TRUE;
361         return Boolean.FALSE;
362     }
```

```
363     // brKeyboard objects
364     // Create
365     public long add_brKeyboard(brKeyboard brk)
366     {
367         SQLiteDatabase db = this.getWritableDatabase();
368         long rval = add_brKeyboard(db, brk);
369         db.close();
370         return(rval);
371     }
373     public long add_brKeyboard(SQLiteDatabase db, brKeyboard brk)
374     {
375         if(db==null||brk==null||db.isReadOnly())
376         {
377             // TODO: Consider if we need to throw an exception or allocate the
                    db ourselves.
378             return(-1);
379         }
380         ContentValues values = getContentValues(brk);
381         db.beginTransaction();
382         long rval = db.insert(BlueReadyDB.BRKeyboard.BR_TABLE_NAME, null, values
                );
383         db.setTransactionSuccessful();
384         db.endTransaction();
385         return(rval);
386     }
387     /**
388      * @param brk
389      * @return
390      */
391     private ContentValues getContentValues(brKeyboard brk) {
392         ContentValues values = new ContentValues();
393         values.put(BlueReadyDB.BRKeyboard.BR_KB_ADDRESS,      brk.get_address());
394         values.put(BlueReadyDB.BRKeyboard.BR_KB_NAME,         brk.get_name());
395         values.put(BlueReadyDB.BRKeyboard.BR_KB_CREATED,      brk.get_ts_created()
                );
396         values.put(BlueReadyDB.BRKeyboard.BR_KB_LAST_SEEN,    brk.get_ts_last_seen
                ());
397         values.put(BlueReadyDB.BRKeyboard.BR_KB_LAST_QUERIED, brk.
                get_ts_last_queried());
398         values.put(BlueReadyDB.BRKeyboard.BR_KB_IS_SDP_DONE,brk.get_is_sdp_done
                ());
399         values.put(BlueReadyDB.BRKeyboard.BR_KB_IS_ON_AIR,    brk.get_is_on_air());
400         values.put(BlueReadyDB.BRKeyboard.BR_KB_IS_HID,       brk.get_is_hid());
401         values.put(BlueReadyDB.BRKeyboard.BR_KB_IS_PAIRED,    brk.get_is_paired());
402         values.put(BlueReadyDB.BRKeyboard.BR_KB_IS_PREFERED,brk.get_is_prefered
                ());
403         values.put(BlueReadyDB.BRKeyboard.BR_KB_CONNECT_COUNT,brk.get_connects()
                );
404         return values;
405     }
406     // Read
407     public brKeyboard get_brKeyboard(long _ID)
408     {
409         SQLiteDatabase db = this.getReadableDatabase();
410         brKeyboard value = get_brKeyboard(db, _ID);
411         db.close();
412         return(value);
413     }
414     public brKeyboard get_brKeyboard(SQLiteDatabase db, long _ID)
415     {
416         if(db==null)
417             return null;
418         Cursor c = db.rawQuery("select * from " +
419                 BlueReadyDB.BRKeyboard.BR_TABLE_NAME +
420                 " where " + BlueReadyDB.BRKeyboard._ID + "=?",
```

```
421                 new String[] {String.valueOf(_ID)});
422         if(c==null)
423             return null;
424         c.moveToFirst();
425         brKeyboard k = new brKeyboard(
426                 c.getInt(c.getColumnIndex(BlueReadyDB.BRKeyboard._ID)),//_ID,
427                 c.getString(c.getColumnIndex(BlueReadyDB.BRKeyboard.BR_KB_NAME))
                    ,//name,
428                 c.getString(c.getColumnIndex(BlueReadyDB.BRKeyboard.
                    BR_KB_ADDRESS)),//address, 0
429                 c.getString(c.getColumnIndex(BlueReadyDB.BRKeyboard.
                    BR_KB_CREATED)),//ts_created,
430                 c.getString(c.getColumnIndex(BlueReadyDB.BRKeyboard.
                    BR_KB_LAST_SEEN)),//ts_last_seen,
431                 c.getString(c.getColumnIndex(BlueReadyDB.BRKeyboard.
                    BR_KB_LAST_QUERIED)),//ts_last_queried,
432                 NullBooleanFieldHelper(c.getInt(c.getColumnIndex(BlueReadyDB.
                    BRKeyboard.BR_KB_IS_SDP_DONE))),//is_sdp_done,
433                 NullBooleanFieldHelper(c.getInt(c.getColumnIndex(BlueReadyDB.
                    BRKeyboard.BR_KB_IS_ON_AIR))),//is_on_air,
434                 NullBooleanFieldHelper(c.getInt(c.getColumnIndex(BlueReadyDB.
                    BRKeyboard.BR_KB_IS_HID))),//is_hid,
435                 NullBooleanFieldHelper(c.getInt(c.getColumnIndex(BlueReadyDB.
                    BRKeyboard.BR_KB_IS_PAIRED))),//is_paired,
436                 NullBooleanFieldHelper(c.getInt(c.getColumnIndex(BlueReadyDB.
                    BRKeyboard.BR_KB_IS_PREFERED))),
437                 c.getInt(c.getColumnIndex(BlueReadyDB.BRKeyboard.
                    BR_KB_CONNECT_COUNT))//connects);
438         );
439         c.close();
440         return k;
441     }
442     public List<brKeyboard> getAll_brKeyboards()
443     {
444         return getAll_brKeyboards(null);
445     }
446     public List<brKeyboard> getAll_brKeyboards(String where)
447     {
448         SQLiteDatabase db = this.getReadableDatabase();
449         return getAll_brKeyboards(db, where);
450     }
451     public List<brKeyboard> getAll_brKeyboards(SQLiteDatabase db, String where)
452     {
453         if(db == null)
454         {
455             return null;
456         }
457         final int capacity = get_brKeyboardCount(db, where);
458         String query = "select " + BlueReadyDB.BRKeyboard._ID + " from " +
               BlueReadyDB.BRKeyboard.BR_TABLE_NAME;
459         if(where!=null && where.length()>0)
460         {
461             query += " where " + where;
462         }
463         query += "order by " + BlueReadyDB.BRKeyboard.DEFAULT_SORT_ORDER;
465         List<brKeyboard> brKeyboardList = new ArrayList<brKeyboard>(capacity);
467         Cursor c = db.rawQuery(query, null);
468         if(c == null)
469         {
470             db.close();
471             return(null);
472         }
473         c.moveToFirst();
474         for(int i = 0; i<c.getCount();i++)
```

```
475         {
476             brKeyboardList.add(get_brKeyboard(db, c.getInt(c.getColumnIndex(
                    BlueReadyDB.BRKeyboard._ID))));
477             c.moveToNext();
478         }
479         return brKeyboardList;
480     }
481     public int get_brKeyboardCount()
482     {
483         return get_brKeyboardCount(null);
484     }
485     /**
486      *
487      * @param where is a string containint any where conditions to be applied,
488      * don't include the where keyword,
489      * they are blindly appended to the end of the query after the where keyword.
490      * @return The count of records matching the optional where condition.
491      */
492     public int get_brKeyboardCount(String where)
493     {
494         SQLiteDatabase db = this.getReadableDatabase();
495         int value = get_brKeyboardCount(db, where);
496         db.close();
497         return value;
498     }
499     public int get_brKeyboardCount(SQLiteDatabase db, String where)
500     {
501         String countQuery = "select count(*) as count from " + BlueReadyDB.
                BRKeyboard.BR_TABLE_NAME;
502         if(where!=null && where.length()>0)
503         {
504             countQuery += " where " + where;
505         }
506         if(db == null)
507             return 0;
508         Cursor c = db.rawQuery(countQuery, null);
509         if(c==null)
510             return 0;
511         c.moveToFirst();
512         int Count = c.getInt(c.getColumnIndex("count"));
513         c.close();
514         return Count;
515     }
516     // Update
517     public int update_brKeyboard(brKeyboard brk)
518     {
519         SQLiteDatabase db = this.getWritableDatabase();
520         int rval = update_brKeyboard(db, brk);
521         db.close();
522         return rval;
523     }
524     public int update_brKeyboard(SQLiteDatabase db, brKeyboard brk)
525     {
526         if(db==null||brk==null||db.isReadOnly()||brk.get_ID()==null)
527         {
528             return 0;
529         }
530         ContentValues values = getContentValues(brk);
531         db.beginTransaction();
532         int rval = db.update(BlueReadyDB.BRKeyboard.BR_TABLE_NAME, values,
                BlueReadyDB.BRKeyboard._ID + "=?", new String[]{String.valueOf(brk.
                get_ID())});
533         db.setTransactionSuccessful();
534         db.endTransaction();
535         return rval;
536     }
537     // Delete
```

```
538     public void delete_brKeyboard(brKeyboard brk)
539     {
540         SQLiteDatabase db = this.getWritableDatabase();
541         delete_brKeyboards(db, brk);
542         db.close();
543     }
544     public void delete_brKeyboards(SQLiteDatabase db, brKeyboard brk) {
545         if(db==null||db.isReadOnly()||brk==null||brk.get_ID()==null)
546         {
547             return;
548         }
549         db.beginTransaction();
550         db.delete(BlueReadyDB.BRKeyboard.BR_TABLE_NAME, BlueReadyDB.BRKeyboard.
                _ID, new String[] {String.valueOf(brk.get_ID())});
551         db.setTransactionSuccessful();
552         db.endTransaction();
553     }
555 }
```

C.5 The `ConnectActivity.java` Class

This class provides the facility to enter the the RADIO_ON state, initiate SEARCHING and CONNECTING states. It also enables the user to select the desired Bluetooth device for the connection attempt.

Listing C.5: Bluetooth control and Device Selection Activity

```
1  /*
2   * Always Blue Ready
3   * Copyright (C) 2012
4   * By Fred Houweling
5   * fred@houweling.com.au
6   * All Rights Reserved
7   * Version 1.01 - 14 Sep 2012
8   *
9   * Description:
10  * This class enumeration of devices and provides the button to trigger
        transition to connecting state.
11  *
12  * Revision:
13  * Date          Version  Who         Comments
14  * ============  =======  ==========  ========================================
15  * 14 Sep 2012     1.01   Fred        Cleanup.
16  * 11 Sep 2012     1.00   Fred        Original version. (Suspect on date)
17  */
18 package au.com.houweling.BlueReadyProof;
20 import android.app.Activity;
21 import android.content.pm.PackageInfo;
22 import android.content.pm.PackageManager;
23 import android.database.Cursor;
24 import android.database.sqlite.SQLiteDatabase;
25 import android.database.sqlite.SQLiteQueryBuilder;
26 import android.os.Bundle;
```

```
27 import android.os.Message;
28 import android.util.Log;
29 import android.view.View;
30 import android.view.View.OnClickListener;
31 import android.widget.AdapterView;
32 import android.widget.AdapterView.OnItemSelectedListener;
33 import android.widget.BaseAdapter;
34 import android.widget.Button;
35 import android.widget.CheckedTextView;
36 import android.widget.RadioGroup;
37 import android.widget.TextView;
38 import android.widget.RadioGroup.OnCheckedChangeListener;
39 import android.widget.SimpleCursorAdapter;
40 import android.widget.Spinner;
41 import android.widget.Toast;
42 import au.com.houweling.BlueReadyProof.BlueReadyDB.BRKeyboard;
43 import au.com.houweling.BlueReadyProof.BlueReadyStateBasedMachine.BLUESTATES;

45 public class ConnectActivity extends Activity {
46     // private static final String DEBUG_TAG = "BlueReady";
47     public static String ignoreNull(String in) {
48         if (in == null)
49             return "";
50         return in;
51     }

53     private static CharSequence statusMessageText = null;
54     private static CheckedTextView statusMessage = null;
55     private static Button scanButton = null;
56     private static Button connectButton = null;

58     // public static final String PREFERENCES_FILE = "TestTab";

60     public static void setButtonState(boolean State) {
61         if (scanButton == null||connectButton == null) {
62             return;
63         }
64         if (State) {
65             scanButton.setVisibility(View.VISIBLE);
66             scanButton.setEnabled(true);
67             connectButton.setVisibility(View.VISIBLE);
68             connectButton.setEnabled(true);
69         } else {
70             scanButton.setEnabled(false);
71             scanButton.setVisibility(View.GONE);
72             connectButton.setEnabled(false);
73             connectButton.setVisibility(View.GONE);
74         }
75     }

77     public static void setText(CharSequence text) {
78         if (statusMessage != null) { // Check if tv is not null
79             if (text != null) {
80                 statusMessage.setText(text);
81             } else if (statusMessageText != null) {
82                 statusMessage.setText(statusMessageText);
83                 statusMessageText = null;
84             }

86         } else { // Class instance onCreate not yet called, cache the text for
87                  // later.
88             statusMessageText = text;
89         }
90     }

92     public static void requery() {
93         if (brCursor != null) {
94             brCursor.requery();
95             if (adapter != null) {
96                 ((BaseAdapter) adapter).notifyDataSetChanged();
```

```
97              }
99          }
100     }
102     private static Cursor brCursor;
104     // Needed to query the database.
105     private BlueReadyDBHelper brDatabase;
106     private SQLiteDatabase BrDb;
107     protected static SimpleCursorAdapter adapter;
108     private Spinner deviceSpinner;
110     void fillSpinner() {
112         // Enable the database access
113         brDatabase = new BlueReadyDBHelper(this);
114         BrDb = brDatabase.getReadableDatabase();
116         // Populate the Keyboard Dropdown Spinner
117         SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
118         queryBuilder.setTables(BRKeyboard.BR_TABLE_NAME);
119         queryBuilder.appendWhere(BRKeyboard.BR_TABLE_NAME + "."
120                 + BRKeyboard.BR_KB_IS_HID + " != 0");
121         String returnedColumns[] = {
122                 BRKeyboard.BR_TABLE_NAME + "." + BRKeyboard._ID,
123                 BRKeyboard.BR_TABLE_NAME + "." + BRKeyboard.BR_KB_NAME,
124                 BRKeyboard.BR_TABLE_NAME + "." + BRKeyboard.BR_KB_ADDRESS,
125                 BRKeyboard.BR_TABLE_NAME + "." + BRKeyboard.BR_KB_IS_ON_AIR,
126                 BRKeyboard.BR_TABLE_NAME + "." + BRKeyboard.BR_KB_IS_PREFERED
128         };
130         // TODO: Fix this:- Database connection not yet available.
131         brCursor = queryBuilder.query(BrDb, returnedColumns, null, null, null,
132                 null, BRKeyboard.DEFAULT_SORT_ORDER);
133         // startManagingCursor(brCursor);
134         brCursor.moveToFirst();
135         startManagingCursor(brCursor);
137         deviceSpinner = (Spinner) findViewById(R.id.selectDevice);
138         OnItemSelectedListener spinnerListener = new myOnItemSelectedListener(
139                 this, adapter);
140         deviceSpinner.setOnItemSelectedListener(spinnerListener);
142         // MyCursorAdapter mca = new MyCursorAdapter(this.getParent(), brCursor,
143         // deviceSpinner);
144         // myarray = ArrayAdapter();
145         adapter = new SimpleCursorAdapter(this,
146         // android.R.layout.simple_spinner_dropdown_item,
147                 R.layout.br_keyboard_item, brCursor, new String[] {
148                         BRKeyboard._ID, BRKeyboard.BR_KB_NAME,
149                         BRKeyboard.BR_KB_ADDRESS // ,
150                 // BRKeyboard.BR_KB_IS_ON_AIR//,
151                 // BRKeyboard.BR_KB_IS_PREFERED
152                 }, new int[] { R.id.text1, R.id.deviceName, R.id.deviceAddress
                    // ,
153                 // R.id.onAirIndicator
154                 }); // , R.id.Selected });
155         deviceSpinner.setAdapter(adapter);
157     }
159     public class myOnItemSelectedListener implements OnItemSelectedListener {
161         /*
162          * provide local instances of the mLocalAdapter and the mLocalContext
163          */
165         SimpleCursorAdapter mLocalAdapter;
166         Activity mLocalContext;
```

```
168         /**
169          * Constructor
170          *
171          * @param c
172          *            - The activity that displays the Spinner.
173          * @param ad
174          *            - The Adapter view that controls the Spinner. Instantiate
175          *            a new listener object.
176          */
177         public myOnItemSelectedListener(Activity c, SimpleCursorAdapter ad) {
179             this.mLocalContext = c;
180             this.mLocalAdapter = ad;
182         }
184         /**
185          * When the user selects an item in the spinner, this method is invoked
186          * by the callback chain. Android calls the item selected listener for
187          * the spinner, which invokes the onItemSelected method.
188          *
189          * @see android.widget.AdapterView.OnItemSelectedListener#onItemSelected
                (android.widget.AdapterView,
190          *      android.view.View, int, long)
191          * @param parent
192          *            - the AdapterView for this listener
193          * @param v
194          *            - the View for this listener
195          * @param pos
196          *            - the 0-based position of the selection in the
197          *            mLocalAdapter
198          * @param row
199          *            - the 0-based row number of the selection in the View
200          */
201         public void onItemSelected(AdapterView<?> parent, View v, int pos,
202                 long row) {
204             /*
205              * Set the value of the text field in the UI
206              */
207             BlueReadyDBHelper db = new BlueReadyDBHelper(getBaseContext());
209             brKeyboard brk = db.get_brKeyboard(row);
210             TestingActivity.setSelectedDevice(brk.get_address(), brk.get_name());
211             // TODO: Finish this.
212             //TestingActivity.setSelectedDevice(address, name);
213             Toast.makeText(getApplicationContext(),
214                     "You selected: " + brk.get_name(), Toast.LENGTH_SHORT).show
                        ();
215         }
217         /**
218          * The definition of OnItemSelectedListener requires an override of
219          * onNothingSelected(), even though this implementation does not use it.
220          *
221          * @param parent
222          *            - The View for this Listener
223          */
224         public void onNothingSelected(AdapterView<?> parent) {
226             // do nothing
228         }
229     }
231     @Override
232     protected void onStart() {
233         fillSpinner();
234         super.onStart();
235     }
237     @Override
```

```
238     protected void onStop() {
239         brCursor.close();
240         BrDb.close();
241         brDatabase.close();
242         super.onStop();
243     }
245     @Override
246     protected void onDestroy() {
247         super.onDestroy();
248         // Close the cursor, database and helper.
249         if (brCursor != null) {
250             brCursor.close();
251             brCursor = null;
252         }
253         if (BrDb != null) {
254             BrDb.close();
255             BrDb = null;
256         }
257         if (brDatabase != null) {
258             brDatabase.close();
259             brDatabase = null;
260         }
261         // Clear the value of tv.
262         statusMessage = null;
263     }
264     // source: http://ballardhack.wordpress.com/2010/09/28/subversion-revision-
        in-android-app-version-with-eclipse/
265     private String getVersionName() {
266             String version = "??";
267             try {
268                 PackageInfo pi = getPackageManager().getPackageInfo(
                        getPackageName(), 0);
269                 version = pi.versionName;
270             } catch (PackageManager.NameNotFoundException e) {
271                 Log.e("abr.version", "Version name not found in package", e);
272             }
273             return version;
274         }
276         private int getVersionCode() {
277             int version = -1;
278             try {
279                 PackageInfo pi = getPackageManager().getPackageInfo(
                        getPackageName(), 0);
280                 version = pi.versionCode;
281             } catch (PackageManager.NameNotFoundException e) {
282                 Log.e("abr.version", "Version number not found in package", e);
283             }
284             return version;
285         }
287     @Override
288     public void onCreate(Bundle savedInstanceState) {
289         super.onCreate(savedInstanceState);
291         setContentView(R.layout.connection_tab);
292         // Store the object of the activity tabs statusMessage field so other
293         // classes can use it.
294         TextView ver = (TextView) findViewById(R.id.version);
295         if(ver!=null)
296         {
297             ver.setText("Version: "+getVersionName()+"["+getVersionCode()+"]");
298         }
299         statusMessage = (CheckedTextView) findViewById(R.id.statusMessages);
300         connectButton = (Button) findViewById(R.id.connectButton);
301         scanButton = (Button) findViewById(R.id.startScan);
302         scanButton.setOnClickListener(new OnClickListener() {
```

```
304             public void onClick(View v) {
305                 BlueReadyProofActivity.bsbm
306                         .setTargetState(BLUESTATES.SEARCHING);
307             }
308         });
309         connectButton.setOnClickListener(new OnClickListener() {
311             public void onClick(View v) {
312                 if(BlueReadyProofActivity.handler!=null)
313                 {
314                     Message msg = BlueReadyProofActivity.handler.obtainMessage();
315                     if(msg!=null)
316                     {
317                         msg.what = 2000;
318                         msg.obj = "test";
319                         msg.arg1 = BLUESTATES.CONNECTING.ordinal();
320                         BlueReadyProofActivity.handler.dispatchMessage(msg);
321                     }
322                 }
323             }
324         });
325         // Ensure any cached text strings get displayed.
326         setText(null);
327         RadioGroup autoEnableGrp = (RadioGroup) findViewById(R.id.
               AutoEnableGroup);
328         autoEnableGrp.setOnCheckedChangeListener(new OnCheckedChangeListener() {
330             public void onCheckedChanged(RadioGroup group, int checkedId) {
331                 // RadioButton rbYes = (RadioButton)
332                 // findViewById(R.id.AutoEnableYes);
333                 // RadioButton rbNo = (RadioButton)
334                 // findViewById(R.id.AutoEnableNo);
335                 // if(rbYes.isChecked())
336                 if (checkedId == R.id.AutoEnableYes) {
337                     BlueReadyProofActivity.bsbm
338                             .setTargetState(BLUESTATES.RADIO_ON);
339                 }
340             }
341         });
342     }
343 }
```

C.6 The DummyActivity.java Class

This class is a template for additional tabs that will be generated in the future. It acts as a place-holder for the Tutorial and configuration tabs.

Listing C.6: Tab Template Class

```
1 /*
2  * Always Blue Ready
3  * Copyright (C) 2012
4  * By Fred Houweling
```

```
 * fred@houweling.com.au
 * All Rights Reserved
 * Version 1.01 - 14 Sep 2012
 *
 * Description:
 * This class is a dummy place holder for further tabs, consider it a template
    of sorts.
 *
 * Revision:
 * Date          Version  Who         Comments
 * ============  =======  ==========  ==========================================
 * 14 Sep 2012     1.01   Fred        Cleanup.
 * 21 Mar 2012     1.00   Fred        Original version.
 */
package au.com.houweling.BlueReadyProof;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class DummyActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView myTextView = new TextView(this);
        myTextView.setText("This is my Dummy Tab!.");
        setContentView(myTextView);
    }
}
```

C.7 The `BlueReadyDB.java` Class

This class is designed to physically name the tables and fields in the database, this is to reduce the risk of typographical errors causing application errors at runtime, instead forcing typographical errors to occur at compile time.

Listing C.7: SQL Database Class

```
/*
 * Always Blue Ready
 * Copyright (C) 2012
 * By Fred Houweling
 * fred@houweling.com.au
 * All Rights Reserved
 * Version 1.02 - 14 Sep 2012
 *
 * Description:
 * This class is designed to physically name the tables and fields in the
    database.
 *
 * Revision:
 * Date          Version  Who         Comments
 * ============  =======  ==========  ==========================================
 * 14 Sep 2012     1.02   Fred        Cleanup.
 * 13 Sep 2012     1.02   Fred        Added uuid related tables and a field to track
    connect counts.
 * 26 Jul 2012     1.00   Fred        Original version.
 */
```

```
19 package au.com.houweling.BlueReadyProof;

21 import android.provider.BaseColumns;

23 public final class BlueReadyDB {

25     public BlueReadyDB() {

27     }

29     public static final class BRKeyboard implements BaseColumns {
30         private BRKeyboard() {
31         }

33         // Table Name
34         public static final String BR_TABLE_NAME = "tb_br_keyboard";
35         // Data Columns
36         public static final String BR_KB_NAME = "name";
37         public static final String BR_KB_ADDRESS = "address";
38         public static final String BR_KB_CREATED = "ts_created";
39         public static final String BR_KB_LAST_SEEN = "ts_last_seen";
40         public static final String BR_KB_LAST_QUERIED = "ts_last_queried";
41         public static final String BR_KB_IS_SDP_DONE = "is_sdp_done";
42         public static final String BR_KB_IS_ON_AIR = "is_on_air";
43         public static final String BR_KB_IS_HID = "is_hid";
44         public static final String BR_KB_IS_PAIRED = "is_paired";
45         public static final String BR_KB_IS_PREFERED = "is_prefered";
46         public static final String BR_KB_CONNECT_COUNT = "connects";
47         // Default Sort
48         public static final String DEFAULT_SORT_ORDER = "is_on_air ASC, name ASC
               , address ASC";
49     }

51     public static final class BRuuid implements BaseColumns {
52         private BRuuid() {
53             // Empty
54         }

56         public static final String BR_TABLE_NAME = "tb_br_uuid";
57         public static final String BR_UUID = "uuid";
58         public static final String BR_CREATED = "ts_created";
59         public static final String BR_CONNECT_COUNT = "connects";
60         public static final String DEFAULT_SORT_ORDER = BR_UUID + " ASC";
61     }

63     public static final class BRkb_uuid_map implements BaseColumns {
64         private BRkb_uuid_map() {
65             // Empty
66         }

68         public static final String BR_TABLE_NAME = "tb_br_kb_uuid_map";
69         public static final String BR_CREATED = "ts_created";
70         public static final String BR_UUID_ID = "uuid_id";
71         public static final String BR_KB_ID = "kb_id";
72         public static final String DEFAULT_SORT_ORDER = _ID + " ASC";
73     }
74 }
```

C.8 The `bluereadyime.java` Class

This class currently acts as a stub for when development can progress to implementing an input method editor. It will extend the InputMethodService class.

Listing C.8: Input Method Editor

```
1  /*
2   * Always Blue Ready
3   * Copyright (C) 2012
4   * By Fred Houweling
5   * fred@houweling.com.au
6   * All Rights Reserved
7   * Version 1.01 - 12 Sep 2012
8   *
9   * Description:
10  * This class is a shell to provide a hint when starting to develop the IME code.
11  *
12  * Revision:
13  * Date          Version  Who         Comments
14  * ============  =======  ==========  ========================
15  * 12 Sep 2012     1.01   Fred        Cleanup.
16  * 31 Jul 2012     1.00   Fred        Original version.
17  */
18 package au.com.houweling.BlueReadyProof;
19
20 import android.inputmethodservice.InputMethodService;
21
22 public class bluereadyime extends InputMethodService {
23
24 }
```

C.9 The `brKeyboard.java` Class

This class provides the Java object version of the sqlite database record for the Bluetooth keyboard entries.

Listing C.9: Single Bluetooth keyboard object.

```
1  /*
2   * Always Blue Ready
3   * Copyright (C) 2012
4   * By Fred Houweling
5   * fred@houweling.com.au
6   * All Rights Reserved
7   * Version 1.01 - 14 Sep 2012
8   *
9   * Description:
10  * This class encapsulates the data stored in the SQLite database.
11  *
12  * Revision:
```

```
13  * Date          Version  Who          Comments
14  * ============  =======  ===========  ==============================
15  * 14 Sep 2012      1.01  Fred         Cleanup.
16  * 12 Sep 2012      1.00  Fred         Original version.
17  */
18 package au.com.houweling.BlueReadyProof;

21 public class brKeyboard {
22     /**
23      * @param _ID
24      * @param name
25      * @param address
26      * @param ts_created
27      * @param ts_last_seen
28      * @param ts_last_queried
29      * @param is_sdp_done
30      * @param is_on_air
31      * @param is_hid
32      * @param is_paired
33      * @param is_prefered
34      * @param connects
35      */
36     public brKeyboard(
37             Integer _ID,
38             String name,
39             String address,
40             String ts_created,
41             String ts_last_seen,
42             String ts_last_queried,
43             Boolean is_sdp_done,
44             Boolean is_on_air,
45             Boolean is_hid,
46             Boolean is_paired,
47             Boolean is_prefered,
48             Integer connects) {
49         this._ID = _ID;
50         this._name = name;
51         this._address = address;
52         this._ts_created = ts_created;
53         this._ts_last_seen = ts_last_seen;
54         this._ts_last_queried = ts_last_queried;
55         this._is_sdp_done = is_sdp_done;
56         this._is_on_air = is_on_air;
57         this._is_hid = is_hid;
58         this._is_paired = is_paired;
59         this._is_prefered = is_prefered;
60         this.set_connects(connects);
61     }
62     /**
63      * @param name
64      * @param address
65      * @param ts_created
66      * @param ts_last_seen
67      * @param ts_last_queried
68      * @param is_sdp_done
69      * @param is_on_air
70      * @param is_hid
71      * @param is_paired
72      * @param is_prefered
73      */
74     public brKeyboard(
75             String name,
76             String address,
77             String ts_created,
78             String ts_last_seen,
79             String ts_last_queried,
80             Boolean is_sdp_done,
81             Boolean is_on_air,
82             Boolean is_hid,
83             Boolean is_paired,
84             Boolean is_prefered,
```

```
85              Integer connects) {
86          this._name = name;
87          this._address = address;
88          this._ts_created = ts_created;
89          this._ts_last_seen = ts_last_seen;
90          this._ts_last_queried = ts_last_queried;
91          this._is_sdp_done = is_sdp_done;
92          this._is_on_air = is_on_air;
93          this._is_hid = is_hid;
94          this._is_paired = is_paired;
95          this._is_prefered = is_prefered;
96          this.set_connects(connects);
97      }

99      private Integer _ID;
100     private String _name;
101     private String _address;
102     private String _ts_created;
103     private String _ts_last_seen;
104     private String _ts_last_queried;
105     private Boolean _is_sdp_done;
106     private Boolean _is_on_air;
107     private Boolean _is_hid;
108     private Boolean _is_paired;
109     private Boolean _is_prefered;
110     private Integer _connects;

112     public brKeyboard() {
113         // Empty
114     }

116     public Integer get_ID() {
117         return _ID;
118     }

120     public void set_ID(Integer _ID) {
121         this._ID = _ID;
122     }

124     public String get_name() {
125         return _name;
126     }

128     public void set_name(String _name) {
129         this._name = _name;
130     }

132     public String get_address() {
133         return _address;
134     }

136     public void set_address(String _address) {
137         this._address = _address;
138     }

140     public String get_ts_created() {
141         return _ts_created;
142     }

144     public void set_ts_created(String _ts_created) {
145         this._ts_created = _ts_created;
146     }

148     public String get_ts_last_seen() {
149         return _ts_last_seen;
150     }

152     public void set_ts_last_seen(String _ts_last_seen) {
153         this._ts_last_seen = _ts_last_seen;
154     }

156     public String get_ts_last_queried() {
157         return _ts_last_queried;
```

```
158     }
160     public void set_ts_last_queried(String _ts_last_queried) {
161         this._ts_last_queried = _ts_last_queried;
162     }
164     public Boolean get_is_sdp_done() {
165         return _is_sdp_done;
166     }
168     public void set_is_sdp_done(Boolean _is_sdp_done) {
169         this._is_sdp_done = _is_sdp_done;
170     }
172     public Boolean get_is_on_air() {
173         return _is_on_air;
174     }
176     public void set_is_on_air(Boolean _is_on_air) {
177         this._is_on_air = _is_on_air;
178     }
180     public Boolean get_is_hid() {
181         return _is_hid;
182     }
184     public void set_is_hid(Boolean _is_hid) {
185         this._is_hid = _is_hid;
186     }
188     public Boolean get_is_paired() {
189         return _is_paired;
190     }
192     public void set_is_paired(Boolean _is_paired) {
193         this._is_paired = _is_paired;
194     }
196     public Boolean get_is_prefered() {
197         return _is_prefered;
198     }
200     public void set_is_prefered(Boolean _is_prefered) {
201         this._is_prefered = _is_prefered;
202     }
203     public Integer get_connects() {
204         return _connects;
205     }
206     public void set_connects(Integer _connects) {
207         this._connects = _connects;
208     }
209 }
```

C.10 The BlueReadyStateBasedMachine.java Class

This class implements the state based machine and manages state related events and transitions. It would benefit from a separation of pure state logic into a abstract class to be extended by this class.

Listing C.10: State Based Machine Implementation

```
/*
 * Always Blue Ready
 * Copyright (C) 2012
 * By Fred Houweling
 * fred@houweling.com.au
 * All Rights Reserved
 * Version 1.01 - 14 Sep 2012
 *
 * Description:
 * This implements the state based machine.
 *
 * Revision:
 * Date          Version  Who          Comments
 * ============  =======  ===========  ==============================
 * 14 Sep 2012     1.01   Fred         Cleanup.
 * 09 May 2012     1.00   Fred         Original version.
 */
package au.com.houweling.BlueReadyProof;

import java.io.IOException;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Message;
import android.util.Log;

/**
 * @author Fred
 *
 */
public class BlueReadyStateBasedMachine {
    enum BLUESTATES {
        INACTIVE, ENABLE_RADIO, RESTORE_RADIO, RADIO_ON, SEARCHING, PAIRING,
            CONNECTING, CONNECTED, STANDBY, DISCONNECTING, DISCONNECTED, DESTROY,
            NO_RADIO
    };

    BluetoothAdapter deviceBluetoothAdapter;
    // State variables to manage the threads various activities.
    private BLUESTATES CurrentState;
    private BLUESTATES TargetState;
    // A variable to track if the bluetooth radio was already on or not.
    private boolean BlueRadioEnabled;
    private boolean AutoEnable;
    private String address;

    class BLUETRANSITION {
        private BLUESTATES CurrentState;
        private List<BLUESTATES> TargetState;
        private BLUESTATES TransitionState;
        private boolean Enabled;
        private boolean Used;

        /**
         * @return the currentState
         */
        public BLUESTATES getCurrentState() {
            return CurrentState;
        }

        /**
         * @param currentState
         *            the currentState to set
```

```
71          */
72         public void setCurrentState(BLUESTATES currentState) {
73             CurrentState = currentState;
74         }
75
76         /**
77          * @return the targetState
78          */
79         public List<BLUESTATES> getTargetState() {
80             return TargetState;
81         }
82
83         /**
84          * @param targetState
85          *            the targetState to set
86          */
87         public void addTargetState(BLUESTATES targetState) {
88             TargetState.add(targetState);
89         }
90         /**
91          * @param targetState
92          *            the targetState to set
93          */
94         public void addTargetState(List<BLUESTATES> targetState) {
95             if(TargetState!=null)
96             {
97                 Iterator<BLUESTATES> it = targetState.iterator();
98                 while(it.hasNext())
99                 {
100                    this.addTargetState(it.next());
101                }
102            }
103        }
104
105        /**
106         * @return the transitionState
107         */
108        public BLUESTATES getTransitionState() {
109            return TransitionState;
110        }
111
112        /**
113         * @param transitionState
114         *            the transitionState to set
115         */
116        public void setTransitionState(BLUESTATES transitionState) {
117            TransitionState = transitionState;
118        }
119
120        /**
121         * @return the enabled
122         */
123        public boolean isEnabled() {
124            return Enabled;
125        }
126
127        /**
128         * @param enabled
129         *            the enabled to set
130         */
131        public void setEnabled(boolean enabled) {
132            Enabled = enabled;
133        }
134
135        /**
136         * @param used
137         *            the used to set
138         */
139        public void setUsed(boolean used) {
140            Used = used;
```

```
141         }
143         /**
144          * @return the used
145          */
146         public boolean isUsed() {
147             return Used;
148         }
150         /**
151          * @param used
152          *            the used to set
153          */
154         public BLUESTATES matchState(BLUESTATES CurrentState,
155                 BLUESTATES TargetState) {
156             if (this.CurrentState == CurrentState)
157             {
158                 Iterator<BLUESTATES> it = this.TargetState.iterator();
159                 while(it.hasNext())
160                 {
161                     if(it.next().compareTo(TargetState)==0)
162                     {
163                         Used = true;
164                         return this.TransitionState;
165                     }
166                 }
167             }
168             return (null);
169         }
171         private void resetUsed() {
172             Used = false;
173         }
175         public BLUETRANSITION(BLUESTATES CurrentState, ArrayList<BLUESTATES>
               TargetState,
176                                 BLUESTATES TransitionState, boolean Enabled) {
177             this.CurrentState = CurrentState;
178             this.TargetState = new ArrayList<BlueReadyStateBasedMachine.
                   BLUESTATES>();
179             this.addTargetState(TargetState);
180             this.TransitionState = TransitionState;
181             this.Enabled = Enabled;
182             resetUsed();
183         }
185         public BLUETRANSITION(BLUESTATES CurrentState, BLUESTATES TargetState,
186                 BLUESTATES TransitionState, boolean Enabled) {
187             this.CurrentState = CurrentState;
188             this.TargetState = new ArrayList<BlueReadyStateBasedMachine.
                   BLUESTATES>();
189             this.TargetState.add(TargetState);
190             this.TransitionState = TransitionState;
191             this.Enabled = Enabled;
192             resetUsed();
193         }
195     };
197     private BroadcastReceiver yourReceiver;
199     /**
200      * @return the yourReceiver
201      */
202     public BroadcastReceiver getYourReceiver() {
203         return yourReceiver;
204     }
206     /**
207      * @param yourReceiver
```

```
208         *            the yourReceiver to set
209         */
210        public void setYourReceiver(BroadcastReceiver yourReceiver) {
211            this.yourReceiver = yourReceiver;
212        }
214        /**
215         * @return the myiFilter
216         */
217        public IntentFilter getMyiFilter() {
218            return myiFilter;
219        }
221        /**
222         * @param myiFilter
223         *            the myiFilter to set
224         */
225        public void setMyiFilter(IntentFilter myiFilter) {
226            this.myiFilter = myiFilter;
227        }
229        static List<BLUETRANSITION> TransitionTable = new ArrayList<
               BlueReadyStateBasedMachine.BLUETRANSITION>(
230                90);
231        //static BLUETRANSITION radio_on = null;
232        private IntentFilter myiFilter;
233        private blueToothConnection mbc;
235        /**
236         *
237         */
238        public BlueReadyStateBasedMachine() {
240            //      The full state transition table, with redundancy reduced.
241            //      Code should lookup the Current State, once found, check that the
                        Target State is in the list, and if found, transition to the
                       transition state.
242            // This block of code is generated by the state table matrix stored in
                       the file: StateTable.xlsx
243            // is this state table changes, refresh the pivot, copy cell range at
                       roughly AL20:AL41 in Sheet3 after removing all double quote symbols.
244            // It is done this way to reduce risk of transcription errors.
246            // DONE: Finish the transition table.
247            ArrayList<BLUESTATES> targets = new ArrayList<BlueReadyStateBasedMachine
                       .BLUESTATES>();
249            // <editor-fold defaultstate="collapsed" desc="State Table">
250 //         targets.clear();
251 //         targets.add(BLUESTATES.INACTIVE);
252 //         TransitionTable.add(new BLUETRANSITION( BLUESTATES.INACTIVE, targets,
       BLUESTATES.INACTIVE, true));
253            targets.clear();
254            targets.add(BLUESTATES.INACTIVE);
255            targets.add(BLUESTATES.DESTROY);
256            targets.add(BLUESTATES.NO_RADIO);
257            targets.add(BLUESTATES.ENABLE_RADIO);
258            targets.add(BLUESTATES.RESTORE_RADIO);
259            TransitionTable.add(new BLUETRANSITION( BLUESTATES.RADIO_ON, targets,
                       BLUESTATES.RESTORE_RADIO, true));
260            targets.clear();
261            targets.add(BLUESTATES.INACTIVE);
262            targets.add(BLUESTATES.RADIO_ON);
263            targets.add(BLUESTATES.DESTROY);
264            targets.add(BLUESTATES.NO_RADIO);
265            targets.add(BLUESTATES.ENABLE_RADIO);
266            targets.add(BLUESTATES.RESTORE_RADIO);
267            TransitionTable.add(new BLUETRANSITION( BLUESTATES.SEARCHING, targets,
                       BLUESTATES.RADIO_ON, true));
```

```
268         targets.clear();
269         targets.add(BLUESTATES.INACTIVE);
270         targets.add(BLUESTATES.RADIO_ON);
271         targets.add(BLUESTATES.SEARCHING);
272         targets.add(BLUESTATES.DISCONNECTING);
273         targets.add(BLUESTATES.DISCONNECTED);
274         targets.add(BLUESTATES.DESTROY);
275         targets.add(BLUESTATES.NO_RADIO);
276         targets.add(BLUESTATES.ENABLE_RADIO);
277         targets.add(BLUESTATES.RESTORE_RADIO);
278         TransitionTable.add(new BLUETRANSITION( BLUESTATES.PAIRING, targets,
              BLUESTATES.RADIO_ON, true));
279         targets.clear();
280         targets.add(BLUESTATES.INACTIVE);
281         targets.add(BLUESTATES.RADIO_ON);
282         targets.add(BLUESTATES.SEARCHING);
283         targets.add(BLUESTATES.PAIRING);
284         targets.add(BLUESTATES.DESTROY);
285         targets.add(BLUESTATES.NO_RADIO);
286         targets.add(BLUESTATES.ENABLE_RADIO);
287         targets.add(BLUESTATES.RESTORE_RADIO);
288         TransitionTable.add(new BLUETRANSITION( BLUESTATES.CONNECTING, targets,
              BLUESTATES.RADIO_ON, true));
289         targets.clear();
290         targets.add(BLUESTATES.INACTIVE);
291         targets.add(BLUESTATES.RADIO_ON);
292         targets.add(BLUESTATES.SEARCHING);
293         targets.add(BLUESTATES.PAIRING);
294         targets.add(BLUESTATES.CONNECTING);
295         targets.add(BLUESTATES.DISCONNECTING);
296         targets.add(BLUESTATES.DISCONNECTED);
297         targets.add(BLUESTATES.DESTROY);
298         targets.add(BLUESTATES.NO_RADIO);
299         targets.add(BLUESTATES.ENABLE_RADIO);
300         targets.add(BLUESTATES.RESTORE_RADIO);
301         TransitionTable.add(new BLUETRANSITION( BLUESTATES.CONNECTED, targets,
              BLUESTATES.DISCONNECTING, true));
302         targets.clear();
303         targets.add(BLUESTATES.INACTIVE);
304         targets.add(BLUESTATES.RADIO_ON);
305         targets.add(BLUESTATES.SEARCHING);
306         targets.add(BLUESTATES.PAIRING);
307         targets.add(BLUESTATES.CONNECTING);
308         targets.add(BLUESTATES.CONNECTED);
309         targets.add(BLUESTATES.DISCONNECTED);
310         targets.add(BLUESTATES.DESTROY);
311         targets.add(BLUESTATES.NO_RADIO);
312         targets.add(BLUESTATES.STANDBY);
313         targets.add(BLUESTATES.ENABLE_RADIO);
314         targets.add(BLUESTATES.RESTORE_RADIO);
315         TransitionTable.add(new BLUETRANSITION( BLUESTATES.DISCONNECTING,
              targets, BLUESTATES.DISCONNECTED, true));
316         targets.clear();
317         targets.add(BLUESTATES.INACTIVE);
318         targets.add(BLUESTATES.RADIO_ON);
319         targets.add(BLUESTATES.SEARCHING);
320         targets.add(BLUESTATES.PAIRING);
321         targets.add(BLUESTATES.CONNECTING);
322         targets.add(BLUESTATES.CONNECTED);
323         targets.add(BLUESTATES.DISCONNECTING);
324         targets.add(BLUESTATES.DESTROY);
325         targets.add(BLUESTATES.NO_RADIO);
326         targets.add(BLUESTATES.STANDBY);
327         targets.add(BLUESTATES.ENABLE_RADIO);
328         targets.add(BLUESTATES.RESTORE_RADIO);
```

```
329         TransitionTable.add(new BLUETRANSITION( BLUESTATES.DISCONNECTED, targets
                , BLUESTATES.RADIO_ON, true));
330         targets.clear();
331         targets.add(BLUESTATES.INACTIVE);
332         targets.add(BLUESTATES.RADIO_ON);
333         targets.add(BLUESTATES.SEARCHING);
334         targets.add(BLUESTATES.PAIRING);
335         targets.add(BLUESTATES.CONNECTING);
336         targets.add(BLUESTATES.CONNECTED);
337         targets.add(BLUESTATES.DISCONNECTING);
338         targets.add(BLUESTATES.DISCONNECTED);
339         targets.add(BLUESTATES.DESTROY);
340         targets.add(BLUESTATES.NO_RADIO);
341         targets.add(BLUESTATES.ENABLE_RADIO);
342         targets.add(BLUESTATES.RESTORE_RADIO);
343         TransitionTable.add(new BLUETRANSITION( BLUESTATES.STANDBY, targets,
              BLUESTATES.CONNECTED, true));
344         targets.clear();
345         targets.add(BLUESTATES.INACTIVE);
346         targets.add(BLUESTATES.RADIO_ON);
347         targets.add(BLUESTATES.SEARCHING);
348         targets.add(BLUESTATES.PAIRING);
349         targets.add(BLUESTATES.CONNECTING);
350         targets.add(BLUESTATES.CONNECTED);
351         targets.add(BLUESTATES.DISCONNECTING);
352         targets.add(BLUESTATES.DISCONNECTED);
353         targets.add(BLUESTATES.DESTROY);
354         targets.add(BLUESTATES.NO_RADIO);
355         targets.add(BLUESTATES.STANDBY);
356         targets.add(BLUESTATES.RESTORE_RADIO);
357         TransitionTable.add(new BLUETRANSITION( BLUESTATES.ENABLE_RADIO, targets
                , BLUESTATES.RADIO_ON, true));
358         targets.clear();
359         targets.add(BLUESTATES.INACTIVE);
360         targets.add(BLUESTATES.RADIO_ON);
361         targets.add(BLUESTATES.SEARCHING);
362         targets.add(BLUESTATES.PAIRING);
363         targets.add(BLUESTATES.CONNECTING);
364         targets.add(BLUESTATES.CONNECTED);
365         targets.add(BLUESTATES.DISCONNECTING);
366         targets.add(BLUESTATES.DISCONNECTED);
367         targets.add(BLUESTATES.DESTROY);
368         targets.add(BLUESTATES.NO_RADIO);
369         targets.add(BLUESTATES.STANDBY);
370         targets.add(BLUESTATES.ENABLE_RADIO);
371         TransitionTable.add(new BLUETRANSITION( BLUESTATES.RESTORE_RADIO,
              targets, BLUESTATES.INACTIVE, true));
372         targets.clear();
373         targets.add(BLUESTATES.RADIO_ON);
374         targets.add(BLUESTATES.SEARCHING);
375         targets.add(BLUESTATES.PAIRING);
376         targets.add(BLUESTATES.CONNECTING);
377         targets.add(BLUESTATES.CONNECTED);
378         targets.add(BLUESTATES.DISCONNECTING);
379         targets.add(BLUESTATES.DISCONNECTED);
380         targets.add(BLUESTATES.STANDBY);
381         targets.add(BLUESTATES.ENABLE_RADIO);
382         targets.add(BLUESTATES.RESTORE_RADIO);
383         TransitionTable.add(new BLUETRANSITION( BLUESTATES.INACTIVE, targets,
              BLUESTATES.ENABLE_RADIO, true));
384 //      targets.clear();
385 //      targets.add(BLUESTATES.RADIO_ON);
386 //      TransitionTable.add(new BLUETRANSITION( BLUESTATES.RADIO_ON, targets,
    BLUESTATES.RADIO_ON, true));
387         targets.clear();
```

```
388             targets.add(BLUESTATES.SEARCHING);
389             targets.add(BLUESTATES.PAIRING);
390             TransitionTable.add(new BLUETRANSITION( BLUESTATES.RADIO_ON, targets,
                    BLUESTATES.SEARCHING, true));
391 //          targets.clear();
392 //          targets.add(BLUESTATES.SEARCHING);
393 //          TransitionTable.add(new BLUETRANSITION( BLUESTATES.SEARCHING, targets,
        BLUESTATES.SEARCHING, true));
394             targets.clear();
395             targets.add(BLUESTATES.PAIRING);
396             targets.add(BLUESTATES.CONNECTING);
397             targets.add(BLUESTATES.CONNECTED);
398             targets.add(BLUESTATES.DISCONNECTING);
399             targets.add(BLUESTATES.DISCONNECTED);
400             targets.add(BLUESTATES.STANDBY);
401             TransitionTable.add(new BLUETRANSITION( BLUESTATES.SEARCHING, targets,
                    BLUESTATES.PAIRING, true));
402 //          targets.clear();
403 //          targets.add(BLUESTATES.PAIRING);
404 //          TransitionTable.add(new BLUETRANSITION( BLUESTATES.PAIRING, targets,
        BLUESTATES.PAIRING, true));
405             targets.clear();
406             targets.add(BLUESTATES.CONNECTING);
407             targets.add(BLUESTATES.CONNECTED);
408             targets.add(BLUESTATES.DISCONNECTING);
409             targets.add(BLUESTATES.DISCONNECTED);
410             targets.add(BLUESTATES.STANDBY);
411             TransitionTable.add(new BLUETRANSITION( BLUESTATES.RADIO_ON, targets,
                    BLUESTATES.CONNECTING, true));
412             targets.clear();
413             targets.add(BLUESTATES.CONNECTING);
414             targets.add(BLUESTATES.CONNECTED);
415             targets.add(BLUESTATES.STANDBY);
416             TransitionTable.add(new BLUETRANSITION( BLUESTATES.PAIRING, targets,
                    BLUESTATES.CONNECTING, true));
417 //          targets.clear();
418 //          targets.add(BLUESTATES.CONNECTING);
419 //          TransitionTable.add(new BLUETRANSITION( BLUESTATES.CONNECTING, targets,
        BLUESTATES.CONNECTING, true));
420             targets.clear();
421             targets.add(BLUESTATES.CONNECTED);
422             targets.add(BLUESTATES.DISCONNECTING);
423             targets.add(BLUESTATES.DISCONNECTED);
424             targets.add(BLUESTATES.STANDBY);
425             TransitionTable.add(new BLUETRANSITION( BLUESTATES.CONNECTING, targets,
                    BLUESTATES.CONNECTED, true));
426 //          targets.clear();
427 //          targets.add(BLUESTATES.CONNECTED);
428 //          TransitionTable.add(new BLUETRANSITION( BLUESTATES.CONNECTED, targets,
        BLUESTATES.CONNECTED, true));
429             // </editor-fold>

431             setAutoEnable(false);
432             CurrentState = BLUESTATES.INACTIVE;
433             TargetState = BLUESTATES.INACTIVE;
434             myiFilter = new IntentFilter();
435             myiFilter.addAction(BluetoothDevice.ACTION_FOUND);
436             myiFilter.addAction(BluetoothAdapter.ACTION_STATE_CHANGED);
437             myiFilter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
438             this.yourReceiver = new BroadcastReceiver() {

440                 @Override
441                 public void onReceive(Context context, Intent intent) {
442                     Log.i("ABR.INTENT", intent.getAction());
443                     if (BluetoothDevice.ACTION_FOUND.contentEquals(intent
444                             .getAction())) {
```

```
445                     // Get the BluetoothDevice object from the Intent
446                     BluetoothDevice device = intent
447                             .getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
448                     // Insert record into database
449                     BlueReadyDBHelper.InsertOrUpdate(getContext(), null,
450                             device.getName(), device.getAddress(), true, null,
451                             null, true, null, null, null);
452                     dataChanged();
453                 } else if (BluetoothAdapter.ACTION_STATE_CHANGED
454                         .contentEquals(intent.getAction())) {
456                 } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED
457                         .contentEquals(intent.getAction())) {
458                     Message msg = BlueReadyProofActivity.handler
459                             .obtainMessage();
460                     msg.what = 4;
461                     msg.arg1 = BLUESTATES.RADIO_ON.ordinal();
462                     BlueReadyProofActivity.handler.dispatchMessage(msg);
463                     // Need to transition to RADIO_ON state.
464                 }
465             }
466         };
467     }
468     /**
469      *
470      * @param Current The current state
471      * @param Transition The transition state
472      * @param newState The flag that will now be set for this transition entry.
473      * @return true is an entry was identified, false if not, a false is a
          programming logic error.
474      */
475     private boolean setTransitionEnabledState(BLUESTATES Current,    BLUESTATES
        Transition , boolean newState)
476     {
477         if (TransitionTable != null)
478         {
479             Iterator<BLUETRANSITION> transitionItterator = TransitionTable
480                     .iterator();
481             while (transitionItterator.hasNext()) {
482                 BLUETRANSITION state = transitionItterator.next();
483                 if(state.getCurrentState().equals(Current)&&state.
                       getTransitionState().equals(Transition))
484                 {
485                     // FOUND The required state.
486                     state.setEnabled(newState);
487                     return true;
488                 }
489             }
490         }
491         // Should probably throw an exception, this is a programming logic bug,
              if reached.
492         return false;
493     }
494     /*
495      * Attempt to transition the current state.
496      *
497      * @return true if transition was not blocked, false if it was blocked.
498      */
499     private boolean transition() {
500         if (CurrentState == TargetState) {
501             return (true);
502         }
503         if (TransitionTable != null) {
504             boolean found = false;
505             Iterator<BLUETRANSITION> transitionItterator = TransitionTable
506                     .iterator();
507             while (transitionItterator.hasNext()) {
508                 BLUETRANSITION state = transitionItterator.next();
```

```
509                 BLUESTATES tState = state.matchState(this.CurrentState,
510                         this.TargetState);
511                 if (tState != null) {
512                     found = true;
513                     if (state.isEnabled() == true) {
514                         CurrentState = tState;
515                         return (true);
516                     }
517                 }
518             }
519             if (found == false) // Was the transition identified?
520             { // If not assume it is a direct transition
521                 CurrentState = TargetState;
522                 return (true);
523             }
524         }
525         return false;
526     }
528     private boolean Stale = true;
530     public void dataChanged() {
531         Stale = true;
532     }
534     public boolean isStale() {
535         if (Stale) {
536             Stale = false;
537             return (true);
538         }
539         return Stale;
540     }
542     private int retryLimit;
544     public int process() {
545         int returnValue = 0;
546         if (CurrentState != TargetState) {
547             BLUESTATES OldState = CurrentState;
548             boolean success = transition();
549             Log.d("ABR." + BlueReadyProofActivity.AppName, "Transition "
550                     + success + " from: " + OldState + " To: " + CurrentState);
551             if (success) {
552                 returnValue = 1;
553             }
554             // SingleBroadcastReceiver mBTReceiver;
555             switch (CurrentState) {
556             case ENABLE_RADIO:
557                 if (success) {
558                     setTransitionEnabledState(BLUESTATES.ENABLE_RADIO,
                           BLUESTATES.RADIO_ON, false);
559                     deviceBluetoothAdapter = BluetoothAdapter
560                             .getDefaultAdapter();
561                     if (deviceBluetoothAdapter == null) {
562                         setTargetState(BLUESTATES.NO_RADIO);
563                     } else {
564                         if (OldState == BLUESTATES.INACTIVE
565                                 && deviceBluetoothAdapter.isEnabled()) {
566                             setBlueRadioEnabled(true); // Store radio state
567                             // TargetState = BLUESTATES.RADIO_ON;
568                         } else {
569                             setBlueRadioEnabled(false); // Store radio state
570                             // Use of enable justified because we are providing
571                             // the user a critical user service.
572                             deviceBluetoothAdapter.enable(); // Permission
573                                                              // acquired by
574                                                              // user
575                                                              // selecting Yes
```

```
576                                                                 // in AutoEnable
577                                                                 // radio group.
578                         }
579                     }
580                 } else {
581                     if (deviceBluetoothAdapter.isEnabled()
582                             && deviceBluetoothAdapter.getState() ==
                                BluetoothAdapter.STATE_ON) {
583                         setTransitionEnabledState(BLUESTATES.ENABLE_RADIO,
                            BLUESTATES.RADIO_ON, true);
584                     }
585                 }
586                 break;
587             case RADIO_ON:
588                 // Block the radio on transition from enable radio.
589                 setTransitionEnabledState(BLUESTATES.ENABLE_RADIO, BLUESTATES.
                    RADIO_ON, false);
590                 if (OldState == BLUESTATES.SEARCHING) {
591                     deviceBluetoothAdapter.cancelDiscovery();
592                 }
593                 break;
594             case RESTORE_RADIO:
595                 if (isBlueRadioEnabled() == false) {
596                     deviceBluetoothAdapter.disable();
597                 }
598                 break;
599             case SEARCHING:
600                 if (deviceBluetoothAdapter.isDiscovering()) {
601                     deviceBluetoothAdapter.cancelDiscovery();
602                 }
603                 BluetoothDevice dev = null;
604                 blueToothConnection.startServiceDiscovery(dev);
605                 deviceBluetoothAdapter.startDiscovery();
606                 // mBTReceiver = new SingleBroadcastReceiver();
607                 // Enumerate bonded devices
608                 // ref:
609                 // http://developer.android.com/guide/topics/connectivity/
                    bluetooth.html
610                 Set<BluetoothDevice> pairedDevices = deviceBluetoothAdapter
611                         .getBondedDevices();
612                 // If there are paired devices
613                 if (pairedDevices.size() > 0) {
614                     // Loop through paired devices
615                     for (BluetoothDevice device : pairedDevices) {
616                         // Add the name and address to the database.
617                         BlueReadyDBHelper.InsertOrUpdate(getContext(), null,
618                                 device.getName(), device.getAddress(), false,
619                                 false, null, null, null, true, null);
620                         dataChanged();
621                     }
622                 }
623                 break;
624             case PAIRING:
625                 deviceBluetoothAdapter.cancelDiscovery();
626             case NO_RADIO:
627                 returnValue = -1;
628                 break;
629             case CONNECTING:
630                 if (success) {
631                     mbc = new blueToothConnection(
632                             deviceBluetoothAdapter
633                                     .getRemoteDevice(getAddress()));
634                     retryLimit = 5;
635                 }
636                 break;
637             case CONNECTED:
638                 break;
```

```
639             case DISCONNECTING:
640                 mbc.close();
641                 break;
642             case DISCONNECTED:
643                 mbc = null;
644                 break;
645             default:
646                 break;
647             }
648         } else { // Process any work that must occur whilst remaining in the
649                  // same state.
650             switch (CurrentState) {
651             case ENABLE_RADIO:
652                 if (deviceBluetoothAdapter.isEnabled()) {
653                     setTransitionEnabledState(BLUESTATES.ENABLE_RADIO,
                            BLUESTATES.RADIO_ON, true);
654                     TargetState = BLUESTATES.RADIO_ON;
655                 }
656                 break;
657             case CONNECTING:
658                 deviceBluetoothAdapter.cancelDiscovery();
659                 if (mbc.connect() == true)
660                 {
661                     TargetState = BLUESTATES.CONNECTED;
662                 }
663                 retryLimit--;
664                 if (retryLimit <= 0) {
665                     TargetState = BLUESTATES.RADIO_ON;
666                 }
667                 break;
668             case CONNECTED:
669                 byte[] buffer = new byte[8];
670                 // Please see p60-61 of the HID1_11.pdf standard.
671                 buffer[0] = (byte) (128 + 32 + 1);  // bmRequestType: Device to
                        Host + Class + Interface
672                 buffer[1] = 0x01;                   // bRequest: Get Report
673                 buffer[2] = 0x01;                   // wValue High: Input
674                 buffer[3] = 0;                      // wValue Low: Report ID (
                        Not Used)
675                 buffer[4] = 0;
676                 buffer[5] = 0;
677                 buffer[6] = 0;
678                 buffer[7] = 0;
679
680                 try {
681                     OutputStream out = mbc.get_oStream();
682                     for(int i = 0 ; i<8 ; i--)
683                         //out.write(buffer, 0, 8);
684                         out.write(buffer[i]);
685                     out.flush();
686                 } catch (IOException e1) {
687                     Log.e("abr.Error sending Get_Report request", e1.getMessage
                            ());
688                 }
689                 String buff = "";
690                 try {
691                     while (mbc.get_iStream().available() > 0) {
692                         int cval = mbc.get_iStream().read(); // Will block if no
693                                                              // data
694                         if (cval >= 0 || cval <= 255)
695                             buff += String.valueOf(cval);
696                     }
697                 } catch (IOException e) {
698                     e.printStackTrace();
699                 }
700                 if (buff.length() > 0) {
701                     Message msg = BlueReadyProofActivity.handler
```

```
702                             .obtainMessage();
703                     msg.what = 2;
704                     msg.obj = buff;
705                     BlueReadyProofActivity.handler.dispatchMessage(msg);
706                 }
707                 break;
708             case SEARCHING:
709                 // Monitor for new discoveries and pass them back to the UI
710                 // after a quick interrogation.
711                 break;
712             default:
713                 break;
715             }
716         }
717         /*
718          * Execute a single pass through the state based machine >=0 to keep
719          * looping.
720          */
721         return returnValue; // true == keep looping
722     }
724     public BLUESTATES getCurrentState() {
725         return CurrentState;
726     }
728     public void setCurrentState(BLUESTATES currentState) {
729         CurrentState = currentState;
730     }
732     public BLUESTATES getTargetState() {
733         return TargetState;
734     }
736     public void setTargetState(BLUESTATES targetState) {
737         TargetState = targetState;
738     }
740     /**
741      * @return the autoEnable
742      */
743     public boolean isAutoEnable() {
744         return AutoEnable;
745     }
747     private static Context s_context;
749     /**
750      * @return the context
751      */
752     public static Context getContext() {
753         return s_context;
754     }
756     /**
757      * @param context
758      *             the context to set
759      */
760     public static void setContext(Context context) {
761         s_context = context;
762     }
764     /**
765      * @param autoEnable
766      *             the autoEnable to set
767      */
768     public void setAutoEnable(boolean autoEnable) {
769         AutoEnable = autoEnable;
770     }
772     /**
773      * @return the blueRadioEnabled
```

```
774          */
775         public boolean isBlueRadioEnabled() {
776             return BlueRadioEnabled;
777         }
778
779         /**
780          * @param blueRadioEnabled
781          *            the blueRadioEnabled to set
782          */
783         private void setBlueRadioEnabled(boolean blueRadioEnabled) {
784             BlueRadioEnabled = blueRadioEnabled;
785         }
786
787         public static boolean compareIntToState(int state, BLUESTATES bs) {
788             if (BLUESTATES.values()[state] == bs)
789                 return (true);
790             return (false);
791         }
792
793         public static CharSequence getStatusString(int arg1) {
794
795             CharSequence message;
796             // DONE: Fix this to use string.xml
797             switch (BLUESTATES.values()[arg1]) {
798             case INACTIVE:
799                 message = getContext().getText(R.string.state_INACTIVE);
800                 break;
801             case RADIO_ON:
802                 message = getContext().getText(R.string.state_RADIO_ON);
803                 break;
804             case SEARCHING:
805                 message = getContext().getText(R.string.state_SEARCHING);
806                 break;
807             case PAIRING:
808                 message = getContext().getText(R.string.state_PAIRING);
809                 break;
810             case CONNECTING:
811                 message = getContext().getText(R.string.state_CONNECTING);
812                 break;
813             case CONNECTED:
814                 message = getContext().getText(R.string.state_CONNECTED);
815                 break;
816             case DISCONNECTING:
817                 message = getContext().getText(R.string.state_DISCONNECTING);
818                 break;
819             case DISCONNECTED:
820                 message = getContext().getText(R.string.state_DISCONNECTED);
821                 break;
822             case ENABLE_RADIO:
823                 message = getContext().getText(R.string.state_ENABLE_RADIO);
824                 break;
825             case RESTORE_RADIO:
826                 message = getContext().getText(R.string.state_RESTORE_RADIO);
827                 break;
828             case STANDBY:
829                 message = getContext().getText(R.string.state_STANDBY);
830                 break;
831             case NO_RADIO:
832                 message = getContext().getText(R.string.state_NO_RADIO);
833                 break;
834             case DESTROY:
835                 message = getContext().getText(R.string.state_DESTROY);
836                 break;
837             default:
838                 message = getContext().getText(R.string.state_INVALID);
839                     break;
840             }
841             return (message);
842         }
```

```
844     // These are called by the main activity, they are to allow the state based
845     // machine to make any last minute changes.
846     public void onPause() {
848     }
850     public void onResume() {
852     }
854     public void onStop() {
856     }
858     public void onDestroy() {
859         if (deviceBluetoothAdapter != null) {
860             if (isBlueRadioEnabled() == false) {
861                 CurrentState = BLUESTATES.INACTIVE;
862                 TargetState = BLUESTATES.INACTIVE;
863                 deviceBluetoothAdapter.disable();
864             }
865         }
866     }
868     public String getAddress() {
869         return address;
870     }
872     public void setAddress(String address) {
873         this.address = address;
874     }
876 }
```

C.11 The AndroidManifest.xml Application Manifest

This file provides the Android operation system with key information, such as what activities exist and any special permissions are required.

Listing C.11: Manifest Resource

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="au.com.houweling.BlueReadyProof"
4      android:versionCode="1"
5      android:versionName="1.0.533" >
7      <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="10" />
8      <uses-permission android:name="android.permission.BLUETOOTH" />
9      <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
11     <application
12         android:icon="@drawable/ic_launcher_brk"
13         android:label="@string/app_name" >
14         <activity android:name=".ConnectActivity" />
15         <activity android:name=".TestingActivity" />
16         <activity android:name=".DummyActivity" />
17         <activity
```

```
18              android:name=".BlueReadyProofActivity"
19              android:label="@string/app_name"
20              android:theme="@android:style/Theme.NoTitleBar" >
21              <intent-filter>
22                  <action android:name="android.intent.action.MAIN" />
24                  <category android:name="android.intent.category.LAUNCHER" />
25              </intent-filter>
26          </activity>
27      </application>
28  </manifest>
```

C.12 The `main.xml` Layout Resource

Main layout, defines the tabs but not much else.

Listing C.12: Application Master Layout

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <TabHost xmlns:android="http://schemas.android.com/apk/res/android"
3      android:id="@android:id/tabhost"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent" >
7      <LinearLayout
8          android:id="@+id/linearLayout1"
9          android:padding="5dp"
10         android:layout_width="fill_parent"
11         android:layout_height="fill_parent"
12         android:orientation="vertical">
13         <TabWidget
14             android:id="@android:id/tabs"
15             android:layout_width="fill_parent"
16             android:layout_height="wrap_content" />
17         <FrameLayout
18             android:id="@android:id/tabcontent"
19             android:layout_width="fill_parent"
20             android:layout_height="fill_parent"
21             android:padding="5dp" />
22     </LinearLayout>
23 </TabHost>
```

C.13 The `testing_tab.xml` Layout Resource

The layout for the connected state tab.

Listing C.13: Testing Tab Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/testingName"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/KeyboardName"
            android:textAppearance="?android:attr/textAppearanceLarge"
            android:textColor="@drawable/grey" />

        <TextView
            android:id="@+id/testingAddress"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/KeyboardAddress"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textColor="@drawable/red" />

        <TextView
            android:id="@+id/TestingStatus"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textColor="@drawable/yellow" />

        <Button
            android:id="@+id/disconnectButton"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/disconnect" />

        <TextView
            android:id="@+id/textView4"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/key_test_message"
            android:textAppearance="?android:attr/textAppearanceLarge" />

        <EditText
            android:id="@+id/keycapturelog"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:gravity="center_vertical|top"
            android:inputType="text|textMultiLine" >

            <requestFocus />
        </EditText>

</LinearLayout>
```

C.14 The `connection_tab.xml` Layout Resource

This resource defines the layout for the tab responsible to manage the state of the Bluetooth hardware and to enable the user to select the desired keyboard

Listing C.14: Bluetooth Selection Layout

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

        <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" >

            <CheckedTextView
                android:id="@+id/AutoEnableCaption"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/AutoBTEnable" />

            <RadioGroup
                android:id="@+id/AutoEnableGroup"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_gravity="fill_horizontal"
                android:orientation="horizontal" >

                <RadioButton
                    android:id="@+id/AutoEnableYes"
                    android:layout_width="wrap_content"
                    android:layout_height="fill_parent"
                    android:text="@string/Yes" />

                <RadioButton
                    android:id="@+id/AutoEnableNo"
                    android:layout_width="wrap_content"
                    android:layout_height="fill_parent"
                    android:checked="true"
                    android:text="@string/No" />
            </RadioGroup>
        </LinearLayout>

        <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" >

            <Button
                android:id="@+id/startScan"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:enabled="false"
                android:text="@string/scan_button"
                android:visibility="gone" />
        </LinearLayout>

        <CheckedTextView
            android:id="@+id/selectDeviceCaption"
```

```
59              android:layout_width="wrap_content"
60              android:layout_height="wrap_content"
61              android:text="@string/device_select" />

64          <Spinner
65              android:id="@+id/selectDevice"
66              android:layout_width="fill_parent"
67              android:layout_height="wrap_content"
68              android:drawSelectorOnTop="true"
69              android:prompt="@string/device_select"
70              tools:listitem="@layout/br_keyboard_item" >

72          </Spinner>

75          <Button
76              android:id="@+id/connectButton"
77              android:layout_width="fill_parent"
78              android:layout_height="wrap_content"
79              android:text="@string/connect" />

81          <CheckedTextView
82              android:id="@+id/statusMessages"
83              android:layout_width="fill_parent"
84              android:layout_height="fill_parent" />

86          <TextView
87              android:id="@+id/version"
88              android:layout_width="fill_parent"
89              android:layout_height="wrap_content"
90              android:text=""
91              android:textAppearance="?android:attr/textAppearanceSmall" />

93      </LinearLayout>

95  </ScrollView>
```

C.15 The br_keyboard_item.xml Layout Resource

This file defines the layout for each keyboard entry when it is displayed in the spinner control.

Listing C.15: Spinner Element Layout

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent" >

8      <TextView
9          android:id="@+id/deviceAddress"
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:layout_alignParentLeft="true"
13         android:layout_alignParentRight="true"
14         android:layout_alignParentTop="true"
15         android:text="@string/KeyboardAddress"
```

```
16          android:textAppearance="?android:attr/textAppearanceSmall"
17          android:textColor="@drawable/red" />

20      <TextView
21          android:id="@+id/deviceName"
22          android:layout_width="wrap_content"
23          android:layout_height="wrap_content"
24          android:layout_alignParentLeft="true"
25          android:layout_alignRight="@+id/deviceAddress"
26          android:layout_below="@+id/deviceAddress"
27          android:text="@string/KeyboardName"
28          android:textAppearance="?android:attr/textAppearanceMedium"
29          android:textColor="@drawable/grey" />

32      <TextView
33          android:id="@+id/text1"
34          android:layout_width="wrap_content"
35          android:layout_height="wrap_content"
36          android:layout_alignParentRight="true"
37          android:layout_alignParentTop="true"
38                  android:textAppearance="?android:attr/textAppearanceSmall" />

40  </RelativeLayout>
```

C.16 The `strings.xml` String Resource

This file contains textual display string, it is standard practice for Android development to externalise display strings into a similar file to enable localisation efforts.

Listing C.16: String Resource File

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>

5      <string name="app_name">Always Blue Ready</string>
6      <string name="bluetooth">Blue Tooth</string>
7      <string name="connect">Connect to Keyboard</string>
8      <string name="device_select">Select Bluetooth Keyboard</string>
9      <string name="test_tab">Test</string>
10     <string name="scan_button">Scan</string>
11     <string name="cancel_scan_button">Cancel Scan</string>
12     <string name="Yes">Yes</string>
13     <string name="No">No</string>
14     <string name="On">On</string>
15     <string name="Off">Off</string>
16     <string name="Null"></string>
17     <string name="AutoBTEnable">Auto Enable Bluetooth</string>
18     <string name="blueready_started">Blue Ready Started</string>
19     <string name="blueready_stopped">Blue Ready Stopped</string>
20     <string name="blueready_bound">Blue Ready Bound</string>
21     <string name="blueready_unbound">Blue Ready Unbound</string>
22     <string name="KeyboardName">Name</string>
23     <string name="KeyboardAddress">Address</string>
24     <string name="state.INVALID">Invalid State</string>
```

```
25      <string name="state.INACTIVE">INACTIVE</string>
26      <string name="state.RADIO_ON">RADIO_ON</string>
27      <string name="state.SEARCHING">SEARCHING</string>
28      <string name="state.PAIRING">PAIRING</string>
29      <string name="state.CONNECTING">CONNECTING</string>
30      <string name="state.CONNECTED">CONNECTED</string>
31      <string name="state.DISCONNECTING">DISCONNECTING</string>
32      <string name="state.DISCONNECTED">DISCONNECTED</string>
33      <string name="state.ENABLE_RADIO">ENABLE_RADIO</string>
34      <string name="state.RESTORE_RADIO">RESTORE_RADIO</string>
35      <string name="state.STANDBY">STANDBY</string>
36      <string name="state.NO_RADIO">NO_RADIO</string>
37      <string name="state.DESTROY">DESTROY</string>
38      <drawable name="white">#ffffff</drawable>
39      <drawable name="black">#000000</drawable>
40      <drawable name="green">#347C2C</drawable>
41      <drawable name="pink">#FF00FF</drawable>
42      <drawable name="violet">#a020f0</drawable>
43      <drawable name="grey">#778899</drawable>
44      <drawable name="red">#C11B17</drawable>
45      <drawable name="yellow">#FFFF8C</drawable>
46      <drawable name="PowderBlue">#b0e0e6</drawable>
47      <drawable name="brown">#2F1700</drawable>
48      <drawable name="Hotpink">#7D2252</drawable>
49      <drawable name="darkgrey">#606060</drawable>
50      <string name="disconnect">Disconnect</string>
51      <string name="key_test_message">Keyboard Input Test Area</string>
52  </resources>
```

Appendix D

Test Cases

The following tables provides individual test case details.

Table D.1: Unit Test Case TUN1 - State Transition

Test Name	State Transition	**Test ID**	TUN1
Type	Unit	**Object**	
Focus	Validation		
Purpose			
Validate State Transition Sequence			
Module / Class	BlueReadyStateBasedMachine.java		
Function	process		
Precondition			
In the Testing Tab, Switch app to test mode. To disable application logic.			
Test Steps			
For all possible target and current state combinations, test that the state transitions through the correct sequence as per state diagram.			
Test pass criteria			
States are transitioned to correctly without entering invalid states and no re-entry/loops are encountered.			
Outcome			
Not yet implemented			
Action			
Implement test harness.			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.2: Regression Test Case TRE2 - Spinner

Test Name	Spinner	**Test ID**	TRE2
Type	Regression	**Object**	
Focus	Validation		
Purpose			
Ensure spinner can drop down the list and allow single selection of an item.			
Module / Class	-		
Function	-		
Precondition			
Application database populated with values from scan/discovery, populating spinner.			
Test Steps			
Switch Auto enable Bluetooth radio button in connection tab to yes. Wait for RADIO_ON state. Press Scan button that appears in the Connection tab. Wait for it to reappear.			
Test pass criteria			
Select Bluetooth Keyboard spinner is populated with values.			
Outcome			
Pass			
Action			
SVN Ver #	514	**Test Date**	7/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.3: Functional Test Case TFU3 - Scanning

Test Name	Scanning	**Test ID**	TFU3
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure application can initiate Bluetooth scanning.			
Module / Class	-		
Function	-		
Precondition			
Application in RADIO_ON state, enable radio, wait for RADIO_ON transition. Bluetooth symbol in status area.			
Test Steps			
Press the scanning button.			
Test pass criteria			
Application enters Searching state, Spinner is populated with entries from previously Paired devices and any devices currently discoverable are added to the list. Returns to RADIO_ON state within 2 min.			
Outcome			
Pass			
Action			
SVN Ver #	514	**Test Date**	7/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.4: Functional Test Case TFU4 - Connecting

Test Name	Connecting	**Test ID**	TFU4
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure application can begin connecting to the selected Bluetooth device.			
Module / Class	-		
Function	-		
Precondition			
With application in RADIO_ON state, select device that is paired and turned on.			
Test Steps			
Press the Connect to Keyboard button.			
Test pass criteria			
Application switches to testing tab, changes application state to connecting and then connected.			
Outcome			
Pass			
Action			
SVN Ver #	514	**Test Date**	7/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.5: Functional Test Case TFU5 - Disconnecting

Test Name	Disconnecting	**Test ID**	TFU5
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure the application can disconnect from a connected Bluetooth device.			
Module / Class	-		
Function	-		
Precondition			
Successfully follow test TFU4 - Connecting.			
Test Steps			
Press the Disconnect button.			
Test pass criteria			
Application returns to the Connection tab, Application state changes to Disconnecting, Disconnected then RADIO_ON state.			
Outcome			
Fail			
Action			
Implement missing code to cause disconnection to operate.			
SVN Ver #	514	**Test Date**	7/12/2012
Pass/Fail	Fail	**Tester**	Fred

Table D.6: Functional Test Case TFU5 - Disconnecting

Test Name	Disconnecting	**Test ID**	TFU5
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure the application can disconnect from a connected Bluetooth device.			
Module / Class	-		
Function	-		
Precondition			
Successfully follow test TFU4 - Connecting.			
Test Steps			
Press the Disconnect button.			
Test pass criteria			
Application returns to the Connection tab, Application state changes to Disconnecting, Disconnected then RADIO_ON state.			
Outcome			
Pass			
Action			
SVN Ver #	533	**Test Date**	14/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.7: Functional Test Case TFU6 - Enable Radio

Test Name	Enable Radio	**Test ID**	TFU6
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure application can enable the Bluetooth radio.			
Module / Class	-		
Function	-		
Precondition			
Bluetooth turned off on device. Application running in INACTIVE state.			
Test Steps			
Switch Auto enable Bluetooth radio button in connection tab to yes. Wait for RADIO_ON state.			
Test pass criteria			
RADIO_ON state reached after or within 3 seconds of device Bluetooth symbol in status bar appearing. Max time 1 min.			
Outcome			
Action			
SVN Ver #	502	**Test Date**	6/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.8: Functional Test Case TFU7 - Radio Restore Off

Test Name	Radio Restore Off	**Test ID**	TFU7
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure the application restores the radio state correctly.			
Module / Class	-		
Function	-		
Precondition			
Start application with Bluetooth radio off.			
Test Steps			
Switch Auto enable Bluetooth radio button in connection tab to yes. Wait for RADIO_ON state. Exit application using the back button.			
Test pass criteria			
Application exits and device Bluetooth symbol automatically turns off/disappears indicating Bluetooth is off.			
Outcome			
Pass			
Action			
SVN Ver #	514	**Test Date**	7/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.9: Functional Test Case TFU8 - Radio Restore On

Test Name	Radio Restore On	**Test ID**	TFU8
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure application does not turn off the radio			
Module / Class	-		
Function	-		
Precondition			
Start application with Bluetooth radio on.			
Test Steps			
Switch Auto enable Bluetooth radio button in connection tab to yes. Wait for RADIO_ON state. Exit application using the back button.			
Test pass criteria			
Application exits and device Bluetooth symbol remaining on indicating Bluetooth is on.			
Outcome			
Pass			
Action			
SVN Ver #	514	**Test Date**	7/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.10: Unit Test Case TUN9 - Key Polling

Test Name	Key Polling	**Test ID**	TUN9
Type	Unit	**Object**	
Focus	Validation		
Purpose			
Ensure application can poll for key strokes.			
Module / Class	BlueReadyStateBasedMachine.java		
Function	process		
Precondition			
Connected to Bluetooth device.			
Test Steps			
Press keys on the connected Bluetooth device. In eclipse, debug application using break-points in the process function monitor execution in the CONNECTED state.			
Test pass criteria			
Key information is displayed by the application.			
Outcome			
logcat error: Transport endpoint is not connected			
Action			
Investigate cause of polling failure through debugging.			
SVN Ver #	514	**Test Date**	7/12/2012
Pass/Fail	Fail	**Tester**	Fred

Table D.11: Functional Test Case TFU10 - Tab Switching

Test Name	Tab Switching	**Test ID**	TFU10
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure application switches tabs without losing form data or application state.			
Module / Class	-		
Function	-		
Precondition			
Application running			
Test Steps			
Switch tabs, taking note of current values, input and selections. Switch tabs and return. Repeat 4 more time after altering form values.			
Test pass criteria			
Values, input and selections are retained.			
Outcome			
Action			
SVN Ver #	514	**Test Date**	7/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.12: System Test Case TSY11 - Data Persistence SQL

Test Name	Data Persistence SQL	**Test ID**	TSY11
Type	System	**Object**	
Focus	Validation		
Purpose			
Ensure application data within the SQL database survives device restart.			
Module / Class	-		
Function	-		
Precondition			
Application database populated with values from scan/discovery. (Spinner is populated)			
Test Steps			
View application spinner to confirm values, if none: cause them to be populated by starting a scan. Quit application, restart device, run application and confirm values are retained.			
Test pass criteria			
Database values are retained after a reboot. Spinner is still populated. May also inspect applications SQLite database to confirm no loss of data.			
Outcome			
Action			
SVN Ver #	514	**Test Date**	7/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.13: System Test Case TSY12 - Data Persistence Application State

Test Name	Data Persistence Application State	**Test ID**	TSY12
Type	System	**Object**	
Focus	Validation		
Purpose			
Ensure form data and application state retain settings and values during application suspension.			
Module / Class	-		
Function	-		
Precondition			
Application running, select random values.			
Test Steps			
Press home key, start external activity, (First simple: Phone Dialler, SMS, Calculator, Then more demanding, PDF viewer, Word document editor, 3D game etc.) Quit external app within 2 mins, return to application under test.			
Test pass criteria			
Form values, inputs and selections are retained.			
Outcome			
Action			
Implement code to save and restore state data.			
SVN Ver #	514	**Test Date**	7/12/2012
Pass/Fail	Fail	**Tester**	Fred

Table D.14: Unit Test Case TUN13 - Database Upgrade

Test Name	Database Upgrade	**Test ID**	TUN13
Type	Unit	**Object**	
Focus	Validation		
Purpose			
Ensure that the database is correctly upgraded during a upgrade			
Module / Class	BlueReadyDBHelper.java		
Function	onUpgrade		
Precondition			
Repeat once with all older publicly released versions, first time load old version to device, populate database through scanning for devices.			
Test Steps			
upgrade the software to the version under test (New, Current, Head, Latest etc.)			
Test pass criteria			
Using SQLite DB inspection software, Original values are retained after each upgrade. Database is structured correctly to current expectations without data loss.			
Outcome			
Action			
SVN Ver #	514	**Test Date**	7/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.15: Unit Test Case TUN14 - Database Creation

Test Name	Database Creation	**Test ID**	TUN14
Type	Unit	**Object**	
Focus	Validation		
Purpose			
Ensure that the SQLite database is created correctly during first time install.			
Module / Class	BlueReadyDBHelper.java		
Function	onCreate		
Precondition			
Remove old version of application, install version under test.			
Test Steps			
Open application under test.			
Test pass criteria			
Database tables are created without logcat errors. No force close.			
Outcome			
Action			
SVN Ver #	514	**Test Date**	7/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.16: Integration Test Case TIN15 - CRUD

Test Name	CRUD	**Test ID**	TIN15
Type	Integration	**Object**	
Focus	Validation		
Purpose			
Ensure that the record Create, Read, Update and Delete functions work correctly.			
Module / Class	BlueReadyDBHelper.java		
Function	-		
Precondition			
Run automated test battery.			
Test Steps			
Execute CRUD tests.			
Test pass criteria			
Inspect logcat output and application testing report.			
Outcome			
Action			
Implement test harness.			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.17: Regression Test Case TRE16 - Thread Creation

Test Name	Thread Creation	**Test ID**	TRE16
Type	Regression	**Object**	
Focus	Validation		
Purpose			
Validate that the thread for background processing is created correctly			
Module / Class	-		
Function	-		
Precondition			
nil			
Test Steps			
Code inspection, review logcat messages and debug monitoring			
Test pass criteria			
Background thread runs, no logcat messages or force closes related to long running operations on user interface (UI) thread.			
Outcome			
Pass			
Action			
SVN Ver #	514	**Test Date**	7/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.18: System Test Case TSY17 - Message Passing

Test Name	Message Passing	**Test ID**	TSY17
Type	System	**Object**	
Focus	Validation		
Purpose			
Ensure that message passing between UI and background threads are operating correctly.			
Module / Class	-		
Function	-		
Precondition			
nil			
Test Steps			
Code inspection, review logcat messages and debug monitoring			
Test pass criteria			
Messages are passed in a timely manner between threads, no logcat errors or force closes due to threads accessing other threads objects.			
Outcome			
Pass			
Action			
SVN Ver #	514	**Test Date**	7/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.19: Integration Test Case TIN18 - Intent Reception

Test Name	Intent Reception	**Test ID**	TIN18
Type	Integration	**Object**	yourReceive
Focus	Validation		
Purpose			
Ensure that the operating system Intent messages are received and processed.			
Module / Class	BlueReadyStateBasedMachine.java		
Function	onReceive		
Precondition			
nil			
Test Steps			
Code inspection, review logcat messages and debug monitoring			
Test pass criteria			
Breakpoint in function onReceive, debug live target, cause INTENT event and watch for intent reception. Alternative is to monitor logcat for ABR.INTENT messages.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.20: Functional Test Case TFU19 - App State Switch Exit

Test Name	App State Switch Exit	**Test ID**	TFU19
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure application can correctly transition states in the background when application exited with the back button.			
Module / Class	-		
Function	-		
Precondition			
Application running, radio off.			
Test Steps			
Enable radio, press back button, invoke other application, return to application under test.			
Test pass criteria			
Application displays state RADIO_ON			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.21: Functional Test Case TFU20 - App State Switch Home

Test Name	App State Switch Home	**Test ID**	TFU20
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure application can correctly transition states in the background when application exited with the home button.			
Module / Class	-		
Function	-		
Precondition			
Application running, radio off.			
Test Steps			
Enable radio, press home button, invoke other application, return to application under test.			
Test pass criteria			
Application displays state RADIO_ON			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.22: Functional Test Case TFU21 - Interface response to device rotation

Test Name	Interface response to device rotation	**Test ID**	TFU21
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure application form data and state is retained during device rotation.			
Module / Class	-		
Function	-		
Precondition			
Application running.			
Test Steps			
Rotate device to cause a orientation change from portrait to landscape and the inverse.			
Test pass criteria			
Application selected values and state are retained.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.23: Structural Test Case TST22 - Low task memory

Test Name	Low task memory	**Test ID**	TST22
Type	Structural	**Object**	
Focus	Validation		
Purpose			
Determine application behaviour when device task ram is low.			
Module / Class	-		
Function	-		
Precondition			
Application running.			
Test Steps			
Use adb shell and kill process with PID from ps or Kill it using DDMS, Restart application.			
Test pass criteria			
Application is restart able, not force close issues. Form data and state is retained.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.24: Structural Test Case TST23 - Low storage space

Test Name	Low storage space	**Test ID**	TST23
Type	Structural	**Object**	
Focus	Validation		
Purpose			
Determine application stability when data storage is running out.			
Module / Class	-		
Function	-		
Precondition			
Application running, free application storage space 0 bytes.			
Test Steps			
Attempt to use application, noting any unexpected behaviour. These will feed into new test cases and issues.			
Test pass criteria			
Application is graceful in it's execution, no force closes, understandable warning/error messages.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.25: Functional Test Case TFU24 - Interface portrait small form factor

Test Name	Interface portrait small form factor	**Test ID**	TFU24
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure interface is clear and unobstructed on a small form factor device in portrait orientation.			
Module / Class	-		
Function	-		
Precondition			
Application running in portrait orientation on a small form factor device (phone) less than 5 inch diagonal.			
Test Steps			
Review form elements, will need to alter application states to view different perspectives.			
Test pass criteria			
Interface is unobscured, icons are readable, no scrolling or panning required to view form elements and efficient use of space.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.26: Functional Test Case TFU25 - Interface landscape small form factor

Test Name	Interface landscape small form factor	**Test ID**	TFU25
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure interface is clear and unobstructed on a small form factor device in landscape orientation.			
Module / Class	-		
Function	-		
Precondition			
Application running in landscape orientation on a small form factor device (phone) less than 5 inch diagonal.			
Test Steps			
Review form elements, will need to alter application states to view different perspectives.			
Test pass criteria			
Interface is unobscured, icons are readable, no scrolling or panning required to view form elements and efficient use of space.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.27: Functional Test Case TFU26 - Interface portrait large form factor

Test Name	Interface portrait large form factor	**Test ID**	TFU26
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure interface is clear and unobstructed on a large form factor device in portrait orientation.			
Module / Class	-		
Function	-		
Precondition			
Application running in portrait orientation on a large form factor device (tablet) more than 7 inch diagonal.			
Test Steps			
Review form elements, will need to alter application states to view different perspectives.			
Test pass criteria			
Interface is unobscured, icons are readable, no scrolling or panning required to view form elements and efficient use of space.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.28: Functional Test Case TFU27 - Interface landscape large form factor

Test Name	Interface landscape large form factor	**Test ID**	TFU27
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure interface is clear and unobstructed on a large form factor device in landscape orientation.			
Module / Class	-		
Function	-		
Precondition			
Application running in portrait orientation on a large form factor device (tablet) more than 7 inch diagonal.			
Test Steps			
Review form elements, will need to alter application states to view different perspectives.			
Test pass criteria			
Interface is unobscured, icons are readable, no scrolling or panning required to view form elements and efficient use of space.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.29: Functional Test Case TFU28 - Input Method Editor

Test Name	Input Method Editor	**Test ID**	TFU28
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Make sure that the input method editor (IME) occupies less that 14% of the device screen in the vertical direction.			
Module / Class	-		
Function	-		
Precondition			
Application running as a input method editor, and active input method. Select text field in any application.			
Test Steps			
Cause IME to display and measure the IME height (Vertical).			
Test pass criteria			
IME displays and its measured size (using a ruler placed on the device screen) in the vertical direction is less than 14% of the total screen length in the vertical.			
Outcome			
Not yet implemented			
Action			
Implement critical feature.			
SVN Ver #	514	**Test Date**	9/12/2012
Pass/Fail	Fail	**Tester**	Fred

Table D.30: Unit Test Case TUN29 - Correct operation of NullBooleanFieldHelper

Test Name	Correct operation of NullBooleanFieldHelper	**Test ID**	TUN29
Type	Unit	**Object**	
Focus	Validation		
Purpose			
Prove function operates correctly in response to null, true and false input condition.			
Module / Class	BlueReadyDBHelper.java		
Function	NullBooleanFieldHelper		
Precondition			
nil			
Test Steps			
Run automated tests			
Test pass criteria			
Review test log and confirm test for this function passed.			
Outcome			
Action			
Implement test harness.			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.31: Functional Test Case TFU30 - Device Compatibility Review

Test Name	Device Compatibility Review	**Test ID**	TFU30
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Validate application installs and executes on available hardware			
Module / Class	-		
Function	-		
Precondition			
Bluetooth stack within device, Android based, Minimum release version 7, Bluetooth devices nearby and configured to be visible for discovery. Not already paired with device under test.			
Test Steps			
Load application on Sony Ericson Xperia X10, LG P690f and Kogan Agora 10inch; ensure application installs and executes without error, Truncate the database, Initiate a scan, Ensure Bluetooth devices are discovered and populate the spinner.			
Test pass criteria			
Spinner populates with entries for Bluetooth devices currently visible.			
Outcome			
Application operates as expected			
Action			
None			
SVN Ver #	514	**Test Date**	10/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.32: Functional Test Case TFU31 - Android OS Compatibility Survey

Test Name	Android OS Compatibility Survey	**Test ID**	TFU31
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure application operates on all popular OS releases =¿ minimum supported version.			
Module / Class	-		
Function	-		
Precondition			
Install application on all devices available/accessible with the target os versions and mandatory Bluetooth support.			
Test Steps			
Create a checklist for each popularly released version (based on published market share data) and verify installation and operation.			
Test pass criteria			
Application installs and runs on all os versions tested.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.33: Functional Test Case TFU32 - Bluetooth keystroke capture

Test Name	Bluetooth keystroke capture	**Test ID**	TFU32
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure application can capture keystrokes from Bluetooth device.			
Module / Class	-		
Function	-		
Precondition			
Connected to Bluetooth device.			
Test Steps			
Press keys on the connected Bluetooth device.			
Test pass criteria			
Key information is displayed by the application.			
Outcome			
nothing is displayed			
Action			
Find defect and resolve issue failing in test TUN9			
SVN Ver #	514	**Test Date**	9/12/2012
Pass/Fail	Fail	**Tester**	Fred

Table D.34: Functional Test Case TFU33 - Keystroke utilised by user process.

Test Name	Keystroke utilised by user process.	**Test ID**	TFU33
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure application inserts keystroke into operating systems key stroke queue.			
Module / Class	-		
Function	-		
Precondition			
Run application, such as a text editor. Have Test application as the input method.			
Test Steps			
Type keys on Bluetooth device.			
Test pass criteria			
Typed information appears in host application as typed.			
Outcome			
Not yet implemented			
Action			
Implement missing feature.			
SVN Ver #	514	**Test Date**	9/12/2012
Pass/Fail	Fail	**Tester**	Fred

Table D.35: Functional Test Case TFU34 - Test application executes on phone device.

Test Name	Test application executes on phone device.	**Test ID**	TFU34
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure application can execute on mobile phone devices.			
Module / Class	-		
Function	-		
Precondition			
Load application to phone device smaller than 5 inch diagonal.			
Test Steps			
Run application.			
Test pass criteria			
Application runs without error or Force Close issues.			
Outcome			
Pass			
Action			
SVN Ver #	514	**Test Date**	9/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.36: Functional Test Case TFU35 - Test application executes on tablet device.

Test Name	Test application executes on tablet device.	**Test ID**	TFU35
Type	Functional	**Object**	
Focus	Validation		
Purpose			
Ensure application can execute on tablet devices.			
Module / Class	-		
Function	-		
Precondition			
Load application to tablet device larger than 7 inch diagonal.			
Test Steps			
Run application.			
Test pass criteria			
Application runs without error or Force Close issues.			
Outcome			
Pass			
Action			
SVN Ver #	514	**Test Date**	9/12/2012
Pass/Fail	Pass	**Tester**	Fred

Table D.37: Beta Test Case TBE36 - Application Upgrade

Test Name	Application Upgrade	**Test ID**	TBE36
Type	Beta	**Object**	
Focus	Validation		
Purpose			
Ensure application upgrade is smooth from all publicly released versions to this release			
Module / Class	-		
Function	-		
Precondition			
Repeat once with all older publicly released versions, first time load old version to device, populate database through scanning for devices.			
Test Steps			
Upgrade to version under test.			
Test pass criteria			
Application upgrades without error, user data and configuration is retained.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.38: System Test Case TSY37 - Heavy CPU Load

Test Name	Heavy CPU Load	**Test ID**	TSY37
Type	System	**Object**	
Focus	Validation		
Purpose			
Ensure application functions under simulated heavy CPU load.			
Module / Class	-		
Function	-		
Precondition			
Application running with testing application in background causing artificial heavy CPU load.			
Test Steps			
Attempt to use application.			
Test pass criteria			
Application responds within 3 seconds, not force close or errors.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.39: Structural Test Case TST38 - Restart ability after forced termination.

Test Name	Restart ability after forced termination.	**Test ID**	TST38
Type	Structural	**Object**	
Focus	Validation		
Purpose			
Ensure application can gracefully restart after an unexpected termination, e.g. battery removal.			
Module / Class	-		
Function	-		
Precondition			
Application running and in RADIO_ON state.			
Test Steps			
Remove device battery without touching application interface.			
Test pass criteria			
Upon reboot of Android application can execute without errors or force closures.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.40: System Test Case TSY39 - Connected Bluetooth Device turn off

Test Name	Connected Bluetooth Device turn off	**Test ID**	TSY39
Type	System	**Object**	
Focus	Validation		
Purpose			
Ensure application detects event and changes state to RADIO_ON			
Module / Class	-		
Function	-		
Precondition			
Application running and in CONNECTED state.			
Test Steps			
Turn off selected and connected external Bluetooth device. Don't touch application.			
Test pass criteria			
Application automatically transitions back to RADIO_ON			
Outcome			
Not yet implemented			
Action			
Implement			
SVN Ver #	514	**Test Date**	9/12/2012
Pass/Fail	Fail	**Tester**	Fred

Table D.41: System Test Case TSY40 - Low battery behaviour

Test Name	Low battery behaviour	**Test ID**	TSY40
Type	System	**Object**	
Focus	Validation		
Purpose			
Ensure graceful behaviour in response to low battery conditions			
Module / Class	-		
Function	-		
Precondition			
Application running on device with 16+% battery remaining, on battery only. Application state RADIO_ON.			
Test Steps			
Monitor application state transition as battery decreases.			
Test pass criteria			
Application transitions to INACTIVE state automatically between 15% and 10% battery remaining.			
Outcome			
Not yet implemented			
Action			
Implement feature			
SVN Ver #	514	**Test Date**	9/12/2012
Pass/Fail	Fail	**Tester**	Fred

Table D.42: System Test Case TSY41 - Standby behaviour

Test Name	Standby behaviour	**Test ID**	TSY41
Type	System	**Object**	
Focus	Validation		
Purpose			
Establish impact on device battery running time			
Module / Class	-		
Function	-		
Precondition			
Device fully charged, Minimal configuration, no unnecessary third party software loaded, Wi-Fi off, GPS off, Sync off, Backlight minimum. Bluetooth Radio on.			
Test Steps			
Run device on battery only, Time device running time without application running, Allow device to remain in this state till battery warning occurs. Fully charge device and repeat new timing with Application running and in RADIO_ON state, on battery.			
Test pass criteria			
Compare duration of application not running vs. application running, impact to battery life with application running must not reduce running time by more that 10%			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.43: System Test Case TSY42 - Device configuration change

Test Name	Device configuration change	**Test ID**	TSY42
Type	System	**Object**	
Focus	Validation		
Purpose			
Language change			
Module / Class	-		
Function	-		
Precondition			
Run application, exit.			
Test Steps			
Change Android configuration, change language, alter settings, move app to SD and back to internal storage etc. Re Run application			
Test pass criteria			
Application runs without error or Force Close issues.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.44: System Test Case TSY43 - Other app turning Bluetooth off

Test Name	Other app turning Bluetooth off	**Test ID**	TSY43
Type	System	**Object**	
Focus	Validation		
Purpose			
Ensure application detects event and either restores RADIO_ON state or INACTIVE dependant on current state.			
Module / Class	-		
Function	-		
Precondition			
Run application.			
Test Steps			
Get application to RADIO_ON state. Using Android System settings, manually turn of Bluetooth.			
Test pass criteria			
Application state automatically changes to INACTIVE state, after a RESTORE_RADIO state change.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.45: System Test Case TSY44 - Bluetooth device pairing.

Test Name	Bluetooth device pairing.	**Test ID**	TSY44
Type	System	**Object**	
Focus	Validation		
Purpose			
Ensure application can hand hold the pairing process.			
Module / Class	-		
Function	-		
Precondition			
Application running and in RADIO_ON state. Device Bluetooth radio is on.			
Test Steps			
Select unpaired device, click Pair button.			
Test pass criteria			
Device pairing process commences and completes successfully.			
Outcome			
Not yet implemented			
Action			
Implement missing feature.			
SVN Ver #	514	**Test Date**	9/12/2012
Pass/Fail	Fail	**Tester**	Fred

Table D.46: Functional Test Case TFU45 - Responsiveness

Test Name	Responsiveness	**Test ID**	TFU45
Type	Functional	**Object**	
Focus	Responsiveness		
Purpose			
Ensure that the application is responsive to the users interaction.			
Module / Class	-		
Function	-		
Precondition			
Application running.			
Test Steps			
Use application, activate all form elements.			
Test pass criteria			
Form elements respond with sub second response times, state transitions can take longer.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.47: Functional Test Case TFU46 - Usability verification

Test Name	Usability verification	**Test ID**	TFU46
Type	Functional	**Object**	
Focus	Interface		
Purpose			
Ensure the interface is usable to the typical adult user.			
Module / Class	-		
Function	-		
Precondition			
Application running.			
Test Steps			
Visit all tabs and invoke all states to show all form elements.			
Test pass criteria			
By inspection, no touch point (e.g. button) or text displayed is so small it is unusable. All touchable elements must be easy to activate with a large adult finger. All text must be readable at a distance of 30 cm.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Table D.48: Structural Test Case TST47 - Data Security

Test Name	Data Security	**Test ID**	TST47
Type	Structural	**Object**	
Focus	Security		
Purpose			
Inspection of Code and Stored data to ensure sensitive user information is not placed at risk of theft/disclosure.			
Module / Class	-		
Function	-		
Precondition			
Run application, connect to Bluetooth device.			
Test Steps			
Type simulated sensitive information on connected Bluetooth device. Use Debug, File and Database tools to attempt to recover sensitive information.			
Test pass criteria			
Sensitive information cannot be recovered. (Except as intended through display element for keystroke data or passed through to operating system/recipient application.) Security inspection should be limited to application under test, this test does not seek to identify security flaws in third party software.			
Outcome			
Action			
SVN Ver #		**Test Date**	
Pass/Fail		**Tester**	Fred

Index